Path-based Partitioning Methods for 3D Networks-on-Chip with Minimal Adaptive Routing

Masoumeh Ebrahimi, Masoud Daneshtalab, Pasi Liljeberg, Juha Plosila, José Flich, Hannu Tenhunen

Abstract— combining the benefits of 3D ICs and Networks-on-Chip (NoCs) schemes provides a significant performance gain in Chip Multi-Processors (CMPs) architectures. As multicast communication is commonly used in cache coherence protocols for CMPs and in various parallel applications, the performance of these systems can be significantly improved if multicast operations are supported at hardware level. In this paper, we present several partitioning methods for the path-based multicast approach in 3D mesh-based NoCs, each with different levels of efficiency. In addition, we develop novel analytical models for unicast and multicast traffic to explore the efficiency of each approach. In order to distribute the unicast and multicast traffic more efficiently over the network, we propose Minimal and Adaptive Routing (MAR) algorithm for the presented partitioning methods. The analytical and experimental results show that an advantageous method named Recursive Partitioning (RP) outperforms the other approaches. RP recursively partitions the network until all partitions contain a comparable number of switches and thus the multicast traffic is equally distributed among several subsets and the network latency is considerably decreased. The simulation results reveal that the RP method can achieve performance improvement across all workloads while performance can be further improved by utilizing the MAR algorithm. 19% average and 42% maximum latency reduction is obtained on SPLASH-2 and PARSEC benchmarks running on a 64-core CMP.

Index Terms—3D Networks-on-Chip, unicast and multicast communication, partitioning methods, analytical models, adaptive routing algorithm

1. INTRODUCTION

Networks-on-chip (NoCs) have been proposed as a promising solution for designing the interconnect fabric of multi-core systems [1][2]. Planar (2D) chip fabrication technology is facing new challenges in the deep submicron regime [3]. Wire delay and power consumption increase significantly by the usage of global interconnects in 2D designs. To overcome these limitations, technology is moving rapidly towards the concept of 3D ICs where multiple active silicon layers are vertically stacked. The major advantages of 3D NoCs are the considerable reduction in the average wire length and wire delay, resulting in lower power consumption and higher performance [3][4][5].

Unicast and multicast communication can be considered for a NoC. In the unicast communication case, a message is sent from a source node to a single destination node, while in the multicast communication, a message is delivered from one source node to an arbitrary number of destinations. Multicast can be easily implemented with no hardware overhead by assuming a multicast message is replicated and every instance is sent to a particular destination (this is termed unicast-based multicast). However, this implementation is inefficient. This inefficiency arises because sending multiple copies of the same message into the network not only causes a significant amount of traffic, but also introduces a large serialization delay at the injection point. The vast majority of traffic in Multi-Processor Systems-on-Chip (MPSoCs) consists of unicast traffic and most studies have assumed that the traffic is only unicast. Thereby, the concept of unicast communication has been studied extensively in the literature. The proposed unicast protocols are efficient when all injected messages are

unicast. However, if only a small percentage of the total traffic is multicast, the efficiency of the overall system is considerably reduced. Indeed, multicast communication has a large impact on Chip Multi-Processor (CMP) systems performance. The multicast communication is frequently present in many cache coherence protocols (e.g. directory-based protocols, token-based protocols, and Intel QPI protocol [6][7]). For example, around *5%* of total traffic in a SGI-Origin protocol (which is a directory based protocol) consists of multicast messages [7]. In this protocol, message latency can be reduced by *50%*, if multicast is supported in hardware, thus highlighting the importance of hardware-level multicast support. In this paper, we performed some analysis to determine the percentage of multicast messages generated by coherence protocols. We analyzed synthetic and application traces (i.e. SPLASH-2 [8], PARSEC [9][10]) on top of two popular coherence protocols, MESI [11] and token-based MOESI [12] (the detailed system configuration parameters and workloads can be found in Table 3). Based on experimental results, the bulk of the traffic in MESI protocol is generated by unicast messages while token-based MOESI protocol is heavily multicast-based. On account of our analysis, on average, around *10%* of MESI traffic and more than *80%* of token-based MOESI traffic are multicast.

Hardware-based multicast schemes can be broadly classified into path-based [14][15] and tree-based [14] methods. In the tree-based method, a spanning tree is built at the source switch and a multicast message is sent down the tree. The source switch is considered as the root while destinations are the leaves of this tree. The message is replicated along its route at switches and forwarded along multiple outgoing channels reaching to disjoint subsets of destinations [2]. In the path-based multicast method, a source switch prepares a message for delivery to a set of destinations by placing the list of destinations in the header of the message. The message is routed along the path until it reaches the first destination. The message is delivered both to the local core and to the corresponding output channel for continuing the path toward the next destination in the list. By repeating this process, the message is eventually delivered to all specified destinations. A number of studies have shown that path-based methods exhibit superior performance characteristics over tree-based counterparts [16][17]. The path-based approach does not replicate messages within the network, thus not increasing message contention. However, the path visiting all switches can become large. To reduce the length of the multicast path, destinations can be divided into several disjoint subsets at the network interface of the source switch, and then copies of the message are sent across several separate multicast paths with different destination sets [2]. Partitioning methods try to reduce latency and increase the performance via an efficient partitioning of destinations into disjoint subsets [28].

Additionally routing algorithms can be classified into deterministic or adaptive algorithms. A deterministic routing algorithm uses a fixed path for each pair of switches resulting in increased network latency especially in congested networks. In contrast, in adaptive routing, a message is not restricted to a single path while traveling from a source switch to its destination(s). A message may take a different output at a given switch when the other paths are congested. Therefore, adaptive routing algorithms can obtain better performance [18][19][20].

In this paper, we tackle how to efficiently implement routing in 3D mesh-based NoCs, addressing both unicast and multicast traffic. To do this, we present several partitioning methods, named TBP, VBP, and RP, for the path-based multicast approach, each with different levels of efficiency[1]. In Two-Block Partitioning method (TBP), destinations are divided into two groups and a multicast message is responsible to deliver the message to all destinations within each group. This algorithm performs well when the network size is small. However, as the network enlarges, a message may take a long path to deliver the multicast message to all destinations and thus increasing latency. In Vertical-Block Partitioning method (VBP), destinations are divided into more number of groups depending on their vertical columns. This method suggests a better degree of parallelism and lower latency as a message is dedicated to a smaller set of destination switches and thus a shorter path is taken by each message. The main disadvantage of this method is regarding to the creation of unbalanced partitions as a group may contain a large set of switches compared with others. This results in taking long paths by some messages and short paths by others, keeping the multicast latency still high. Recursive Partitioning method (RP) tries to have the comparable number of switches within each partition while keeping the number of messages low. The RP method suggests a lower average latency compared with TBP and VBP methods. To explore the efficiency of each approach, in addition to simulation experiments, we develop novel analytical models for unicast and multicast traffic. The analytical and experimental results show that RP outperforms the other approaches. On top of all partitioning methods, and in order to efficiently distribute the unicast and multicast messages, we design a minimal and adaptive routing algorithm, named MAR, based on the Hamiltonian path for all partitioning methods. The algorithm is simple and does not require any virtual channel for neither unicast nor multicast messages. The main properties of the final approach which is a combination of the RP and MAR methods can be summarized as follows: 1) decreasing the latency of messages by addressing the non-optimal solutions of ordinary partitioning methods; 2) alleviating the traffic congestion by presenting an adaptive routing

[1]It has been published in the proceeding of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS) [21].

algorithm for both unicast and multicast messages; and 3) causing a relatively small area overhead mainly by not using virtual channels for deadlock avoidance and providing a simple implementation of the routing algorithm.

The rest of this paper is organized as follows: Section 2 reviews related work. A brief background on the Hamiltonian path strategy along with the proposed partitioning methods is discussed in Section 3. The minimal adaptive routing is presented in Section 4. The results are given in Section 5 while we summarize and conclude in the last section.

2. Related Work

Due to the fact that the multicast communication is used commonly in various parallel applications, there have been several attempts to improve the performance of multicast communication in 2D NoCs. "Virtual Circuit Tree Multicasting" (VCTM) [6], "Recursive Partitioning Multicast" (RPM) [22], and "Hamiltonian path-based multicast algorithm for NoCs" [15] are three recent works focused on 2D NoCs. VCTM and RPM are tree-based methods and the proposed algorithm in [15] is a path-based method. In VCTM method, when the number of destinations is large, a large number of setup messages must be delivered into the network (before the real multicast message is delivered) which decreases performance significantly. The area overhead of VCTM is relatively high due to maintaining a table at each switch to store the information of a virtual circuit tree. In RPM method, the processing of the header information is complex and performed several times for each multicast message. The common disadvantage of VCTM and RPM method is that a message may hold several channels for extended periods of time to receive all requested output channels, thereby increasing network contention [2]. Finally, both RPM and VCTM methods are based on deterministic algorithms and cannot provide adaptiveness to neither unicast nor multicast messages. A solution to overcome the disadvantages of tree-based multicast is to utilize path-based multicast routing. The authors in [15] present a deadlock-free adaptation of the dual-path multicast algorithm for 2D mesh NoCs and evaluate the performance impact of the proposed method, demonstrating the efficiency of the proposed multicast algorithm. This method provides some degree of adaptiveness for routing unicast/multicast messages. To the best of our knowledge, there has not been any prior study on path-based multicast routing in 3D NoCs. However, some related studies can be found in the multicomputer domain [23][24][25]. An adaptive multicast communication in 3D mesh networks is discussed in [23]. The algorithm is based on an extension of a theory defined in [24] from 2D to 3D mesh networks. The algorithm utilizes the Hamiltonian path but provides adaptiveness and prevents deadlocks by using virtual channels. However, adding virtual channels is costly in NoCs due to increased arbitration complexity and buffering requirements [26]. Two additional methods of unicast/multicast communication in 3D mesh-based networks are presented

in [24] and [25]. The proposed methods are guaranteed to be deadlock-free by the means of the Hamiltonian path. However, the presented algorithms are deterministic and suffer from low performance and their inability to partition the network efficiently.

3. PARTITIONING METHODS

The performance of a multicast operation can be measured in terms of its latency in delivering a message to all its destinations. Multicast latency consists of two components: the startup latency and the network latency. The startup latency (startup-latency; SL) is the time required to break down a multicast message into several messages (each with a different set of destinations), prepare the messages, and start injecting them into the network. The network latency for multicast messages is defined as the time elapsed from the first flit injection into the network to the reception of the last flit in all destinations of the multicast message. Based on that, we define the mean network multicast latency (mean-mul-latency; MML) and the maximum network multicast latency (max-mul-latency; MxML).

