Parallel Computing 39 (2013) 408–423

Contents lists available at SciVerse ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Avoiding request–request type message-dependent deadlocks in networks-on-chips

Xiaohang Wang [a,b], Peng Liu [b,*], Mei Yang [c], Yingtao Jiang [c]

[a] Intelligent Chips and Systems Research Centre, Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences, Guangzhou, Guangdong 511458, China
[b] Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, Zhejiang 310027, China
[c] Department of Electrical and Computer Engineering, University of Nevada, Las Vegas 89154, USA

ARTICLE INFO

Article history:
Available online 1 June 2013

Keywords:
Streaming protocols
Network-on-chip (NoC)
Message-dependent deadlock
Virtual channel

ABSTRACT

When an application is running on a network-on-chip (NoC)-based multiprocessor system-on-chip (MPSoC), two types of deadlocks may occur: (i) the routing-dependent deadlocks, and (ii) the message-dependent deadlocks. The former type of deadlocks can be avoided by removing any cyclic paths on the application's channel dependency graph. The message-dependent deadlocks, caused by mutual dependency of different control and/or data messages, on the other hand, are very complicated to deal with. In this paper, we focus our study on the request–request type message-dependent deadlocks which may appear in a peer-to-peer streaming system. This type of deadlocks can have devastating effects on applications using streaming protocols that often demands real-time processing over continuous data streams. We show that request–request type of deadlocks can be avoided by proper inclusion of virtual channels (VCs) for the links along the selected routing path. These VCs are not bounded to a particular communication path. Instead, they can be shared among multiple existing communication flows. In this paper, we have formally proved a sufficient condition that determines the minimum number of VCs actually needed for each link of a communication flow such that, request–request type message-dependent deadlocks can be completely avoided. Following this sufficient condition, we propose a path selection and minimum VC allocation (PSMV) algorithm to help determine the minimum number of non-uniform VCs for each link. The PSMV algorithm consists of two major steps. In the first step, we attempt to minimize the maximum number of VCs among all the links. This problem is NP-complete in nature, and it is solved using the proposed mixed integral linear programming (MILP)-based algorithm. In the second step, based on the solution suggested in the first step, the minimum number of VCs for each link is finally determined. The PSMV algorithm can literally be integrated with any existing application mapping algorithm to provide deadlock-free mapping results. One such deadlock-free mapping algorithm is suggested in this paper. Our experiments also show that, compared to an existing flow control based deadlock avoidance method (CTC) and a deadlock recovery method (DR), increase of buffers size in PSMV is within 5% compared to a baseline network configuration. The message latency of PSMV is the lowest among all three designs.

1. Introduction

Network-on-chip (NoC) has been widely accepted as a viable communication infrastructure for current and future Multiprocessor System on Chip (MPSoC) designs [1]. Particularly, NoCs are used in MPSoCs designed for running applications

* Corresponding author.
E-mail addresses: xh.wang@giat.ac.cn (X. Wang), liupeng@zju.edu.cn (P. Liu), mei.yang@unlv.edu (M. Yang), yingtao@egr.unlv.edu (Y. Jiang).

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.05.002

which use streaming protocols ranging from VoIP telephony, playback audio/video, IPTV, online multi-user games, security surveillance, to sensor data analysis [2–4]. These vastly diversified applications using streaming protocols share one commonality as they all demand real time processing over continuous, high volume data streams.

A typical application using streaming protocols is composed of a serial of computation intensive tasks, such as mathematical algorithms. These tasks can be executed in multiple parallel processing units organized in a pipelined fashion to enable higher processing throughput and flexibility. There are two noticeable features in the execution of applications using streaming protocols.

1. First and foremost, the tasks are executed iteratively. Since a data stream passes through the pipeline periodically, the same tasks shall be executed repeatedly. As a result, the communications among tasks/processors will last for the duration of the application.
2. Second, in a pipelined application using streaming protocols, there are two types of data dependencies [5] that determine the order of the tasks to be executed. That is, (i) the execution of a producer task (a task that produces the data needed by other task(s)) must precede execution of a consumer task (a task that actually consumes the data generated from preceding tasks), and (ii) a task can start its n-th iteration only if its $(n-1)$-th iteration is finished.

There are unfortunately two types of deadlocks that may occur to a pipelined application using streaming protocols on an NoC-based architecture: (1) the routing-dependent deadlocks [6,7] and (2) the message-dependent deadlocks [8]. The routing-dependent deadlocks, caused by cyclic channel dependency, can be avoided by removing any cycle that exists in the application's channel dependency graph [9]. Message-dependent deadlocks [8], caused by the dependency of different messages, on the other hand, are more complicated to deal with. As reviewed in [8], there are three major types of message dependencies, namely, request–response dependency, response–request dependency and request–request dependency.

