Network-on-Chip Multicasting with Low Latency Path Setup

ABSTRACT

A low-latency path setup approach with multiple setup packets for parallel set is presented. It reduces the header overhead compared to multiaddress encoding. Further, we propose four variants of deadlock-free multicast routing algorithms using different subpath generation methods, different destination partitioning, and channel sharing strategies. Experimental results show that the quatuor partitions path-like tree outperforms other algorithms.

1. INTRODUCTION

Today, over billions of transistors can be integrated on a single chip. According to ITRS (International Technology Road map for Semiconductor) [1], the 20-nm era is expected to begin sometime in 2013 or 2014. This means that hundreds of cores and great amount of memory can be integrated on a single chip to further advance multiprocessor systems-on-chip (MPSoC) [2]. An efficient communication mechanism is important to the performance of MPSoCs. For MPSoCs with a small amount of cores, traditional bus solutions are efficient and enough. However, for MPSoCs containing a large number of IP bolcks, buses can not meet the requirement of performance and are not scalable. Network-on-Chip (NoC) has been proposed to address the global interconnect problems of these systems. A number of NoCs have been developed since year 2000, such as NOSTRUM [3], RAW [4], TRIPS [5], SPIN [6], and Æthereal [7] etc.

NoC communication can be classified into two categories. One is *unicast*, which forms packets at a source node and transfers it to only one destination node. The other is *multicast*, which sends packets to multiple destination nodes. Multicast can be applicable to numerous parallel algorithms, single-program multiple-data (SPMD) program model, data-parallel programming model, cache coherency in distributed shared-memory architectures [8]. Multicast can be implemented by sending multiple copies of a packet to destination nodes via unicasts, but it incurs high overhead and is inefficient [9].

This paper proposes a low-latency multicast path setup method. With this approach, the setup is achieved by using several parallel packets instead of one large setup packet, which contains all the destination nodes' addresses. During the data transfer, the ID-based approach significantly reduces the overhead of packet header compared with multi-address encoding. Based on this method, four multicast algorithms are presented. They use different partitionings, different sub-path generation methods, and channel sharing strategies.

The paper is organized as follows. In Section 2, a brief review of related work is presented. In Section 3, the multicast mechanism and algorithm realization are discussed. In Section 4, we describe the router implementation and discuss deadlock freedom. The experimental results are shown in Section 5. Finally, conclusion and future work are given in Section 6.

WOODSTOCK '97 El Paso, Texas USA

2. RELATED WORK

Multicast algorithms can be classified into two categories: *tree based* and *path based*. A router gets a packet from the input buffer and forwards it to the outgoing channels according to the result of the routing computation. In a tree-based approach, the number of outgoing channels to forward packet could be 1 to $n-1$, where n is the number of outgoing channels that a router has. In a path-based approach, the number of outgoing channels is limited to at most two, one of which should be the local channel. Some researches have proposed the tree-based approaches. A hardware support for a tree-based multicast named XHiNoC is proposed in [10]. With this scheme, they used ID-manager (IDM) to manage multicast [10]. Each multicast is assigned a table ID at a port, by which the packet can get the routing result. The same multicast packet may get a different ID at a different router. When the packet is transferred to outgoing channel, the IDM will change the ID field of the packet with the new table ID [10]. Another tree-based routing approach named Virtual Circuit Tree Multicasting (VCTM) is introduced in [11]. Different from XHiNoC, VCTM used an identical ID to manage multicast. VCTM constructs the multicast tree incrementally by sending several unicast setup packets to destinations. Each setup packet is routed using the Dimension-Ordered Routing (DOR) algorithm and the routing result is stored in a table according to the identical ID [11]. A hardware supported multicast routing on any shape of tree-based paths is proposed in [12]. Based on this scheme, two power-efficient tree-based multicast routing algorithms are presented [12].

The tree-based approaches which adopt wormhole routing are very easy to block at branch nodes and trap into the status of deadlock. Path-based approaches can be a promising way to overcome the shortcoming of tree-based counterparts. A connection-oriented multicasting is proposed in [13]. In their scheme, a multicast procedure consists of establishment, communication and release phases [13]. This scheme is suitable for large block transmission. For frequent small-size message transmission, the setup and release processes occupy a high percentage of time overhead. Common path-based routing algorithms are Hamilton path algorithm where a unidirectional Hamilton path of the network is constructed [14]. In [14], dual path (DP) and multi path (MP) are presented. For DP routing algorithm, the destination node set is divided into two subsets: D_H and D_L. For MP

routing algorithm, the destination node set is divided into four subsets [14]. MP can reduce the path lengths efficiently. Column path (CP) routing algorithm partitions the destination set into $2k$ subsets, where k is the number of columns in the mesh network [15]. A short-distance path-based multicast routing algorithm is proposed, which optimizes the destination order to gain benefits upon message latency [16].