As previously commented, partitioning methods help in reducing the network latency component [28]. In particular, these methods divide the network into several logical partitions and assign destinations to different sets, one set for each partition and including destinations that belong to that partition. Smart partitioning methods must balance the sets and reduce the path length within each partition. However, breaking the network into logical partitions may have the following deficiencies: 1- a large number of network partitions will lead to additional latency as more startup messages (SM) will need to be prepared at the source node and this latency is usually high. 2- an unbalanced configuration of partitions will create long paths within the network. In both cases the latency of the multicast operation will be increased.

3.1. Hamiltonian Path

The Hamiltonian path strategy [14] guarantees that the network will be free of deadlocks for both unicast and multicast traffic. As shown in Fig. 1(a), for each switch, a label is assigned from *1* to *N* where *N* is the number of switches in the network. A Hamiltonian path visits all the switches[2], and each switch is visited exactly once. Several Hamiltonian paths can be considered in the mesh topology. In $a \times b \times c$ mesh network, each switch is labeled by an integer value according to its x, y and z coordinates. The following equations show one possibility of assigning the labels, which we utilize in this paper:

$$L(x,y,z) = \{(a \times b \times z) + (a \times y) + (x+1)\} \qquad where\, z: even, y: even$$
$$L(x,y,z) = \{(a \times b \times z) + (a \times y) + (a-x)\} \qquad where\, z: even, y: odd$$
$$L(x,y,z) = \{(a \times b \times z) + (a \times (b-y-1)) + (a-x)\} \qquad where\, z: odd, y: even$$
$$L(x,y,z) = \{(a \times b \times z) + (a \times (b-y-1)) + (x+1)\} \qquad where\, z: odd, y: odd$$

[2] For sake of understanding, we assume the node X is connected to switch X and the labels are for switches, not nodes.

As exhibited in Fig. 1, two directed Hamiltonian paths (or two subnetworks) are constructed by this labeling. The high channel subnetwork starts at switch *1* (Fig. 1(b)), and the low channel subnetwork ends at switch *1* (Fig. 1(c)). In case the label of the destination switch is greater than the label of the source switch, the routing always takes place in the high channel subnetwork (Fig. 1(b)); otherwise it takes place in the low channel subnetwork (Fig. 1(c)). Notice that there are shortcut channels (those drawn in dashed lines) that do not take part in the Hamiltonian path. However, shortcut channels can be used by messages to improve performance. In this case, half of those channels are used for the high channel subnetwork and half for the other subnetwork. Deadlock is prevented as two separate sets of resources are used for each direction and messages never change their directions. Thus, no dependencies are introduced between the two sets.

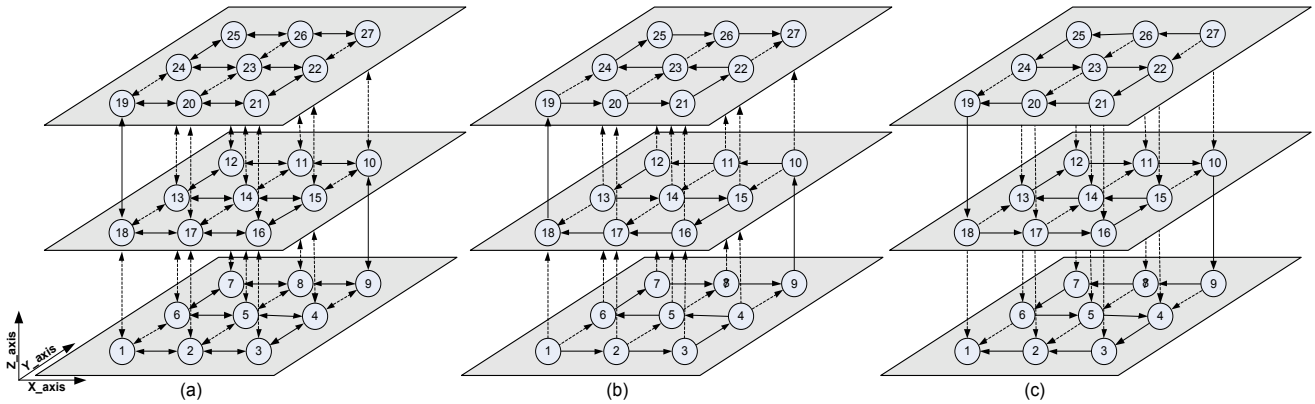

Fig. 1. (a) A *3×3×3* mesh physical network with the label assignment (b) high channel (c) low channel subnetworks. The solid lines indicate the Hamiltonian path and dashed lines indicate the links that could be used to reduce path length in routing.

3.2. Partitioning Methods based on the Hamiltonian Path Strategy

In the partitioning methods, the destinations are grouped in two sets at each source switch. One set includes all the destinations that are reached using the high channel subnetwork, and the other set includes the remaining destinations reached using the low channel subnetwork.

In the next section, we explain the TBP method in detail, and then we introduce two other partitioning methods, VBP and RP. Notice that the TBP method is a straight forward extension of the dual-path multicast in 2D NoCs [15] to 3D NoCs. It can be seen as a naïve method since no effort is made to balance the two sets. For each partitioning method, we provide some analysis on the number of startup messages (SM), latency of multicast operations (MML and MxML), and the average latency of unicast operations (AUL). Analytical models are provided for unicast and multicast messages assuming zero-load latency [5][27]. Based on the zero-load latency, a message never contends for network resources with other messages. Under this assumption, the performance of each approach can be measured based on the number of hops

required for delivering a message from a source node to its destination(s). Contention effects will be investigated both analytically and experimentally in Section 5.

3.2.1. *Two-Block Partitioning (TBP)*

The Two-Block Partitioning (TBP) method is a base scheme in which all switches are split in two disjoint sets: a high set and a low set. As shown in Fig. 2, when considering the label assignment of the Hamiltonian path strategy, all switches located in the same 2D layer as the source switch are distributed between the two sets while all the switches in higher or lower 2D layers are put in the high or low set, respectively. In addition, when multicasting, at maximum one message is created for each set and the destinations within each set are reached according to the Hamiltonian label. Therefore, destinations in the high set are visited in ascending order and destinations in the low set are visited in descending order.

Fig. 2 (a) shows an example of the TBP partitioning policy and the portions of each partition that depends on the source switch position. As illustrated in this figure, if the source switch is located in a middle layer, two partitions cover comparable numbers of switches but still with a large number of switches in both partitions. However in Fig. 2 (b), one partition contains considerably more switches than the other. Now, suppose that the multicast message $m=(7,\{2,3,20,26,45\})$ is generated at switch 7. Destination IDs are split into two sets and should be visited accordingly to their labels: $G_H=\{20,26,45\}$ and $G_L=\{3,2\}$. The message created for G_H uses the Hamiltonian path as follows: {7,10,11,12,13,**20**,21,22,23,**26**,39,42,43,44,**45**} where fourteen hops are needed to reach the last destination. The message path for the G_L is:{7,6,**3**,**2**} where three hops are required for delivering the message to all destinations in the low channel subnetwork. In the TBP method, the number of startup messages is low and never gets larger than two. However, it suffers from high network latency due to unbalanced partitions and high probability of long paths within the network.

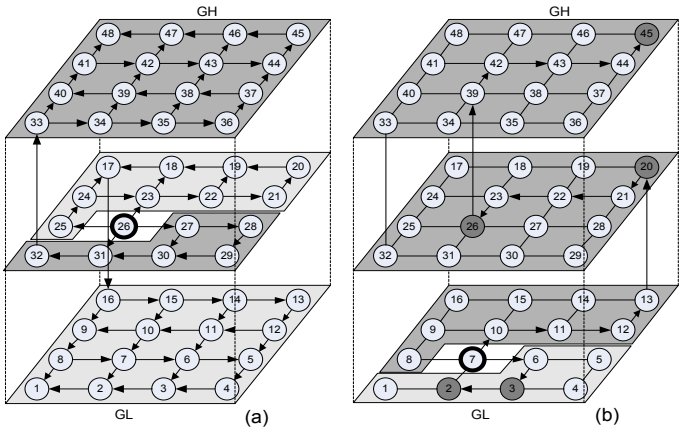

Fig. 2. The TBP method (a) balanced (b) unbalanced partitions.

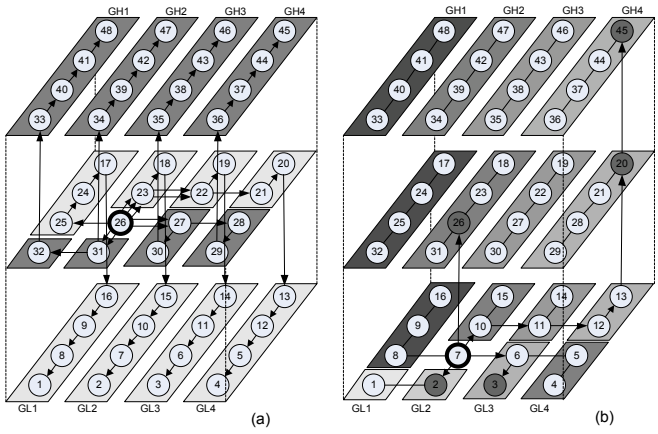

Fig. 3. The VBP method (a) balanced (b) unbalanced partitions.

A. Avg-Uni-Latency (AUL)

Since messages can utilize shortcut channels without introducing new cycles, the path taken by unicast messages is reduced to minimal paths between each pair of source and destinations. Assuming uniform distribution of destinations and using minimal paths for unicast messages, the average unicast latency for a×b×c 3D mesh-based network is [5]:

$$AUL_{3D}=\frac{a^2bc+ab^2c+abc^2-ac-bc-ab}{3abc} \qquad (1)$$

Regardless of the partitioning method used, unicast messages are routed within the network in the same manner, so the formula (1) is valid for the VBP and RP methods. This equation can be easily applied to one-dimensional (when *b=1* and *c=1*) and two-dimensional (when *c=1*) mesh networks.