Our focus is on the important type of message-dependent deadlock caused by request–request message dependency [8] in an NoC-based peer-to-peer streaming system. In such a system, each pair of tasks (or actors) has a virtual point to point communication path [10]. Fig. 1 shows such an example where message Y is dependent on message X. A producer task initiates a communication by sending (or pushing) a request message to its designated consumer task where this request message will be consumed. As a task is iteratively executed in an application using streaming protocols, multiple request messages can be continuously generated from the same producer task. These request messages, if they are not yet been consumed by their designated consumer task, have to be stored at various network places, such as various routers' and network interfaces' (NI) buffers. A message-dependent deadlock is created when some of these messages can never be consumed by the consumer task. Because the consumption of these messages is mutually dependent on each other's arrival because the network resources (e.g. the routers' buffers) are reserved and the messages are blocked in some network places. This request–request type message-dependent deadlock can cause severe effects on an application which uses streaming protocols as it may put the whole system into a complete stall [8].

In the literature, several message dependent deadlock avoidance/recovery methods for NoCs have been proposed, including stricting order and end-to-end flow control [8]. These approaches may either require a large number of on-chip buffers or may result in high message latency. For example, the work in [8] indicates that use of separate virtual channels for each message type can help avoid deadlocks. However, the buffer size of the additional VCs for deadlock avoidance tends to be too large, leading to poor resource utilization.

In this paper, for the first time, we have formally proved a sufficient condition that determines the minimum number of VCs actually needed to avoid the message-dependent deadlocks. These virtual channels can be shared among multiple communication flows. Following this theory, we propose the path selection and minimum VC allocation (PSMV) method to find the minimum number of VCs in order to avoid request–request type message dependent deadlocks. PSMV selects the minimal routing path and allocates non-uniform minimum number of VCs for each communication. The non-uniform VC concept is introduced in [11]. Non-uniform VC planning means that some of the channels have more VCs than others. Non-uniform VC can be used in application-specific design. PSMV could be integrated with existing application mapping algorithms, like the ones reported in [6,12–14], to obtain deadlock-free mapping results. Our experiments have revealed that for many popular applications which use using streaming protocols, such as networking and multimedia applications, the

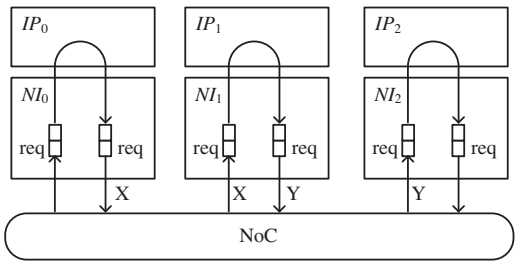

Fig. 1. An example of request–request type message dependency.

number of additional VCs needed by PSMV to avoid deadlocks is fairly modest. That is, with a modest price paid in terms of buffer area, applications using streaming protocols can run in an NoC-based system completely free of deadlock concerns, which is necessary to deliver the QoS guarantee required by these applications.

The rest of the paper is organized as follows. Section 2 introduces related work. The target application and architectural models are defined in Section 3. Section 4 provides a detailed explanation of the cause of request–request type message-dependent deadlocks followed by an illustrative example showing how this type of deadlock can be avoided with proper inclusion of non-uniform VCs. Inspired by this observation, Section 5 presents a theory concerning the minimum number of non-uniform VCs for each channel actually needed to avoid deadlocks; in the same section, an algorithm PSMV to explore this theory to find the minimum number of non-uniform VCs for each channel is described. Section 6 shows how the proposed PSMV algorithm can be integrated with an existing application mapping algorithm to produce deadlock-free mapping results. Finally, Section 7 concludes the paper.

2. Related work on deadlock resolution in NoC

Message-dependent deadlock resolution methods in NoCs can be classified into two types, deadlock avoidance and recovery. Deadlock avoidance methods aim to eliminate the emergence of deadlocks. Deadlock recovery methods allow the occurrence of deadlocks, but manage to recover from deadlock, after which the system can return to normal states.

2.1. Deadlock avoidance methods

There are mainly two types of message-dependent deadlock avoidance methods for NoCs. The first type is using separate networks (physical or virtual networks) for different types of messages, exemplified as the strict ordering method described in [8]. A major drawback of strict ordering is its requirements for too many separate networks, when applied to applications using streaming protocols. As pointed out in [8,15], when a request–request type message dependent deadlock emerges, the number of separate networks will be equal to the number of communication flows in the network.