The path-based approaches mentioned above pack all the destination addresses into the header of the message. This results in high overhead when the destination set is large. In contrast, our proposed method uses ID-tag, which reduces the message size effectively. Since the time overhead of the sequential setup is still high for large size of destination sets, we developed an incremental parallel setup procedure to reduce the setup time.

3. MULTICAST MECHANISM AND ALGORITHM REALIZATION

In this section we describe the mechanism of incremental parallel path setup and multi-address coding. Base on this scheme, we also present four multicast algorithms that use different partitioning strategies, different path generation for destination subsets and channel sharing strategies.

3.1 Multicast path setup mechanism and multi-address coding

The path-based multicast setup process commonly utilizes one packet with multi-destination addresses to implement [13]. Firstly, the packet routes to a nearest destination. If the path is based on the look-up table on router, the corresponding table entry is updated. If the path is based on the holding of the virtual channel, the channel will not be allocated to other packets until the multicast ends. After arriving at the destination node, the packet destination field will be changed to the next destination node by copying the address from the payload, and the setup process continues unless there is no address left in the payload of the packet. For the look-up table based scheme, when the packet is ejected into the last destination node, a reply packet is sent to source node to inform the success of setup. After the source node receiving the reply packet, it can start to transfer multicast data. For the virtual channel holding scheme, the data just follow the last flit that contains destination address. This scheme needs great amount of flits to hold all the destination addresses during each message transfer. It is suitable for the transmission of large infrequent messages. For mass reusing multicast groups, the look-up table based approach is more efficient.

Our multicast scheme divides the multicast group into several destination sets. Each set forms one multicast path, which contains several sub-paths. Each sub-path has one destination and is setup by one packet. The setup packet contains an intermediate node address in the destination (DST) field and a destination node address in the first 6 bits of payload. Usually, the intermediate node is the destination node of another sub-path. The setup process of each sub-path is made up of two phases. During the first phase, packet routes to the intermediate node from source node. After arriving at the intermediate node, packet's DST field is changed to the final destination node. Then the setup process enters the second phase. During the second phase, the routing result (outgoing port) is not only used to forward the packet but also update the multicast look-up table. The entry of update is determined by the source node ID and its local multicast ID's combination. When the destination receives the packet, a sub-path is preserved from the intermediate node to the destination node. The setup process of the sub-paths is performed in parallel to reduce the total setup time.

Fig. 1(a) shows an example of the incremental parallel multicast setup process. To simplify calculating of the setup time, we consider the time per hop of a packet as a constant

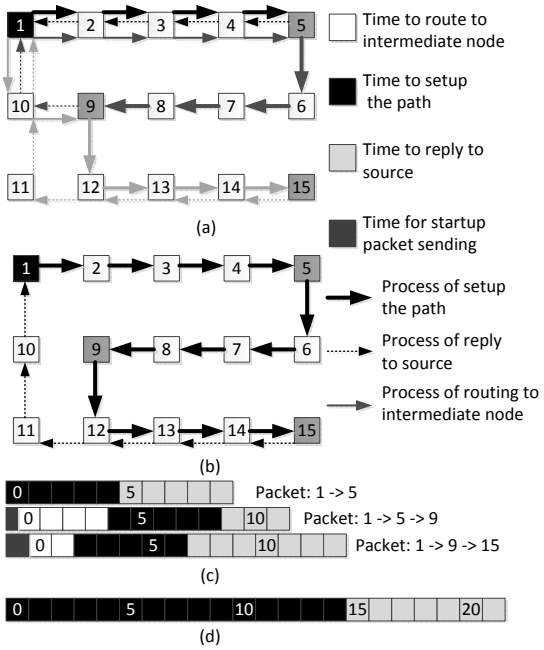

Figure 1: An example of incremental setup process

T_{hop}. The time between preceding setup packet injection and follow-up setup packet injection is T_{gap}, which is typically smaller than T_{hop}. We assume the destination node sends a reply packet immediately, so the time between the setup packet arriving at the destination node and the reply packet injection is also negligible. The whole multicast path is divided into three sub-paths which need 3 packets to set up. Fig. 1(c) shows the three sub-paths' setup time. As the sub-path setup runs in parallel, the total setup time is only $14\,T_{hop} + 2\,T_{gap}$. Fig. 1(b) shows the common way of setup. The setup packet needs 15 T_{hop} to the last destination (include 2 T_{hop} to change destination address and forward packet at node 5 and 9), and the reply packet needs 7 T_{hop}. So the total time of multicast setup is 22 T_{hop} which takes $8\,T_{hop} - 2\,T_{gap}$ more than the incremental scheme (Fig. 1(d)).