B. The Startup Latency and Network Latency for Multicast Messages

The multicast latency depends on the number and the location of destinations. This makes computing the analytical multicast latency complex. In order to simplify the complexity, we consider that the latency of a multicast message is set by the final destination so that the multicast message always takes the longest path within the network (without using shortcut channels). This is the worst case. For instance, in the previous example, the two messages have their final destinations set as *45* and *2*, and their distances from the source switch are fourteen and three hops, respectively. However, in MML, we consider the longest path from the source to destinations *45* and *2* which are forty-one and six hops, respectively. In the TBP method, the path between two destinations to reach in a sequential order is minimal while the path from the source to each destination is not necessarily a minimal path. As an example in Fig. 2, the paths from switch *7* to *20*, *20* to *26*, and *26* to *45* are minimal; however, the paths from switch *7* to *26* and *7* to *45* are non-minimal. MML for every switch *j* in *a×b×c* network can be measured by: (Where *n* is the total number of switches in the network.)

$$MML_j=\frac{1}{n}\left(\sum_{i=1}^{i=j-1} i+\sum_{i=j+1}^{i=n}(i-j)\right) \qquad (2)$$

The average multicast latency for the whole network in the TBP method can be obtained by:

$$MML_{TBP}=\frac{1}{n}\sum_{j=1}^{j=n} MML_j=\frac{1}{n}\sum_{j=1}^{j=n}\left(\frac{1}{n}\left(\sum_{i=1}^{i=j-1} i+\sum_{i=j+1}^{i=n}(i-j)\right)\right)=\frac{n^2-1}{3n} \qquad (3)$$

This equation is proved by using the following set of formulas. The sum of partial factorial formula is given by:

$$\frac{m!}{0!}+\frac{(m+1)!}{1!}+\frac{(m+2)!}{2!}+\ldots+\frac{(m+n-1)!}{(n-1)!}=\frac{(m+n)!}{(m+1)(n-1)!} \qquad (4)$$

For all positive integers, we get the formulas (5) and (6) when $m = 1$ and $m = 2$, respectively:

$$1+2+3+\ldots+n=\sum_{i=1}^{i=n} i=\frac{n(n+1)}{2} \qquad (5) \qquad\qquad 1\times2+2\times3+3\times4+\ldots+n\times(n+1)=\sum_{i=1}^{i=n} i(i+1)=\frac{n(n+1)(n+2)}{3} \qquad (6)$$

By using formulas (5) and (6), MML_{TBP} can be written as follow and the equation (3) is proved:

$$MML_{TBP}=\frac{1}{2n^2}\left(\sum_{j=1}^{j=n}(j-1)(j)+\sum_{j=1}^{j=n}(n-j)(n-j+1)\right)=\frac{(n-1)(n+1)}{3n}=\frac{n^2-1}{3n} \qquad (7)$$

MxML is the time when a multicast operation is completed and a message reaches all its destinations. The MxML for a source switch *j* and the whole network are given by:

$$MxML_j=\begin{cases} n-j & \text{if } 0\le j\le \left\lfloor\frac{n}{2}\right\rfloor \\ j-1 & \text{if } \left\lceil\frac{n}{2}\right\rceil\le j\le n \end{cases} \qquad (8) \qquad\qquad MxML_{TBP}=\frac{2}{n}\sum_{j=1}^{j=n/2}(n-j)=\begin{cases} \frac{3n-2}{4} & \text{if n:even} \\ \frac{3n^2-2n-1}{4n} & \text{if n:odd} \end{cases} \qquad (9)$$

In TBP, destinations are split in two sets. Thus, the maximum startup messages (SM) is set to two regardless of the source switch location. There are two exceptions regarding the first and last switches which can deliver only one multicast message to the network.

3.2.2. *Vertical Block Partitioning (VBP)*

In this method, similar to the TBP method, the network is partitioned into high and low channel subnetworks. Destinations are divided into high and low sets. In an additional step, each subnetwork is vertically partitioned such that switches in the same column (with the same *a* value in *a*×*b*×*c* network) are included in a new set.

As illustrated in Fig. 3, this scheme does not guarantee balanced partitions. For the switch located at *26*, partitions are balanced, but they are not balanced when the source is at switch *7* (i.e. four subnetworks cover more switches than the others). Moreover, the time required to prepare at most eight messages is considered as the number of startup messages. For the multicast message *m=(7,{2,3,20,26,45})*, four sets are formed: G_{H2}*={26}*, G_{H4}*={20,45}*, G_{L2}*={2}* and G_{L3}*={3}*. One message is generated for each set and message paths are *{7,**26**},{7,10,11,12,13,**20**,**45**},{7,**2**}* and *{7,6,**3**}* where the maximum hop count is six.

This scheme has several advantages over the TBP method as it achieves a high level of parallelism; avoids the creation of long paths and reduces the network latency. The VBP method increases, however, the number of startup messages as it requires up to *2a* messages in *a*×*b*×*c* network. In addition, this scheme does not guarantee balanced partitions as it is balanced only when the source switch is located in a middle layer while some partitions may cover considerably more switches than the others when the source switch is located at the top or bottom layer.

C. The Startup Latency and Network Latency for Multicast Messages

Since the network is symmetric and is partitioned vertically, the MML value can be measured in one vertical partition and then generalized to other partitions. For this purpose, we consider that $a\times b\times c$ mesh network is divided into a vertical partitions where each partition contains bc switches. Using formulas (2) and (3), the MML value for a source switch j inside a vertical partition and for all switches in a partition can be computed as follow:

$$MML_j=\frac{1}{bc}\left(\sum_{i=1}^{i=j-1} i+\sum_{i=j+1}^{i=bc}(i-j)\right) \quad (10)$$

$$MML_{bc}=\frac{1}{bc}\sum_{j=1}^{j=bc}\frac{1}{bc}\left(\sum_{i=1}^{i=j-1} i+\sum_{i=j+1}^{i=bc}(i-j)\right)=\frac{(bc)^2-1}{3bc} \quad (11)$$

Moreover, messages are required to travel in the x dimension to reach their relative vertical partitions. For example, if $a=4$ in $a\times b\times c$ network and the source switch is located at the first vertical partition, it takes *1*, *2* and *3* hops to reach the second, third and fourth vertical partitions, respectively. This value should be considered when measuring the MML value.

$$MML_a=\frac{1}{a}\sum_{j=1}^{j=a}\frac{1}{a}\left(\sum_{i=1}^{i=j-1} i+\sum_{i=j+1}^{i=a}(i-j)\right)=\frac{a^2-1}{3a} \quad (12)$$

Finally, the MML value for the whole network is given by:

$$MML_{VBP}=MML_a+MML_{bc}=\frac{a^2-1}{3a}+\frac{(bc)^2-1}{3bc}=\frac{a^2bc+ab^2c^2-bc-a}{3abc} \quad (13)$$

From another point of view, the network can be viewed as a 2D network ($a\times b'$) where $b'=b\times c$. The dimension-order routing can be utilized for messages, and thus, by using formula (1) in a 2D network (when $c=1$) the average multicast latency can be measured by:

$$MML_{VBP}=\frac{a^2b'+ab'^2-a-b'}{3ab'}=\frac{a^2bc+ab^2c^2-bc-a}{3abc}$$

In the VBP method, the network is divided into several vertical partitions according to the value a in $a\times b\times c$ network. Thereby, the following formula is used for computing the MxML value in the network.

$$MxML_{VBP}=\frac{2}{n}\sum_{i=1}^{i=n/2}\left(\left\lceil\frac{n-i}{a}\right\rceil+\frac{a^2-1}{3a}\right) \quad (14)$$

The number of partitions in the VBP method depends on the location of switches that result in different startup messages. The switches in the first row of the first layer and the last row of the last layer divide the network into *4*, *5*, *6*, and *7* partitions, while the other switches divide the network into eight partitions. As a result, the average number of startup messages for the VBP method in $a\times b\times c$ network is:

$$SM_{VBP}=\frac{(3a^2-a)+((abc-2a)(2a))}{abc}=\frac{2a^2bc-a^2-a}{abc} \quad (15)$$

3.2.3. *Recursive Partitioning (RP)*

The objective of the recursive partitioning (RP) method is to optimize the number of switches that can be included in a partition and achieve parallelism. In this method, the network is recursively partitioned until each partition contains k switches. In the worst case, the network is partitioned into $2a$ vertical partitions like in the VBP method. We have considered the value k as a reference value indicating the number of switches in each partition of the VBP method, i.e. $(k=bc)$ in $a\times b\times c$ network. An example of the RP method is illustrated in Fig. 4 (a) where a multicast message is generated at the source switch *26*. The required steps of the RP method can be expressed as follows:

Step1: The value k is set to *12* switches in a $4\times4\times3$ network.

Step2: The network is divided into two partitions using the TBP method. The Fig. 2 (a) shows two formed partitions when the source switch is located at switch *26*.

Step3: If the number of switches in a partition exceeds the reference value k, the partition is divided into two new partitions. This step is repeated until all partitions in the network cover at most k switches. Following the example of Fig. 2 (a), *22* switches are covered by the high channel subnetwork which is greater than $k=12$. The high channel subnetwork needs to be further divided into two new partitions (G_{H1} and G_{H2} as shown in Fig. 4 (a)). The G_{H1} and G_{H2} partitions contain *10* and *12* switches, respectively. Since both numbers are less than or equal to $k=12$, no further partitioning is needed for the high channel subnetwork. The same partitioning technique is applied to the low channel subnetwork.

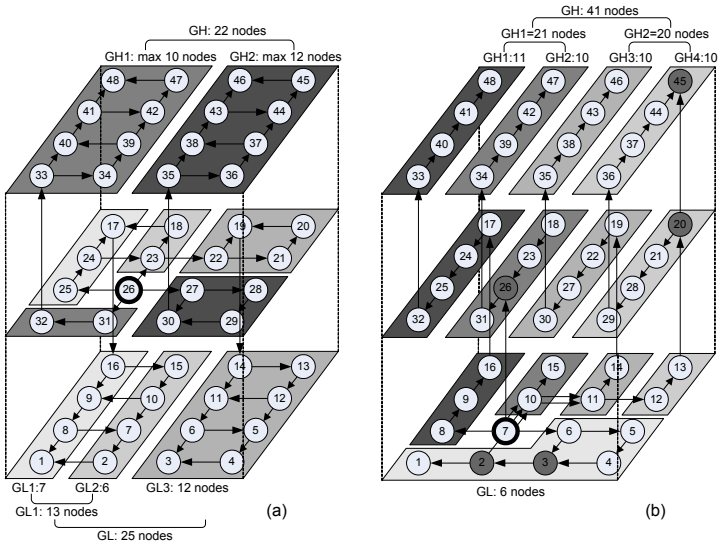

Fig. 4. The RP method when the source is at (a) switch 26 (b) switch 7.

Fig. 4 (b) shows another example of the RP method where the multicast message is $m=(7,\{2,3,20,26,45\})$. In this example three messages are formed and their paths are $\{7,10,11,12,13,\underline{\boldsymbol{20}},\underline{\boldsymbol{45}}\}$,$\{7,\underline{\boldsymbol{26}}\}$, and $\{7,6,\underline{\boldsymbol{3}},\underline{\boldsymbol{2}}\}$ as the maximum latency is six hops.

In brief, this scheme has a similar performance in avoiding long paths as the VBP method while it provides better parallelism as the number of switches is comparable among partitions. By considering the RP method, the creation of balanced partitions is less dependent of the source switch position, and thus it avoids long paths in the network and increases parallelism while keeping the number of startup messages relatively low.