Table 1: Multi-address coding bits requirement

	All Destination Encoding					Bit Stream Encoding	ID-based
destination / mesh	10	20	30	50	100	10-100	10-100
64	60	120	180	300	600	64	less
256	80	160	240	400	800	256	than
1024	100	200	300	500	1000	1024	10

Our multicast routing is based on ID-tag. This scheme is suitable for small messages frequently sent to the same destination group. Jerger et al. showed that there is significant reuse of a small percentage of multicast in various cache coherence protocols, such as directory-based coherence, TokenB, region-based coherence, the Opteron protocol, and the TRIPs operand network [11]. Multicast reuse is defined as multicasts from the same source node intended for the same destination set. From [11], we find when the number of destination group sets reaches 150, the multicast coverage percentage is over 70%. Thus, using 8 bits to identify a multicast group is enough. All Destination Encoding(ADE) and Bit Stream Encoding (BSE) are mentioned in [17]. ADE uses $n \log_2^m$ to represent destinations where n is the destination size and m the total number of the routers. BSE uses m bits to represent all possible destinations. TABLE 1 shows the number of bits needed to code the multi-address between ADE and BSE with different destination number under different network sizes. From TABLE 1, we can see that the drawback of ADE is its significant header overhead when the destination size is large. The drawback of BSE lies in that, when the network size is large and the destination size small, the overhead is also significant. Due to the significant overhead of ADE and BSE, the performance on latency and throughput is strongly affected.

3.2 Multicast routing algorithms

Based on the incremental parallel multicast path setup. we explore four variants of multicast routing algorithms, which consist of three steps:*destination set partition*, *path generation*, *setup packet queue generation*.

3.2.1 Destination set partition

Algorithm: Partitioning algorithm for three subsets
Input: Destination set D, Source node $s : (x_0, y_0)$;
Output: Destination subset D_{up}, D_{down},$D_{mid-right}$;

1: $D \rightarrow \{D_{up}, D_{down}, D_{mid-right}\}$
2: $D_{up} := \{(x, y)|x < x_0||(x = x_0 \&\& y < y_0)\}$
3: $D_{down} := \{(x, y)|x > x_0\}$
4: $D_{mid-right} := \{(x, y)|x = x_0 \&\& y > y_0\}$

Figure 2: Partitioning algorithm for three subsets.

Algorithm: Partitioning algorithm for four subsets
Input: Destination set D, Source node $s : (x_0, y_0)$;
Output: Destination subset D_{lt}, D_{lb},D_{rt},D_{rb};

1: $D \rightarrow \{D_{lt}, D_{lb}, D_{rt}, D_{rb}\}$
2: $D_{lt} := \{(x, y)|x \leq x_0 \&\& y < y_0)\}$
3: $D_{lb} := \{(x, y)|x > x_0 \&\& y < y_0)\}$
4: $D_{rt} := \{(x, y)|x \leq x_0 \&\& y \geq y_0)\}$
5: $D_{rb} := \{(x, y)|x > x_0 \&\& y \geq y_0)\}$

Figure 3: Partitioning algorithm for four subsets.

In this step, all the destination nodes will be partitioned into different subsets according their coordinates. Reasonable partitioning could reduce the path length effectively. Here we consider two approaches: three subsets and four subsets.

Fig. 2 shows the partitioning algorithm for three subsets (PAT). Different from DP [14], the reason of third subset is to avoid the multicast path forming a ring at source node. Fig. 3 shows the partitioning algorithm for four subsets (PAF).

3.2.2 Path generation

After partitioning, a path connecting the source node to all the nodes in a destination subset needs to be generated. Here we proposed two different algorithms, both of which follow the west-first turn model to be free from deadlock. Because the two algorithms are similar, we show them in one pseudo code. Here we define the algorithm with optimization as PGO, the other one without optimization is PG. As shown in Fig. 4, step 1 is to divide the destination nodes into several subsets, of which the nodes are in the same column. In step 2, sort the nodes in the same column. Our algorithms are based on the west-first turn model, so the source node will firstly connect to the most western column that has the destination nodes, then move to the east one column by one column. The difference between the two algorithms is the connect direction change mechanism. For the PG, when a path changes from one column to another, the direction will be changed. But for the PGO, if it is not necessary, it will not change the connect direction. In the pseudo code, steps 6 and 7 are specific for PGO while step 23 for PG.

3.2.3 Setup packets queue generation

After the path generation, a scheme is needed to produce the setup packets. The setup packet format is shown in Fig. 5. As mentioned previously, the setup consists of two phases, which are denoted by two types of packets: MC_SET_1, MC_SET_2. During the first phase, the setup packet routes to an intermediate node that is indicated by the DST field of the packet. After arriving at the intermediate node, the real destination node address will be copied to the DST filed,

Algorithm: Generate path based on the west-first turn model without optimization and with optimization
Input: Destination set D, Source node $s : (x_0, y_0)$, Direction of connection dir, algorithm $algo$;
Output: Pair set D_x, D_y;
Initial: $D_x := \emptyset, D_y := \emptyset, tmp := s$

1: $D_{col} \leftarrow \{C_1, C_2 \ldots, C_n\}$ which meet follow conditions :
   α. $D = \bigcup_{i=1}^{n} C_i$ and $\forall i, j \in \{1 \ldots n\}, i \neq j : C_i \cap C_j = \emptyset$;
   β. $\forall a \in C_i, b \in C_j$: if $i = j$ then $a.y = b.y$;
   if $i < j$ then $a.y < b.y$;
2: **foreach** $C_i \subseteq D$ sort its elements to meet follow condition:
   $C_i = \{a_0 \ldots a_{k_i}\}$ and $\forall m, n \in \{0 \ldots k_i\}, m < n : a_m.x < a_n.x$
3: **for** $i := 1$ to n **do**
4: $\{a_0 \ldots a_{k_i}\} \leftarrow C_i$
5: **if** algo = PG **then goto** step **8 endif**
6: **if** dir = N and $tmp.x < a_{k_i}.x$ **then** dir:= S
7: **elseif**dir = S and $tmp.x > a_0.x$ **then** dir := N
8: **if** dir = N **then**
9: **if** $tmp.x \geq a_{k_i}.x$ **then**
10: $D_y := D_y \cup (tmp, a_{k_i}) \bigcup_{m=0}^{k_i-1} (a_{m+1}, a_m)$
11: **else**
12: $D_x := D_x \cup (tmp, a_{k_i}) \bigcup_{m=0}^{k_i-1} (a_{m+1}, a_m)$
13: **end if**
14: $tmp := a_0$;
15: **else**
16: **if** $tmp.x \geq a_0.x$ **then**
17: $D_x := D_x \cup (tmp, a_0) \bigcup_{m=0}^{k_i-1} (a_m, a_{m+1})$;
18: **else**
19: $D_y := D_y \cup (tmp, a_0) \bigcup_{m=0}^{k_i-1} (a_m, a_{m+1})$;
20: **end if**
21: $tmp := a_{k_i}$;
22: **end if**
23: **if** algo = PG **then** $dir := dir = S?N : S$; **endif**
24: **end for**

Figure 4: Algorithms for generating path.

which means that the packet enters the second phase. During the second phase, the routing result will be produced by the routing computation unit according to the X First/Y First field. If this filed is set, the routing result will be in the Y direction firstly. Only when current node's Y coordinate equals to the destination node, the routing result will change to the X direction. Otherwise, the packet routes in the X direction firstly. After the process of path generation, we obtain two sets: D_x, D_y, the elements of which are used to generate setup packets. If the element belongs to D_x, the corresponding X First/Y First field is set to 0, otherwise is set to 1. The first part of the element is filled into the DST filed while the second is filled into the first 6 bits of payload filed. For each subset, a free local ID will be filled into the field of MCT#.

3.2.4 Four algorithm cases

Here we propose four algorithms, which are combinations of different partitioning and path generation means. Algo-

	Head/ Body /Tail/ HBT	Packet Type	X First/ Y First	SRC	MCT#	DST	VCID	Payload
	2 bits	3 bits	1 bit	6 bits	4bits	6 bits	2 bits	Payload
Multicast setup_ first	11: HBT	001 MC_SET_1	0:X first 1:Y first					6 bits ' 2 nd DST
Multicast setup_ second	11: HBT	010 MC_SET_2	0:X first 1: Yfirst					xxx