D. The Startup Latency and Network Latency for Multicast Messages

In the RP method, each subnetwork is recursively partitioned until all partitions cover around k switches, where $k=bc$. The next set of formulas is concerned only the high channel subnetwork while the low channel subnetwork has similar formulas. According to this assumption, if the high channel subnetwork covers x switches where $x>k$, it is divided into two new partitions. Each of the formed partitions might still cover more than k switches ($x>k$). Thereby, the partition is further divided into two new partitions. In other words, the MML formula is recursively called until all partitions cover at most k switches. Finally, the average multicast latency is computed when the number of switches x in a partition become less than or equal to the value of k:

$$MML_x=\begin{cases}\dfrac{MML_{\lceil \frac{x}{2} \rceil}+MML_{\lfloor \frac{x}{2} \rfloor}}{2} & \text{where } x>k \\ \dfrac{1}{x}\sum_{i=1}^{i=x} i=\dfrac{x+1}{2} & \text{where } 0<x\leq k \\ 0 & \text{where } x=0\end{cases} \quad (16)$$

Similar to (12), in order to deliver messages from the source switch to different partitions, average multicast latency in the x dimension should be taken into account. Finally, the MML for the RP method is given by:

$$MML_{RP}=\frac{1}{2n}\sum_{i=1}^{i=n}\left(MML_{(i-1)}^{low}+MML_{(n-(i+1))}^{high}\right)+MML_a=\frac{1}{n}\sum_{i=1}^{i=n}MML_{(i-1)}+MML_a \quad (17)$$

For measuring MxML, the number of switches in the biggest partition should be identified. To do this, we first find the MxML value for the high and low channel subnetworks and then determine the number of switches in the biggest partition of the network.

$$MxML_j=Max(MxML_{n-j}^{High} \text{ or } MxML_j^{Low}) \text{ where } MxML_x^{High} \text{ or } MxML_x^{Low}=\begin{cases}Max(MxML_{\lceil \frac{x}{2} \rceil},MxML_{\lfloor \frac{x}{2} \rfloor}) & \text{where } x>k \\ \dfrac{1}{x}\sum_{i=1}^{i=x} i=\dfrac{x+1}{2} & \text{where } 0<x\leq k \\ 0 & \text{where } x=0\end{cases} \quad (18)$$

To compute the MxML value, the following formula is utilized.

$$MxML_{RP}=\frac{2}{n}\sum_{i=1}^{i=n/2}(MxML_i+MML_A) \quad (19)$$

In the case that $x \leq k$, the number of startup messages is equal to *1*. However, when $x>k$, the partition needs to be divided into two new partitions and the SM equation is called for every newly formed partition.

$$SM_x = \begin{cases} SM_{\lceil \frac{x}{2} \rceil} + SM_{\lfloor \frac{x}{2} \rfloor} & \text{when } x>k \\ 1 & \text{when } x \leq k \end{cases} \quad (20)$$

3.3. Hardware Implementation

The micro-architectures of the TBP, VBP, and RP methods are illustrated in Fig. 5 (a), (b), and (c), respectively. In all three methods, the source label is compared (using a comparator) with the destinations labels, so that destinations are divided into high and low channel subnetworks (Fig. 5(a)). In the VBP method, by decoding the value x of the destination address, each destination is placed in one partition as shown in Fig. 5(b). In the RP method, however, the number of switches in the high (or low) channel subnetwork is compared with the reference value k. The result of this comparison determines the required number of partitions such that each can cover about k switches. In the next step, destinations are divided into different partitions (Fig. 5(c)). All procedures are performed in the packetizer unit of the network interface and repeated for every destination in the destination set [29]. Finally, each non-empty register is used in the header of a message. Notice that for encoding the addresses in the message header, we have utilized the bit string scheme [2], where each bit corresponds to a switch in a network. For a set of destinations, the corresponding bits in the bit-string become one.

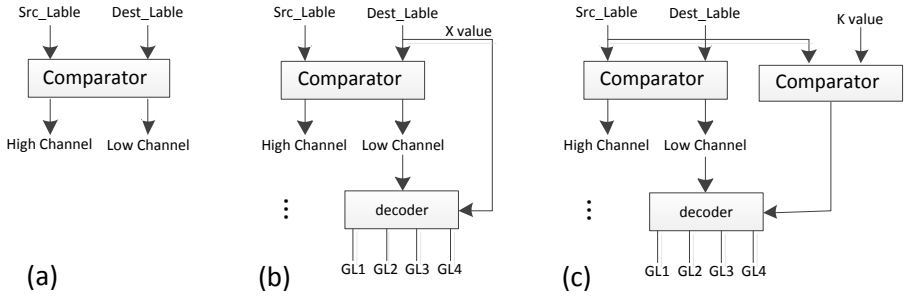

Fig. 5. The micro-architectures of (a) TBP (b) VBP and (c) RP methods

4. Minimal Adaptive Routing (MAR)

In the previous section, we provide different partitioning methods. All of them require a routing algorithm capable of forwarding all the messages to their sets of destinations. In this section, we present a Minimal and Adaptive Routing (MAR) algorithm based on the Hamiltonian path. Using MAR, unicast and multicast messages can be adaptively routed inside the network. The MAR algorithm is implemented at switches and can be described in three steps as follows:

Step1: it determines the neighbors of current switch u that can be used to move a message closer to its destination d. The pseudo code for Step1 is shown in Fig. 6.

Step2: due to the fact that in the Hamiltonian path all switches are visited in ascending order (in the high channel subnetwork) or descending order (in the low channel subnetwork), all of the selected neighbors in Step1 do not necessarily satisfy the ordering constraint. Therefore, if the labels of the selected neighbors (in Step1) are between the label of switch *u* and destination *d*, it/they can be selected as the next hop. The pseudo code for Step2 is shown in Fig. 6.

Step3: since the MAR algorithm provides several choices at each switch, the goal of Step3 is to route a message through the less congested neighboring switch. In the case where the message can be forwarded through multiple neighboring switches, the stress values of the input buffers in the selected neighbors are checked and then the message is sent to the neighbor with the smallest stress value. An example of the MAR algorithm is illustrated in Fig. 7 (a). According to the algorithm, in the first step the neighbors are chosen in a manner that gets the message closer to its destination, i.e. *p={7, 11, 27}*. At the second step, the selected neighbors (in Step1) are checked to determine whether they are in the Hamiltonian path or not. Since the labels of the three selected neighbors are between the labels of the current switch *(u=6)* and destination switch *(d=48)*, the message can be routed via all of them. Suppose that the neighbor *p=11* has a lower stress value than the other neighbors, so the algorithm chooses this neighbor to forward the message. If we continue with the switch *u=11*, this switch has three neighbors belonging to the minimal paths, i.e. *p={10, 14, 22}*. However, only two of them *(p={14, 22})* have the labels greater than the label of the current switch *(u=11)* and lower than the label of the destination switch *(d=48)*. Finally, according to the stress values of the input buffers in the corresponding direction, one of them is selected as the next hop. The algorithm is repeated for the rest of the switches until the message reaches the final destination. Fig. 7 (b) shows all possible shortest paths from the source switch *(u=6)* to the destination switch *(d=48)*. It is worth noting that the stress value is updated whenever a new flit enters or leaves the buffer (flit events: flit_tx or flit_rx). That is, in each flit event, if the number of occupied cells of the input buffer is larger (smaller) than a threshold value, the threshold signal is assigned to one (zero).

The MAR algorithm can be adapted for multicast messages such that alternative paths are used to route a message between the source switch and the first destination and also between successive destinations. An example is shown in Fig. 7 (c) where the source *(u=6)* forwards a multicast message towards its destinations *(D={15, 32, 46})*. The MAR algorithm provides a set of alternative paths to send a message from the source switch to the first destination *(d1=15)*. Similarly, the message can be adaptively routed between each two destinations.

```
Algorithm: Minimal Adaptive Routing (MAR_3D)
Inputs:    current switch label, destination switch label,
           neighboring switches Labels
Begin
            ----------STEP 1----------
   X_dir = East when (x_c<x_d) else West;
   Y_dir = North when (y_c<y_d) else South;
   Z_dir = High when (z_c<z_d) else Low;
Process     ----------STEP 2----------
   Begin
      If ((Label(CurrentSwitch) = Label(DestSwitch)) then
          Select Local;
      Elsif ((Label(CurrentSwitch) < Label(DestSwitch)) then
----------High Channel Subnetwork---------
          If (Label(CurrentSwitch) < Label(Neighbor(X_dir))) and
             (Label(Neighbor(X_dir)) < Label(DestSwitch)) then
                First Choice -> Neighbor(X_dir)
          End if;
          If (Label(CurrentSwitch) < Label(Neighbor(Y_dir))) and
             (Label(Neighbor(Y_dir)) < Label(DestSwitch)) then
                Second Choice -> Neighbor(Y_dir)
          End if;
          If (Label(CurrentSwitch) < Label(Neighbor(Z_dir))) and
             (Label(Neighbor(Z_dir)) < Label(DestSwitch)) then
                Third Choice -> Neighbor(Z_dir)
          End if;
      Elsif ((Label(CurrentSwitch) >Label(DestSwitch)) then
----------Low Channel Subnetwork---------
          If (Label(CurrentSwitch) >Label(Neighbor(X_dir))) and
             (Label(Neighbor(X_dir)) > Label(DestSwitch)) then
                First Choice -> Neighbor(X_dir)
          End if;
          If (Label(CurrentSwitch) > Label(Neighbor(Y_dir))) and
             (Label(Neighbor(Y_dir)) > Label(DestSwitch)) then
                Second Choice -> Neighbor(Y_dir)
          End if;
          If (Label(CurrentSwitch) >Label(Neighbor(Z_dir))) and
             (Label(Neighbor(Z_dir)) > Label(DestSwitch)) then
                Third Choice -> Neighbor(Z_dir)
          End if;
    End If;
End Process;
```

Fig. 6. The pseudo code of the MAR algorithm

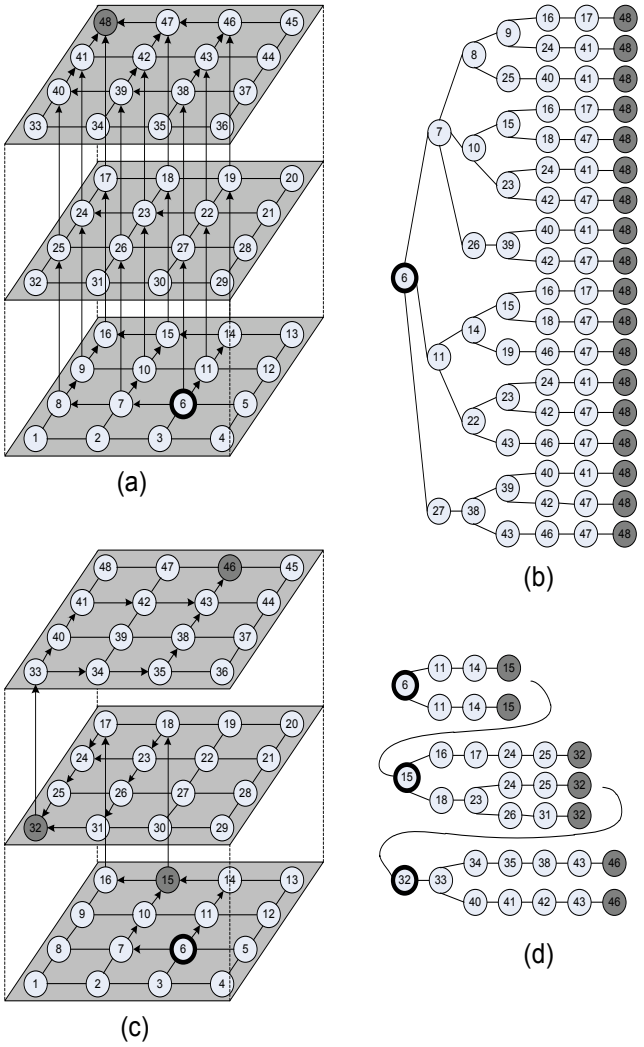

Fig. 7. Example of the MAR algorithm for unicast and multicast messages

For example, at switch *15*, the message can make progress towards destination *32* either by selecting switch *18* in the next layer or switch *16* in the current layer. The MAR algorithm is compatible with all methods supporting the Hamiltonian path in 2D or 3D NoCs. Therefore, all the TBP, VBP and RP methods can utilize the MAR algorithm for both unicast and multicast messages. Fig. 7 (d) shows all possible shortest paths from the source switch *(u=6)* to the destinations 15, 32, and 46.

To show that the proposed algorithm is deadlock free, we need to prove that the channel dependency graph (CDG) is acyclic [28]. To close a cycle in the high channel subnetwork, a message may require requesting a channel that forwards the message to a lower-labeled switch, which is not allowed by the MAR algorithm. The same applies for the low channel subnetwork. Since both in unicast and multicast traffic, messages are routed only in ascending and descending order, the MAR algorithm is deadlock-free. However, in multicast traffic there is a possibility of deadlock in the consumption chan-

nels [28]. This happens when a message should be delivered both to the local core and the output channel (to move towards the next destination). This may cause deadlock if two multicast messages reach the switch and request both channels, but each gets access to only one channel. There is a branch dependency that creates a deadlock situation. This can be solved basically using extra resources between the local core and the corresponding switch to avoid such conflict. In our case, we implement at each switch two ejection channels.

5. Results and Discussion

5.1. Analytical Results

We analyzed and compared the unicast latency, the startup latency, and the network latency of the TBP, VBP, and RP partitioning methods using analytical models. For this purpose, the previously presented factors (SM, MML, and MxML) are utilized. For each method, we explore the values for two different network sizes along with two different numbers of destinations, injection rates, and message lengths. Finally, the total latency is estimated under different configurations and methods.

5.1.1. Startup Latency

We developed formulas to extract the number of startup messages (SM) of the TBP, VBP and RP methods. However, the startup latency not only depends on the SM value but also it is affected by the message length, injection rate, and the number of destinations per multicast message.

A. The Impact of the Number of Destinations on the Startup Latency

We computed the upper-bound value of SM for the TBP, VBP, and RP methods by assuming that there is one message per partition. The 3^{rd} column of Table 1 shows the maximum number of startup messages in the TBP, VBP, and RP methods. However, in reality, the number of messages may be smaller than the number of partitions (e.g. when the number of destinations is lower than the sets or destinations are not evenly distributed among sets). We have assumed uniform distribution and used conditional probabilities to find out the probability that a partition has received a message when there are eight or sixteen destinations per message. Based on this evaluation, the 4^{th} and 7^{th} columns in Table 1 are filled. For example, when there are eight partitions and eight destinations per message, on average, five partitions include at least a destination and three partitions are empty, thus the average of startup messages is five. As the number of destinations per message increases (e.g. from *8* to *16* destinations), with a high probability there is at least one destination per partition. In this

case, the startup messages almost reach the upper-bound values. According to the values in Table 1, the RP method offers a lower startup messages than the VBP method since some partitions are merged together.

The average unicast latency for different network sizes is listed in the *2nd* column. As already mentioned, the unicast latencies of different methods are similar. This is because of the fact that unicast messages are routed similarly in the network using the TBP, VBP, and RP methods. Obviously, the unicast latency is increased as the network scales up.

Table 1. Unicast latency, startup messages for different number of destinations, message lengths, and injection rates.
UL: Unicast Latency; SM: Startup Messages; SL: Startup Latency; D/M: Destination per Message; F/M: Flit per Message; R: Rate.

Method	**1st** Size	**2nd** UL (hop)	**3rd** SM	**4th** SM 8 D/M 1 F/M	**5th** SL 8 D/M 5 F/M 1% R 1st M	**6th** SL 8 D/M 5 F/M 10% R 100th M	**7th** SM 16 D/M 1 F/M	**8th** SL 16 D/M 10 F/M 1% R 1st M	**9th** SL 16 D/M 10 F/M, 10% R 100th M
TBP	4×4×4	3,75	2	2	5	5	2	10	10
TBP	8×8×8	7,88	2	2	5	5	2	10	10
VBP	4×4×4	3,75	8	5	20	20	7	60	60
VBP	8×8×8	7,88	16	6	25	25	11	100	100+990
RP	4×4×4	3,75	5	4	15	15	5	40	40
RP	8×8×8	7,88	10	5	20	20	8	70	70

B. The Impact of Message Length on the Startup Latency

To show the impact of the message length on the startup latency, let us assume that a multicast message includes all destinations, and thus only one message is sent to the network. In the TBP method and when there is no contention in the network, the first flit of the message *1* (mul-msg1) enters the network at cycle *0* while the message *2* (mul-msg2) can start sending its first flit at cycle *N*, where *N* is the number of flits per message (Fig. 8(a)). By partitioning the network in the VBP and TP methods, the destinations are distributed among several sets. In this case, multiple copies of the message *1* (with different sets of destinations) are injected into the network at cycles *0, N, 2N, …* (Fig. 8(b)). The message *2* can deliver its first flit as soon as all copies of the message *1* are delivered into the network. The startup latency is computed by considering the average message length as follow:

$$SL_A=(\text{startup messages}-1)*(\text{flits per message})$$

In Table 1, the *4th* and *5th* columns indicate the differences between the startup latencies when the message size increases from one to five flits. Similarly, the *7th* and *8th* columns show the startup latencies by changing the message size from one to ten flits. The values show an increased in the startup latencies when the message size increases. In all configurations, the TBP method has the lowest startup latency, and then the RP and VBP methods, respectively.

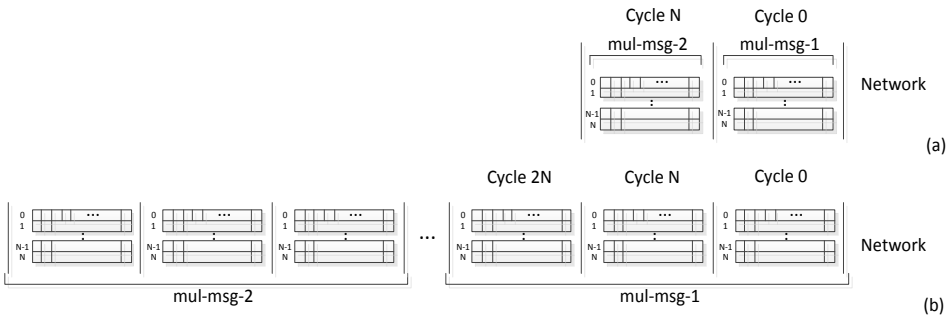

Fig. 8. The Impact of Message Length on the Startup Latency.

C. The Impact of the Injection Rate on the Startup Latency

In a low injection rate, the message *2* is probably generated by the core when all messages of the message *1* have already sent to the network. However, in a case of high injection rate, the message *2* is ready to be sent to the network while the messages of message *1* are still in the queue and have not completely delivered to the network. Therefore, if the number of cycles required for delivering all the messages of a multicast message is larger than (*100 - rate%*), the following formula is obtained: (Latency is cumulative, with each additional generated message)

$$SL_B = SL_A + (\text{total number of generated messages} - 1) * (SL_A - (100 - \text{rate}\%))$$

The Table 1 also includes the results when the injection rate takes into consideration. The values are obtained based on two injection rates, *1%* and *10%*. As can be seen in the 5^{th} and 6^{th} columns (or 8^{th} and 9^{th} columns), in most cases, the startup latencies do not change as the message *1* has delivered all its messages before the message *2* is generated. However, in a high injection rate (i.e. *10%*), the time required to send the startup messages may exceed *100-10=90* cycles. As shown in the 9^{th} column, in one case, it takes more than *90* cycles to deliver startup messages completely to the network. Indeed, the newly generated messages experience considerably larger delays to send their first flit into the network. The values in the 6^{th} and 9^{th} columns are computed for the 100^{th} message, while in 5^{th} and 8^{th} columns are measured for the first message.

5.1.2. Network Latency

Using analytical formulas, we have estimated the MxML and MML values for TBP, VBP, and RP methods in *4×4×4* and *8×8×8* networks. Since MxML and MML reveal the number of hops, to estimate the network latency, the switch delay should also be taken into consideration. By assuming 3-stage pipeline architecture, the network latency is computed by multiplying the number of hops and a factor of three. On the other hand, as the injection rate and contention increases, per-hop delay is increased. We assume that in a *10%* injection rate, the latency is six cycles per hop. According to this assumption, we estimate the total latency using the following formula:

$$Total\ Latency = \begin{cases} MML * 3 + SM & with\ 1\%\ injection\ rate \\ MML * 6 + SM & with\ 10\%\ injection\ rate \end{cases}$$

The values in 2nd and 3rd column of Table 2 indicate that MxML and MML of the TBP method are considerably larger than those of values in the VBP and RP methods. The VBP method can reduce the MML value significantly at a cost of more startup messages. 4th, 5th, 6th, and 7th columns show the total latency values when the startup latency takes into consideration. Since, the high number of startup messages in the VBP method may result in a time-overlapping of different messages, as can be seen in the last column, in some cases the VBP method even behave worse than the TBP method.

Table 2. MML, MXML, and total latency in TBP, VBP, and RP methods.