Figure 5: Setup packet format

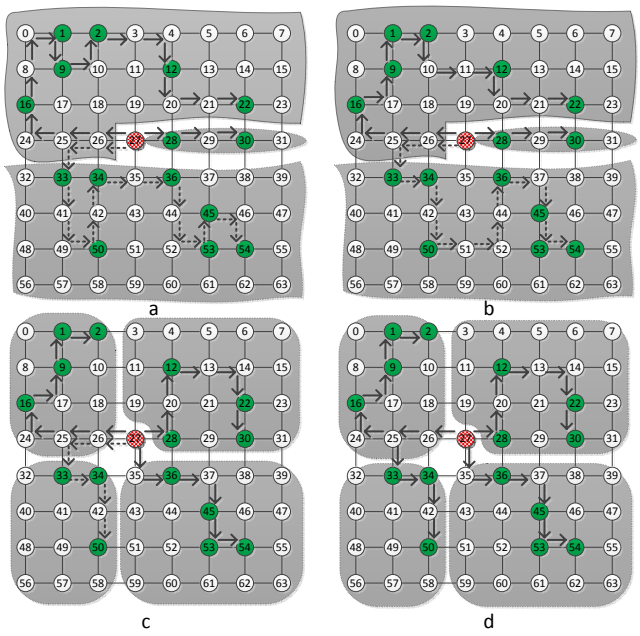

Figure 6: An example for four algorithms: a.TPNOOPT b.TP c.QP d.QPLT

rithm that has three partitioned subsets but no optimization in path generation (TPNOOPT) is defined as a combination of PAT and PG. Algorithm that has three partitions and optimization in path generation (TP) is defined as a combination of PAT and PGO. Algorithm that has quad partitions in destination and optimization in path generation (QP) is defined as a combination of PAF and PGO. The fourth algorithm (QPLT) is basically similar to the QP, and the only difference is the setup packets of all the subsets will share one MCT#, which means that the multicast is not based on path, which we call path-like tree. Fig. 6 shows the example of the algorithms mentioned above. All the subfigures show a 8×8 mesh network where node 27 sends its multicast packets to destinations 1, 2, 9, 12, 16, 22, 28, 30, 33, 34, 36, 45, 50, 53, 54. Fig. 6.a shows the case of TPNOOPT. Three subsets are partitioned. The first subset (D_{up}) contains the nodes that can be reached using one path, including node 1, 2, 9, 12, 16, 22. In order to avoid the generation of ring, the nodes that are to the due east of the source node are included in the second subset ($D_{mid-right}$), where there are node 28 and 30. The third subset (D_{down}) includes the rest of the destination nodes, where there are node 33, 34, 36, 45, 50, 53, 54. The mutlicast paths generated without optimization need 35 hops in total to finish transmission. Fig. 6.b shows the case of TP, where the only difference is that the path generation algorithm uses optimization. The total number of hops is 31, which is 4 hops less than the case of TPNOOPT. Fig. 6.c shows the case of QP, where the only difference from the case of TP is that the partitions is four. For this case, the total number of hops is 27, which is 4 hops less than the case of TP. The case of QP also gets the best result in longest distance path. It only needs 8 hops (from 27 to 2) while the TPNOOPT needs 16 hops, TP needs 14 hops. The more startup time due to more injection is negligible while obtaining the obvious advantage of less hops. The partitioning way and the path generation way between QP and QPLT are the same. The difference is setup packets generation. The QPLT will use only one table entry, which means that the multicast is a path-like tree. The message will share some channels after being injected into network. As shown in Fig. 6.d, the channels from 27 to 25 will be shared, so the total number of hops is reduced to 25.

4. IMPLEMENTATION

We have implemented a router supporting both multicast and unicast. Also, a simple and effective mechanism is adopted to avoid deadlock.

4.1 Router architecture

The router architecture, which supports both multicast and unicast, is shown in Fig. 7. For unicast, the routing is based on the simple XY routing. For multicast, the routing is based on looking up a multicast table. In order to support our proposed incremental parallel setup scheme, a packet type conversion logic (PTC) is integrated in input ports. In buffer writing period, if the packet is MC_SET_1, the PTC is activated and enters the judgement and conversion period. Otherwise the packet is just written into buffer. During the judgement and conversion period, the PTC will judge if the current node is an intermediate node and covert the packet fields. If the packet fields are changed, the setup enters the second period. When the routing result returns, it is not only used to select the output port but also update the corresponding table entry of the multicast table in the router. The multicast table is partitioned statically. Source node ID and local multicast ID's combination indicates the table entry. This scheme is limited in scalability. In the future weill consider dynamic organization of the look-up table.

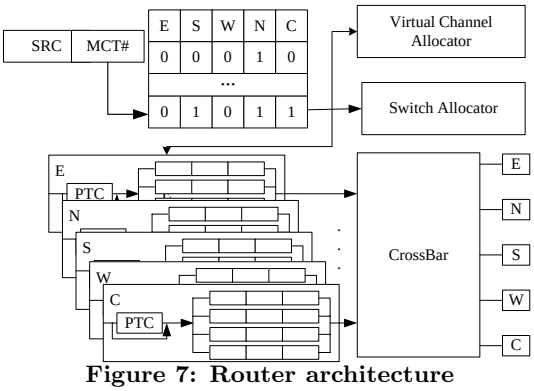

Figure 7: Router architecture

4.2 Deadlock freedom

Two kinds of deadlocks possibly occur in the network. One occurs when dependent resource requirements form a cycle. To avoid this kind of deadlock, we adopt the West-first turn model for routing. For unicast packets, XY routing is used. Because the XY turn model is a subset of the west-first turn model, mixing the two is also deadlock-free.