Method	1st Size	2nd MxML	3rd MML	4th MML*3+SL 5flits,8dests, 1%rate, 1th message	5th MML*6+SL 5flits,8dest, 10% rate, 100th message	6th MML*3+SL 10flits,16dests, 1%rate, 1th message	7th MML*6+SL 10flits,16dest, 10% rate, 100th message
TBP	4×4×4	48	21	68	131	73	136
TBP	8×8×8	384	171	518	1031	523	1036
VBP	4×4×4	14	6	38	56	78	96
VBP	8×8×8	51	24	97	169	172	1234
RP	4×4×4	15	7	36	57	61	82
RP	8×8×8	59	26	98	176	148	226

5.2. Simulation Results

To assess the efficiency of the proposed partitioning methods in experiment, we have developed a cycle-accurate NoC simulator based on wormhole switching in 3D mesh configuration. The simulator inputs include the array size, the routing algorithm, the link width, the buffer size, and the traffic type. The on-chip network, considered for experiment is formed by a typical wormhole-switching structure including input buffers, a routing unit, a switch allocator, and a crossbar. Each switch has *7* input/output ports, a natural extension from a *5*-port 2D switch by adding two ports to make connections to the upper and lower layers [23][31]. There are some other types of 3D switches such as the hybrid switch [3][31] and MIRA [32], however, since switch efficiency is out of the goals of this paper, we have chosen a simple *7*-port switch in our simulation. The arbitration scheme of the switch allocator in the typical switch structure is round-robin. The data width and the frequency were set to *64* bits and *1GHz*, respectively, and each input channel has a buffer size of five flits with the congestion threshold at *80%* of the total buffer capacity. This congestion threshold is utilized by the presented MAR algorithm to choose the less congested path if there would be any alternative path(s). The experiments were performed on a *48*-switch (*4×4×3*) 3D stacked architecture with a constant message size of five flits. For the performance metric, we used the multicast latency defined as the number of cycles between the initiation of a multicast message operation, including preparation and startup latency, and the time when the tail of the multicast message reaches all the destinations. For each load value, the result of message latency is averaged over *80,000* messages after a warm-up session of *20,000* arrived messages.

5.2.1. Performance Evaluation

A. Multicast Traffic Profile

The first set of simulations was performed for a random traffic profile. A uniform distribution is used to construct the destination set of each multicast message [14]. The number of destinations has been set to eight or sixteen. The average communication delay as a function of the average message injection rate has been shown in Fig. 9. As observed from the results, the RP method meets lower delay than the TBP and VBP methods. The foremost reason for this performance gain is due to the efficiency of the RP method which not only reduces the number of hops for multicast messages but also the number of startup messages. In fact, TBP suffers from long paths while the performance of VBP degrades due to a large number of startup messages. Adaptive routing algorithms obtain better performance in congested networks due to using alternative routing paths [18]. In Fig. 10, ARP (Adaptive RP, utilizing MAR in RP), and AVBP (Adaptive VBP, utilizing MAR in VBP), are the adaptive models of the RP and VBP methods, respectively. As illustrated in this figure, adaptive routings become more advantageous when the injection rate increases.

B. Unicast and Multicast (Mixed) Traffic Profile

In this set of simulations, we used a mixture of unicast and multicast traffic, where *70%* of injected messages are unicast messages and the remaining *30%* are multicast messages. Hotspot and transpose traffic model profiles have been taken into account for unicast traffic generation. Under the hotspot traffic pattern, one or more switches are chosen as hotspots receiving an extra portion of the traffic in addition to the regular uniform traffic. In the hotspot traffic model, given a hotspot percentage of h, a newly generated message is directed to each hotspot switch with an additional h percent probability. We simulate hotspot traffic with a single hotspot switch. The hotspot switch is chosen to be the switch *(2,2,2)* in the *4×4×3* mesh network. Fig. 11 shows the performance with $h = 10\%$.

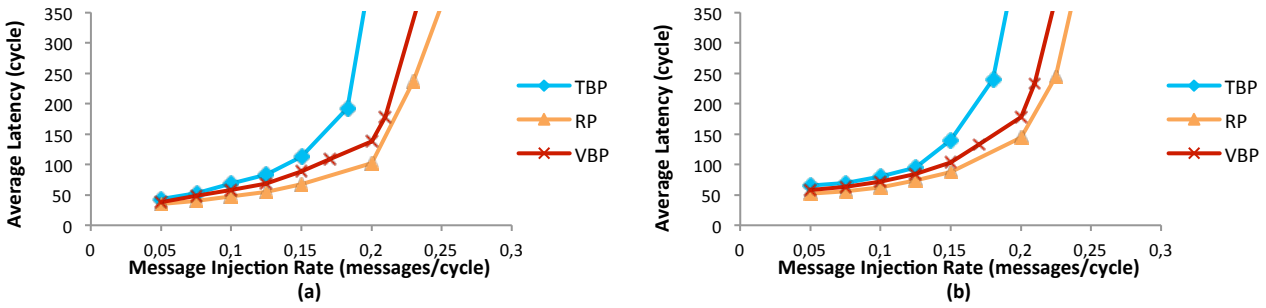

Fig. 9. Performance under different loads in *4×4×3* 3D mesh using deterministic routing with (a) *8* destinations, (b) *16* destinations

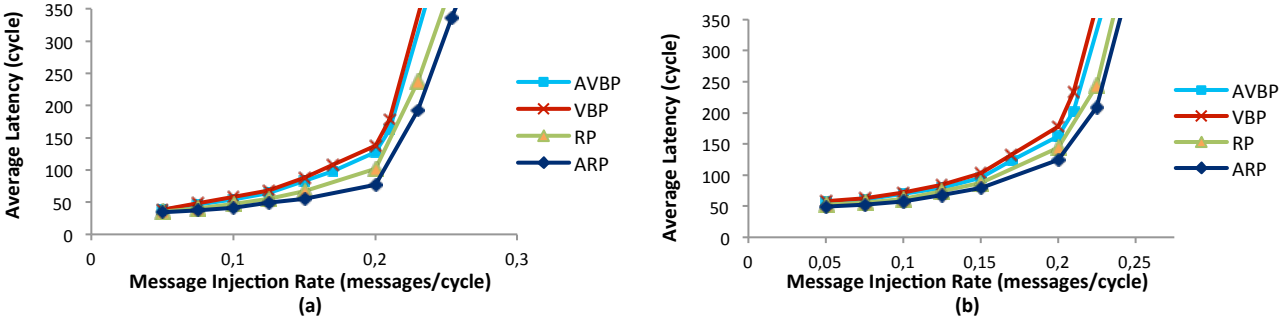

Fig. 10. Performance under different loads in *4×4×3* 3D mesh using adaptive routing with (a) *8* destinations, (b) *16* destinations

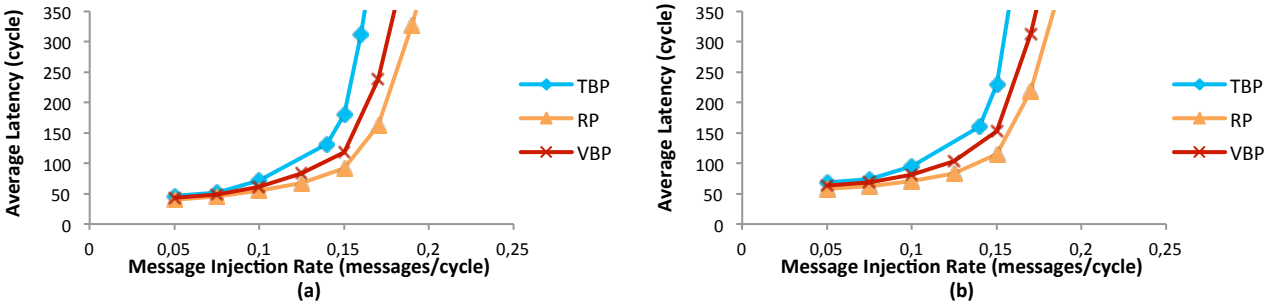

Fig. 11. Performance under different loads in *4×4×3* 3D mesh using deterministic routing with (a) *8* destinations, (b) *16* destinations under mixed traffic (*30%* multicast and *70%* unicast); Unicast traffic is based on the hotspot traffic model with a single hotspot switch *(2,2,2)*, and *h=10%*.

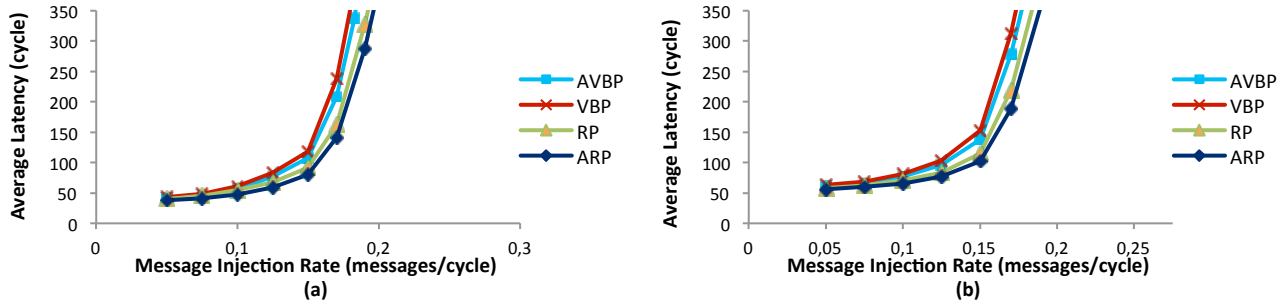

Fig. 12. Performance under different loads in *4×4×3* 3D mesh using adaptive routing with (a) *8* destinations, (b) *16* destinations under mixed traffic (*30%* multicast and *70%* unicast); Unicast traffic is based on the hotspot traffic model with a single hotspot switch *(2,2,2)*, and *h=10%*.

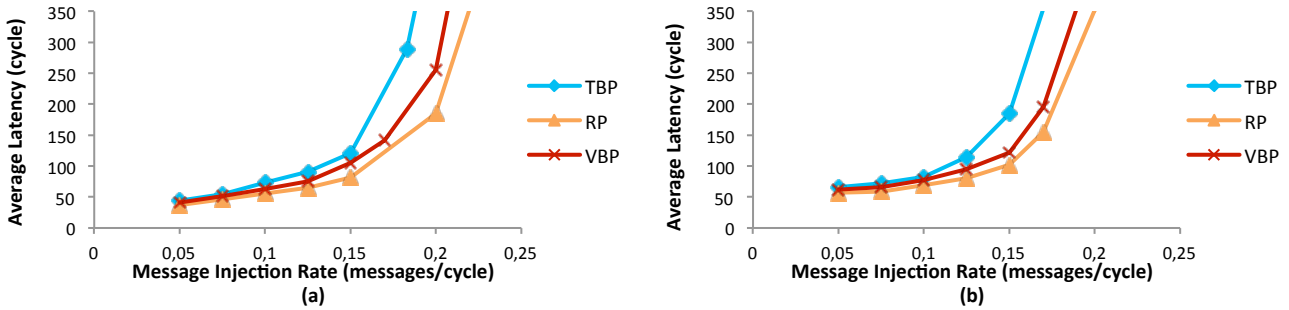

Fig. 13. Performance under different loads in *4×4×3* 3D mesh using deterministic routing with (a) *8* destinations, (b) *16* destinations under mixed traffic (*30%* multicast and *70%* unicast); Unicast traffic is based on the transpose traffic model.