The other kind of deadlock may happen when there are more than one multicast existing in the network. One of this deadlock scenarios is that two multicast packets a and b exist in port E and port W in the same router, and both of them require port S of the northern router and the local port. Packet a gets the grant of using port S while packet b gets the the local port. Because the basic element of transmission is flit, the hold and requirement mechanisms inhibit the packets to be forwarded to all the ports and are blocked in buffers. Some solutions have been proposed to solve this kind of deadlock. Partitioning and systematically allocating virtual channels (VCs) is presented to avoid deadlock in [15]. Kumar proposed a hardware tree-based multicast routing algorithm with a deadlock detection and recovery mechanism [18]. Young proposed a dynamic packet fragmentation to solve the deadlock [19]. This scheme allows to release the hold of an output virtual channel and enable other packets to use the freed VC when necessary.

Our proposed router avoids deadlock by using large enough buffers to hold entire packets and reserving buffer pointers for each port in an input virtual channel. As mentioned above, the packet a gets the grant of using port S. So the

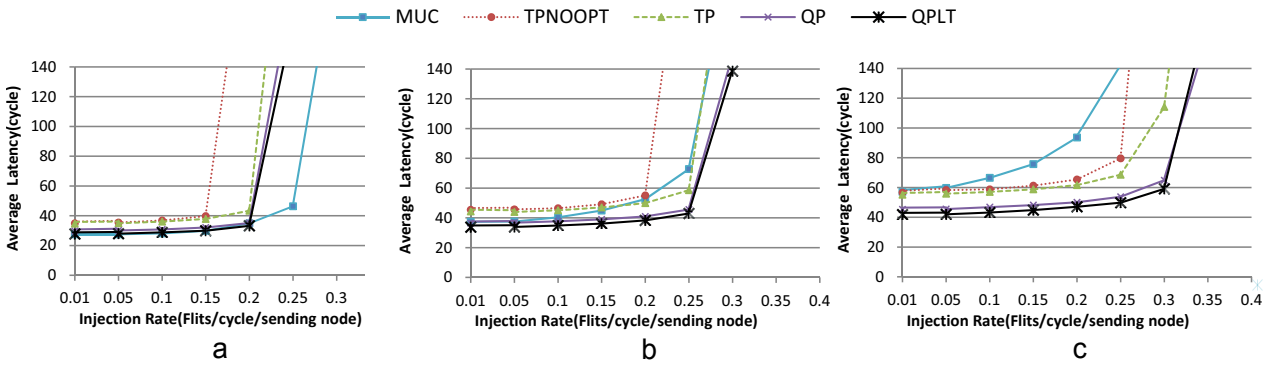

Figure 8: Performance for the 5,10,20 nodes destination group under only multicast traffic.

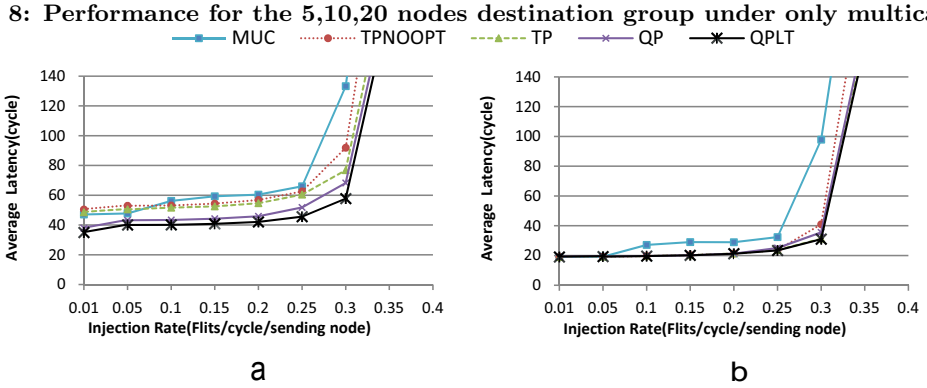

Figure 9: Performance for mixed traffic: a. multicast traffic b. unicast traffic

N pointer in VC can be increased and the following flits can be transferred to the port S of the northern router while the L (local) pointer is still pointing to the first flit. After the whole packet is transferred to the port S, the VC will be released soon and the packet b will get the allocation of port S. Similar to packet a, packet b is transferred to local port and then the packet a can get the local port. The flit could be discarded from the buffer only when all outgoing port pointers are advanced. Base on this scheme, the deadlock is avoided. As buffers are area consuming, messages are preferably short. Long messages can be fragmented into several packets at the source node and transferred to destinations. Thus the buffer size can be kept moderate.

5. EXPERIMENTS

To evaluate the four proposed multicast routing algorithms, we have developed a cycle accurate virtual cut-through NoC simulator implemented in SystemC. The simulator calculates the average packet latency and power consumption for the packet transmission. The network topology is a 8×8 mesh. Multicast routing is based on a lookup table while the unicast routing is based on the XY routing. The number of virtual channels of each input port is 4. The number of MCT entries is set to 64. For the performance metric, we define the packet latency as the number of cycles from when the packet entering into the waiting queue till the time the packet is ejected from the network [20]. As a baseline, multicast with multiple unicast copies (MUC) is also implemented. To evaluate the cases conveniently, we define the injection rate of MUC, TPNOOPT, TP and QP as equivalent injection rate as the QPLT. For example, if a source node destines to 10 nodes and the injection rate is 0.01 flit/cycle for QPLT, the actual injection rate will be 0.1 flit/cycle for MUC. For TPNOOPT, TP and QP, the injection rate will be $0.01 \times n$ flit/cycle, where n is the partition number of the destinations.

5.1 Performance with multicast traffic

In the first experiment we use only multicast traffic. During the simulation, the PEs generate 3-flit packets and inject them into the waiting queue every constant interval based on the injection rate. A uniform distribution was used to construct the destination set of each multicast packet while the source node is also selected randomly. We configure three scenarios: 16 source nodes, each node destines to 5 nodes; 8 source nodes, each node destines to 10 nodes; 4 source nodes, each node destines to 20 nodes.

Fig. 8 shows the average communication delay as a function of the average injection rate of the sending node. Fig. 8.a shows the case of 5 nodes destination group, where the MUC exhibits the best performance. The QPLT performs similarly to MUC, but becomes saturated earlier. The latency of QP is 8% more than the MUC while the TP and TPNOOPT take 26% more than MUC. Among all the algorithms, the TPNOOPT is worst in throughput and gets saturated at 0.15 flit/cycle injection rate.

Fig. 8.b shows the case of 10 nodes destination group. QPLT shows the best performance, but MUC is better than TPNOOPT and TP. TPNOOPT is still worst in these cases. Fig. 8.c shows the case of 20 nodes destination group. In this case, MUC exhibits the worst performance of all. QPLT outperforms the other four algorithms.

From the experimental results, we can conclude that, for small-size destination groups, MUC outperforms other multicast algorithms. For increasing size of destination groups, the multicast algorithms have better performance than MUC. We can also conclude that good partitioning can gain extra performance improvement for the reason that it reduces the average path length. Because of channel sharing, the QPLT always outperforms QP. For large number of destinations, the path optimization will have more effect upon the total performance improvement, but for small-size destination groups, the effect is insignificant.

5.2 Performance with mixed traffic

We also simulated mixed multicast and unicast traffic scenario where multicast traffic accounts for 20% of total traffic, which is similar to the scenarios of some cache coherence protocols [16]. The unicast traffic is also uniformly distributed. Fig. 9.a shows the multicast performance under the mixed traffic. The result indicates that the QPLT outperforms other algorithms for multicast traffic. Fig. 9.b

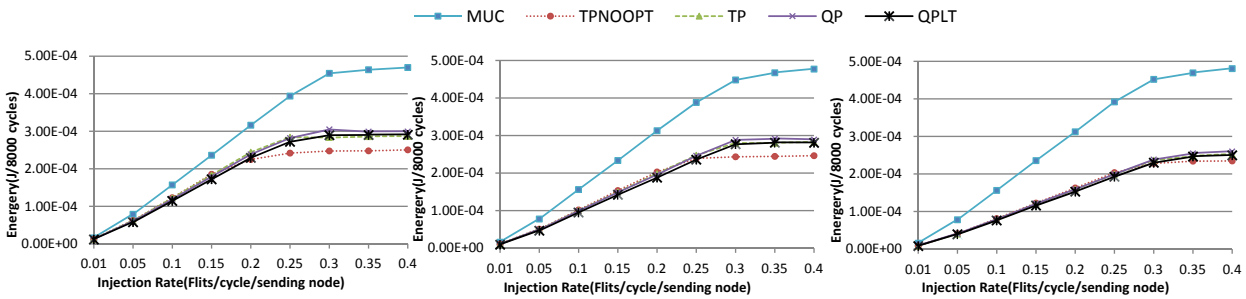

Figure 10: Power consumption of different group size: a. 5 nodes b. 10 nodes c. 20 nodes

shows the performance of unicast traffic under the mixed traffic, where QPLT is still the best. QPLT has the least effect on the performance for unicast traffic for the reason that it utilizes least number of links to realize the multicast.