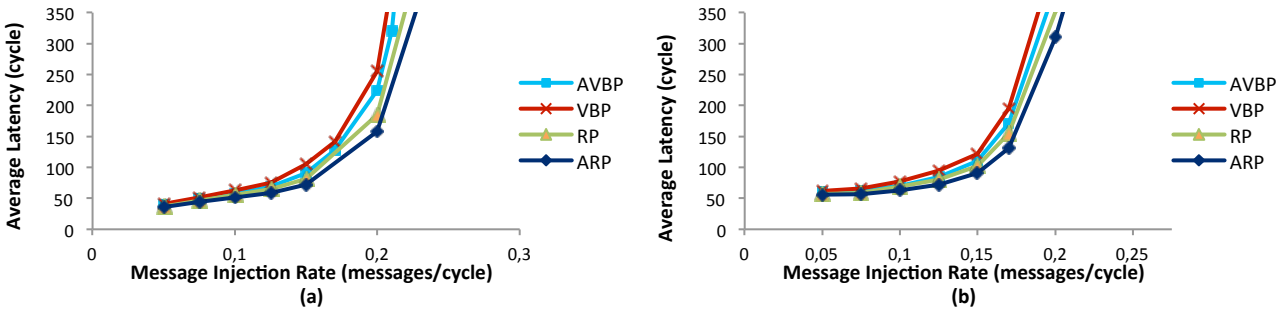

Fig. 14. Performance under different loads in *4×4×3* 3D mesh using adaptive routing with (a) *8* destinations, (b) *16* destinations under mixed traffic (*30%* multicast and *70%* unicast); Unicast traffic is based on the transpose traffic model.

Under the transpose traffic pattern, the source switch positioned at (x, y, z) sends messages to the destination switch (a−1−x, b −1−y, c−1−z) for all x Є {0, …, a − 1}, y Є {0, …, b − 1}, z Є {0, …, c −1}, in $a \times b \times c$ mesh network. As illustrated in Fig. 11 and Fig. 13, the RP method outperforms the two other partitioning methods under both traffic profiles when using a deterministic routing algorithm. This improvement is achieved through using optimized partitions formed by the RP method. Moreover, Fig. 12 and Fig. 14 show the average latency when utilizing the MAR routing algorithm. Based on the presented partitioning methods, the adaptive routing reduces the average latency in comparison with the deterministic routing.

C. Application Traffic Profile

The GEMS full system simulator [13] is used as our simulation platform coupled with a cycle-accurate 3D NoC model. In order to know the real impact of the presented methods, we used traces from some application benchmark suites selected from SPLASH-2 [8], and PARSEC [9][10]. Simulations are run on the Solaris 9 operating system based on SPARC instruction architecture. The adopted mapping strategy used in Solaris 9 is arbitrary mapping. Table 3 summarizes our full system configuration where the cache coherence protocol is token-based MOESI and access latency to the L2 cache is derived from the CACTI [33]. We form a *64*-node on-chip network (*4×4×4*) that four layers are stacked on top of each other, i.e. out of the *64* nodes, *16* nodes are processors and other *48* nodes are *L2* caches. *L2* caches are distributed in the bottom three layers, while all the processors are placed in the top layer close to a heat sink so that the best heat dissipation capability is achieved [32][34]. For the processors, we assume a core similar to Sun Niagara and use SPARC ISA [35]. The memory hierarchy implemented is governed by a two-level directory cache coherence protocol. Each processor has a private write-back *L1* cache (split *L1 I* and *D* cache, *64KB*, *2*-way, *3*-cycle access). The *L2* cache is shared among all processors and split into banks (*48* banks, *1MB* each for a total of *48MB*, *6*-cycle bank access), connected via on-chip switches. The *L1*/*L2* block size is *64B*. The simulated memory hierarchy mimics SNUCA [36] while the off-chip memory is a *4GB* DRAM with a *260*-cycle

access time. The simulator produces, as output, the communication latency for cache access. Fig. 15 shows the average network latency of the real workload traces collected from the aforementioned system configurations, normalized to TBP. However, using the adaptive routing scheme, MAR, diminishes the average delay of each partitioning method significantly under all benchmarks. That is, adaptive routing has an opportunity to improve performance. For instance, under the *fft* application, the performance gain of using MAR in TBP, RP, and VBP is about (ATBP/TBP) *7%*, (AVBP/VBP) *11.5%*, and (ARP/RP) *6%*. We can see that ARP consistently reduces the average network latency across all tested benchmarks. Table 4 lists the performance gains of ARP over TBP, ATBP, RP, VBP, and AVBP where the overall performance gain is about *19%*.

Table 3. System configuration parameters.

Processor Configuration	
Instruction set	SPARC, 16 processors
L1 cache	16KB. 4-way associative, 64-bit line, 3-cycle access time
L2 cache	Shared, distributed in 3 layers, unified, 48MB (48 banks, each 1MB). 64-bitline, 6-Clock
Cache coherence protocol	MESI, Token-based MOESI
Cache hierarchy	SNUCA
Size	4GB DRAM
Access latency	260 cycles
Requests per processor	16 outstanding
Benchmarks	SPLASH-2, PARSEC
Network configuration	
switch scheme	3D mesh with wormhole
Flit size	64 bits
Workloads	
SPLASH-2	Barnes, Cholesky, FFT, LU, Ocean, Radix, Raytrace, Water-Nsq
PARSEC	x264

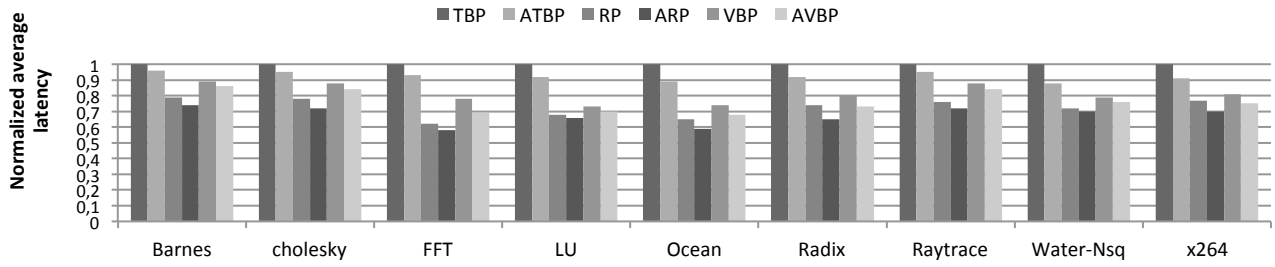

Fig. 15 Performance under different application benchmarks normalized to TBP.

Table 4. Performance gain of ARP over other presented schemes.

	TBP	ATBP	RP	VBP	AVBP	
Barnes	26%	23%	6%	16%	14%	
Cholesky	28%	24%	7%	18%	14%	
FFT	42%	37%	6%	25%	15%	
LU	34%	28%	3%	9%	6%	
Ocean	41%	33%	9%	20%	13%	
Radix	35%	29%	12%	18%	10%	
Raytrace	28%	24%	5%	17%	14%	
Water-Nsq	30%	20%	3%	11%	7%	
x264	30%	23%	9%	13%	6%	**Overall**
Avg.	32%	27%	7%	17%	11%	19%

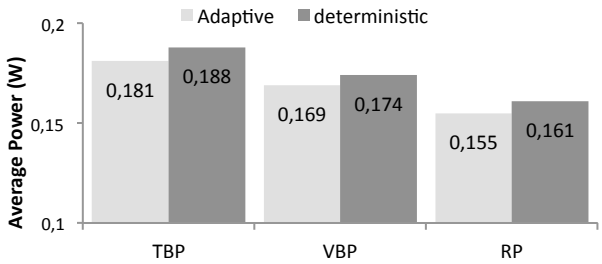

Fig. 16. Average power dissipation results in *4×4×3* 3D mesh under multicast traffic profile.

5.2.2. Hardware Overhead

The presented partitioning methods have been implemented in network interfaces, thereby, to estimate the hardware cost of the proposed methods, the network area of each partitioning scheme, including switches and network interfaces, with the aforementioned configuration were synthesized by Synopsys D.C. using the UMC *90nm* technology with an operating point of *1GHz* and supply voltage of *1V*. We performed place-and-route, using Cadence Encounter, to have precise power and area estimations. Depending on the technology and manufacturing process, the pitches of TSVs can range from $1\mu m$ to $10\mu m$ square [37]. In this work, the pad size for TSVs is assumed to be $5\mu m^2$ with pitch of around $8\mu m^2$. The area of two-unidirectional vertical channels, 2D switch, and 3D switch are $0.01mm^2$, $0.18mm^2$, and $0.23mm^2$, respectively, by considering the link width of *64* bits. Therefore, the overhead of adding TSVs in a 3D switch is less than *4%*. Different numbers of registers were employed for TBP (the base method), VBP, and RP methods to implement their partitioning mechanisms in network interfaces, leading to different values of area overhead. Comparing the area cost of the TBP with VBP and RP schemes indicates *5%* and *6%* additional overhead, respectively. All partitioning methods use the same routing unit, and thus the differences in area overhead values are related to the implementation of different methods in the network interfaces. It is worth mentioning that the area overhead of the network interface unit alone in the TBP method is about $0.0419\ mm^2$. The proposed adaptive routing unit (MAR) imposes less than *0.5%* overhead on a switch in each method and it is independent of the network size.

5.2.3. Power Dissipation

The power dissipation of the TBP, VBP, and RP methods were calculated and compared under the multicast traffic model with sixteen destinations using the simulator based on the Orion [30] and the equation in [5]. The power values of the network interfaces, computed after the place-and-route in the previous subsection, have been also integrated in the Orion functions. The typical clock of *1GHz* is applied in the aforementioned network (*4×4×3* 3D mesh network). The results for the average power under multicast traffic are shown in Fig. 16.

The average power values are computed near the saturation point, *0.16* (messages/cycle), under multicast traffic. As the results show, the average power consumption of the RP scheme is *16%* and *8%* less than that of the TBP and VBP schemes, respectively, when using deterministic routing. In fact, this is achieved by smoothly balancing the traffic over the network using efficient balancing scheme which reduces the number of the hotspots and, hence, lowering the average power.

6. SUMMARY AND CONCLUSION

In this paper, we first presented a set of partitioning methods for 3D mesh-based NoCs along with their analytical models. Among them, the recursive partitioning method achieves higher performance. This method partitions the network recursively until all partitions contain comparable numbers of switches. Experimental results show that the recursive partitioning method reduces the transmission delay and provides a high degree of parallelism compared with the two other methods, TBP and VBP. The paper continued by presenting an adaptive routing algorithm for both unicast and multicast traffic in 3D mesh-based NoCs. The presented algorithm can add adaptivity to the network by taking advantage of the Hamiltonian path strategy without using virtual channels. Using SPLASH-2 and PARSEC benchmarks, the performance gain of the RP method is about *17%* and *27%*, compared with the TBP and VBP methods, respectively, while reducing the power consumption, *12%* on average.