5.3 Power Consumption

We calculated the power consumption according to the library of noxim [21]. In the power model, the power consumption contains the power of routing, selection, forward, incoming and standby. We set the power of routing as routing table based. The power consumption is counted and compared under the multicast traffic with different destination group sizes. Fig. 10 shows the power consumption of the proposed algorithms with different group sizes under different injection rates. The network power consumption is recorded after 8000 cycles of warm-up time. We also normalize the power consumption to that of MUC.

Fig. 11 shows the average power consumption of the proposed algorithms with different group sizes. Fig. 10 and Fig. 11 show that the QPLT outperforms other algorithms. The proposed algorithms become more power efficient when the size of destination groups increases.

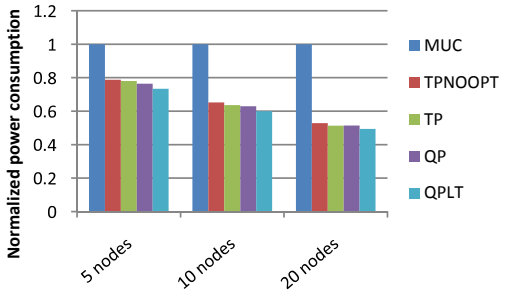

Figure 11: Power consumption with multicast traffic

6. CONCLUSION

A method of incremental parallel setup of multicast paths is presented. This method uses an ID-based approach to route packets, which significantly reduces the overhead of packet header compared with multi-address encoding. Base on this, four different multicast routing algorithms are proposed. They adopt different partitioning, path generation approaches, and link sharing strategies. From the experiments we find that QPLT outperforms other algorithms for its reasonable partitioning, better path generation approach and link sharing. We also find that mutlicasting becomes more efficient when the destination set is larger.

In the future, we will thoroughly study both path-based and tree-based multicast approaches, evaluating and comparing their performance and power consumption under different application scenarios.

7. REFERENCES

[1] http://www.itrs.net.
[2] W. O. Cesario and L. Lyonnardl. Multiprocessor SoC plarforms: a component-based design approach. In *Proceedings of the Design and Test of Computers*, pages 52–63, November 2002.
[3] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the Design Automation and Test in Europe Conference*, February 2004.
[4] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, et al. The Raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, 2002.
[5] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, et al. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In *Proceedings of the 30th annual international symposium on Computer architecture*, February 2003.
[6] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the Design, Automation and Test in Europe Conference*, March 2000.
[7] E. Rijpkema, K. Goossens, J. Dielissen A. Rădulescu, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEE Proceedings: Computers and Digital Technique*, 150(5):294–302, September 2003.
[8] X. Lin, P. K. McKinley, and L. M. Ni. Deadlock-free multicast wormhole routing in 2-d mesh multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):793–804, 1994.
[9] P. Mckinely, H. Xu, and A. H. Esfahanian. Unicast-based multicast communication in wormhole-routed networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1252–1265, 1994.
[10] F. A. Samman, T. Hollstein, and M. Glesner. Multicast parallel pipeline router architecture for network-on-chip. In *Proceedings of the Design, Automation and Test in Europe Conference*, March 2008.
[11] N. E. Jerger, L. S. Peh, and M. Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *Proceedings of the 35th annual international symposium on Computer architecture*, June 2008.
[12] Suppressed for blind review.
[13] Zhonghai Lu, Bei Yin, and Axel Jantsch. Connection-oriented multicasting in wormhole-switched networks on chip. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 205–210, Karlsruhe, Germany, March 2006.
[14] X. lin and L. M. Ni. Multicast communication in multicomputer networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(10):1105–1117, 1993.
[15] R. V. Boppana, S. Chalasani, and C. S. Raghavendra. Resource deadlocks and performance of wormhole multicast routing algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):535–549, 1998.
[16] M. Daneshtalab, M. Ebrahimi, S. Mohammadi, and A. Afzali-kusha. Low-distance path-based multicast routing algorithm for network-on-chips. *IET computers & digital techniques*, 3(5):430–442, 2009.
[17] M. Chiang and L.M. Ni. Multi-address encoding for multicast. *Parallel Computer Routing and Communication*, 853(5):146–160, 1994.
[18] D. R. Kumar, W. A. Najjar, and P. K. Srimani. A new adaptive hardware tree-based multicast routing in k-ary n-cubes. *IEEE Transactions on Parallel and Distributed Systems*, 50(7):647–659, 2001.
[19] H. K. Young, S. Jeff, and D. Jeff. Multicast routing with dynamic packet fragmentation. In *Proceedings of the 19th ACM Greatlakes symposium on VLIS*, pages 113–116, Karlsruhe, Germany, March 2009.
[20] B. Towles and W. J. Dally. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2003.
[21] http://noxim.sourceforge.net.