References

[1] A. Jantsch and H. Tenhunen, “Networks on Chip”, New York: Kluwer, 2003.
[2] J. Duato, S. Yalamanchili, L.M. Ni, “Interconnection networks: an engineering approach”, Morgan Kaufmann Publishers, 2003.
[3] B.S. Feero, P.P. Pande, “Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation”, IEEE Transactions on Computers, v. 58, no. 1, pp. 32-45, 2009.
[4] M. Daneshtalab, M. Ebrahimi, J. Plosila, "HIBS-Novel Inter-layer Bus Structure for Stacked Architectures," in Proceedings of IEEE International 3D Systems Integration Conference (3DIC), pp. 1-7, Jan. 2012, Japan.
[5] V.F. Pavlidis and E.G. Friedman, “3-D topologies for networks-onchip”, IEEE Transactions on Very Large Scale Integration Systems, v.15, i.10, pp.1081-1090, 2007.
[6] N.E. Jerger, L.S. Peh, M. Lipasti, “Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support”, 35th Int. Symp. on Computer Architecture (ISCA), pp. 229-240, 2008.
[7] P. Abad et al., “MRR: Enabling fully adaptive multicast routing for CMP interconnection networks”, IEEE 15th Int. Symp. on High Performance Computer Architecture (HPCA), pp. 355-366, 2009.
[8] S.C. Woo et al., “The splash- 2 programs: Characterization and methodological considerations”, in Proc. of the 22nd Int. Symp. on Computer Architecture, pp. 24-36, 1995.
[9] C. Bienia, S. Kumar, J.P. Singh, and K. Li, “The parsec benchmark suite: characterization and architectural implications”, in Proc. of the 17th Int. Conf. on Parallel architectures and compilation techniques, pp. 72-81,2008.
[10] C. Bienia, S. Kumar, and K. Li, “Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chipmultiprocessors”, in IEEE Int. Symp. on Workload Characterization, pp. 47-56, 2008.
[11] A. Patel and K. Ghose, “Energy-efficient mesi cache coherence with pro-active snoop filtering for multicore microprocessors”, in Proc. of the 13th Int. Symp. on Low power electronics and design, pp. 247-252, 2008.
[12] M. Martin et al., “Token coherence: decoupling performance and correctness”, in Proc. 30th Annual Int. Symp. on Computer Architecture, pp. 182-193, 2003.
[13] M. K. Martin, et al. “Multifacet's general execution driven multiprocessor simulator (GEMS) toolset”, SIGARCH Computer Architecture News, v. 33, No. 4, pp.92-99. November 2005.
[14] X. Lin, L.M. Ni, “Multicast communication in multicomputer networks”, IEEE Trans. Parallel Distrib. Syst., v.4, pp. 1105-1117, 1993.
[15] M. Daneshtalab et al., “A Generic Adaptive path-based routing method for MPSoCs,” Elsevier Journal of Systems Architecture (*JSA-elsevier*), Vol. 57, No. 1, pp. 109-120, 2011.
[16] R. V. Boppana et al., “Resource deadlock and performance of wormhole multicast routing algorithms”, IEEE Transactions on Parallel and Distributed Systems, pp. 535-549, 1998.
[17] D. Panda et al., “Multi destination message passing in wormhole k-ary n-cube networks with base routing conformed paths”, IEEE Transactions on Parallel and Distributed Systems, pp. 76-96,1999.
[18] J. Duato, “On the design of deadlock-free adaptive routing algorithms for multicomputers: Theoretical aspects”, in Proc. Second Europe Distributed Memory Computing Conf., Apr. 1991.
[19] M. Ebrahimi et al., “HARAQ: Congestion-Aware Learning Model for Highly Adaptive Routing Algorithm in On-Chip Networks,” in Proceedings of 6th ACM/IEEE International Symposium on Networks-on-Chip (NOCS), pp. 19-26, May. 2012, Denmark.
[20] M. Dehyadegari et al., “An Adaptive Fuzzy Logic-based Routing Algorithm for Networks-on-Chip,” in Proceedings of 13th IEEE/NASA-ESA International Conference on Adaptive Hardware and Systems (AHS), pp. 208-214, June 2011, USA.
[21] M. Ebrahimi et al., “Exploring Partitioning Methods for 3D Networks-on-Chip Utilizing Adaptive Routing Model,” in Proceedings of 5th ACM/IEEE International Symposium on Networks-on-Chip (NOCS), pp. 73-80, May 2011, USA.
[22] L. Wang, H. Kim and E.J. Kim, “Recursive Partitioning Multicast: A Bandwidth-Efficient routing for Networks-On-Chip”, Int. Symp. on Networks-on-Chip (NOCS), CA, pp. 64-73, 2009.
[23] Z. Liu, J.Duato, “Adaptive Unicast and Multicast in 3D Mesh Networks”, in Proc. of the Twenty-Seventh Hawaii Int. Conf., v.1, pp.173-182, 1994.
[24] J.Duato, “A New Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks”, IEEE Trans. on Parallel and Dist. Sys., pp.1320-1331, 1994.

[25] E. O. Amnah, W.L. Zuo, “Hamiltonian Paths for Designing Deadlock-Free Multicasting Wormhole-Routing Algorithms in 3-D Meshes“, Journal of Applied Sciences, pp. 3410-3419, 2007.
[26] L.M. Ni and P.K. McKinley, “A survey of wormhole routing techniques in direct networks”, IEEE Computer, v.26, i.2, pp.62-76, 1993.
[27] R.S. Ramanujam, B. Lin, “Randomized Partially-Minimal Routing on Three-Dimensional Mesh Networks”, IEEE Computer Architecture Letters, vol. 7, no. 2, pp. 37-40, 2008.
[28] X. Li, P.K. Mckinley, L.M. Ni, “Deadlock-free multicast wormhole routing in 2-D mesh multicomputers“, IEEE transactions on Parallel and Distributed Systems, v.5, i.8,pp. 793-804, 1994.
[29] M. Ebrahimi et al., "A High-Performance Network Interface Architecture for NoCs Using Reorder Buffer Sharing," in Proc. of PDP conference, pp. 547-550, 2010.
[30] H. Wang et al., “Orion: A power-performance simulator for interconnection networks”, In MICRO 35, pages 294–305, 2002.
[31] F. Li, C. Nicopoulos, et. al, “Design and Management of 3D Chip Multiprocessors Using Network-in-Memory”, ISCA-33, pp, 130-141, 2006.
[32] D. Park, et. al , “MIRA: A Multi-Layered On-Chip Interconnect Router Architecture,” 35th International Symp. on Computer Architecture (ISCA), pp.251-261,2008.
[33] N. Muralimanohar, et al., “Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0”, In proc. 40th IEEE/ACM International Symposium on MICRO, pp. 3–14, 1-5 Dec. 2007.
[34] I. Loi and L. Benini, “An Efficient Distributed Memory Interface for Many-Core Platform with 3D Stacked DRAM”, in Proc. of the DATE Conference, Germany, pp. 99-104, 2010.
[35] P. Kongetira, K. Aingaran, and K. Olukotun, “Niagara: a 32-way multithreaded Sparc processor”, IEEE Micro, vol. 25, pp. 21-29, 2005.
[36] B. M. Beckmann and D. A. Wood, “Managing wire delay in large chip-multiprocessor caches”, In Proc. of the 37th annual IEEE/ACM International Symposium on MICRO, pp. 319-330, 2004.
[37] I. Savidis et al., “Electrical modeling and characterization of through-silicon vias (TSVs) for 3D integrated circuits”, Microelectronics Journal, v. 41(1), pp. 9-16, 2010.

Masoumeh Ebrahimi received her B.S. degree in computer engineering from School of Electrical and Computer Engineering, University of Tehran in 2005, and M. S. degree in computer architecture from Azad University, Science and research branch, in 2009. Since spring 2009 she has been working in the Embedded Computer Systems laboratory, University of Turku. Her PhD thesis is focused on routing protocols in 2-D and 3-D NoCs.

Masoud Daneshtalab received his PhD degree in information and communication technology from University of Turku in 2011. He is currently a Senior Researcher in Department of Information Technology at University of Turku, Finland. He has served as a Guest Editor for Elsevier Journal of Systems Architecture (JSA), Springer Computing journal, and ACM Transactions on Embedded Computing Systems (ACM TECS). He also co-organizes a special session on On-Chip Parallel and Network-Based Systems (OCPNBS) in the Euromicro PDP conference. His current research interests include on/off-chip interconnection networks, manycore systems-on-chip, embedded operating systems, 3D stacked architectures, machine learning, data centres architecture, and cloud computing. He is a member of IEEE and has published more than 80 refereed international journals and conference papers. He is currently in a Technical Program Committee member of different IEEE and ACM conferences, including NOCS, ESTIMedia, DSD, PDP, ICESS, NESEA, CASEMANS, NoCArc, and DATICS.

Pasi Liljeberg received his M.Sc. and Ph.D. degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively. He is an Adjunct Professor in embedded computing architectures at the University of Turku, Department of Information Technology. Since January 2010 he has been working in the Embedded Computer Systems laboratory, University of Turku. His current research interests include intelligent network-on-chip communication architectures and fault tolerant.

Juha Plosila is an Adjunct Professor in Digital Systems Design at the University of Turku, Department of Information Technology. He received a PhD degree in Electronics and Information Technology from the University of Turku in 1999. Plosila is an Associate Editor of International Journal of Embedded and Real-Time Communication Systems published by IGI Global. His current research interests include SoC/NoC design issues primarily focusing on on-chip communication architectures, development of a multitasking virtual machine architecture based on an in-house Java processor, fault-tolerance methods, and dynamically reconfigurable service based system architectures.

Jose Flich received the M.S. and Ph.D. degrees in computer science from the Technical University of Valencia, Valencia, Spain, in 1994 and 2001, respectively. He joined the Department of Computer Engineering, Technical University of Valencia, in 1998, where he is currently an Associate Professor of computer architecture and technology. His current research interests include high-performance interconnection networks for multiprocessor systems, cluster of workstations, and networks-on-chip. He has published over 100 papers in peer-reviewed conferences and journals. He has served as a Program Committee Member in different conferences, including ICPP, IPDPS, HiPC, CAC, ICPADS, and ISCC.

Hannu Tenhunen received his PhD from Cornell University, Ithaca, USA in 1985 and since that he has held professor, invited professor, or honorary professor positions in Tampere, Stockholm, Ithaca, Grenoble, Shanghai, Beijing and Hong Kong. During the recent years he has been director of Turku Centre of Computer Science and invited professor at University of Turku where he has established Computer Systems Laboratory, the leading computer architecture and systems research centre in Finland. Prof. Tenhunen's research interest is in new computational architectures, dependability issues, on-chip and off-chip communication and mixed signal and interference issues in complex electronic systems including 3-dimensional integration. He has done over 600 publications or invited key note talks internationally.