The second type is using end-to-end flow control (e.g., the credit based (CB) method [8] and the connection then credits (CTC) method [15]). In CB, each receiver's NI keeps a buffer queue and a credit counter for each sender or predecessor. The sender NI sends a request packet to the receiver to initiate the communication. Then it waits for the response packet from the receiver. Upon receiving the request packet, the receiver NI first checks whether the receiver's buffer queue has sufficient empty slots to accomodate the message, and then it sends the empty slots info (credits) back to the sender. With credit information from the receiver, the sender can begin its data transmission towards the receiver. CB requires an individual input data buffer queue for each sender, an output data buffer queue and a credit counter for each receiver NI. The CTC [15] method is based on CB but it tries to reduce the buffer usage of CB. There is only one input data buffer queue and one request queue per NI. Therefore, an input arbiter and an output arbiter are used for different senders or receivers. The buffer has K flits, where K is a user defined parameter. Before a sender initiates a communication, it sends a request packet to the receiver. The receiver checks whether there are K empty slots in the buffer queue. If so, the receiver notifies the sender of the credit number (K). When the sender gets this credit number, it will send out a sub-message with maximum K flits at each time. Thus, if a message has M flits, both the numbers of sub-messages and response packets are $\lceil M/K \rceil$. When applied to applications which use streaming protocols, this type of avoidance method may incur additional response packets which may increase the probability of network congestion and consequently, tend to negatively impact the message latency.

2.2. Deadlock recovery methods

An alternative way to resolve message dependent deadlocksis is through deadlock recovery. In deadlock recovery methods, deadlocks are allowed to occur. Once a deadlock is detected, the deadlocked packets will be transmitted in reserved or backup channels. mDISHA [16] is a deadlock recovery method proposed for parallel computer networks. In mDISHA, a deadlock situation is first detected and then resolved by using the deadlock channel. Deadlock detection in mDISHA is realized through timers associated with each buffer queue. For a packet, when its timer reaches a certain threshold, it is assumed that a deadlock has occurred. The deadlock channel is realized by dedicated buffers in both NIs and routers. Each destination NI is augmented with additional buffers reserved for deadlocked packets. The deadlocked packet will be routed in the deadlock buffers by preempting the idle output channels until it reaches the destination node.

mDISHA has been extended [17] as a deadlock recovery method (DR) for NoCs. The deadlock recovery method in [17] differs from mDISHA as there is no preemption mechanism in each NI. The work flow of DR is as follows. For each message type, there is one deadlock virtual channel (DVC) dedicated for deadlock recovery usage in each router and NI. Access to the DVCs is controlled by an additional token network where tokens corresponding to different message types are circulating. Thus, once a deadlock has been detected at a router, the deadlocked packet will be redirected to the DVC of the message type if the corresponding token arrives at this router. After the deadlocked packet is transmitted to the destination, the token will be released and placed back again to the token network. The deadlocked packets can arrive at their destination since they are

holding the reserved path composed of the DVCs exclusively. This method, however, may have several drawbacks when applied to applications which use streaming protocols.

- It may increase message latency as the deadlock cannot be detected until a timeout threshold of the timer is reached.
- Additional buffers (i.e. DVC) are introduced in each router and NI.
- An additional network for token distribution is required.

3. Target architectural and application models

In this section, we introduce the model of an application which uses streaming protocols, followed by an architectural model that describes the underline architectures that an application using streaming protocols is operating upon. These application and architectural models are adopted for the illustration of deadlock phenomenon provided in Section 3.1.

3.1. Application model

The tasks of an application can be modeled by a communication trace graph.

Definition 1. A *communication trace graph* [14] is a directed graph $G = (A, E)$.

- Each vertex $a_i \in A$ represents a task. Each task cannot be preempted by other tasks.
- Each directed edge $e_i = (a_k, a_j) \in E$ represents the data communication from a_k to a_j. $\omega(e_i)$ defines the amount of data sent from a_k to a_j in bits per second (bps).

A *source task* is one which has no predecessor task. A *sink task* is one which has no successor task. All the tasks will be executed repeatedly for a number of iterations to process the incoming data stream. A task that needs to communicate with others is allowed to do so either at the beginning (i.e. read input data) or at the end (send output data) of an iteration. Let $(a_k \rightarrow a_j)_m$ denote the data (packets in messages) from a_k to a_j after a_k finishes its m-th iteration.

There are two types of buffers associated with a task: the input data buffers and the output data buffers.

- For any task, one input data buffer is dedicated to receive the messages from just one of its predecessors [10]. That is, the number of input data buffers needed is the same as that of the task's predecessors.
- For any task, messages for one of its successors will be stored in a dedicated output data buffer before they can be sent out [10]. That is, the number of output data buffers is the same as that of the task's successors.

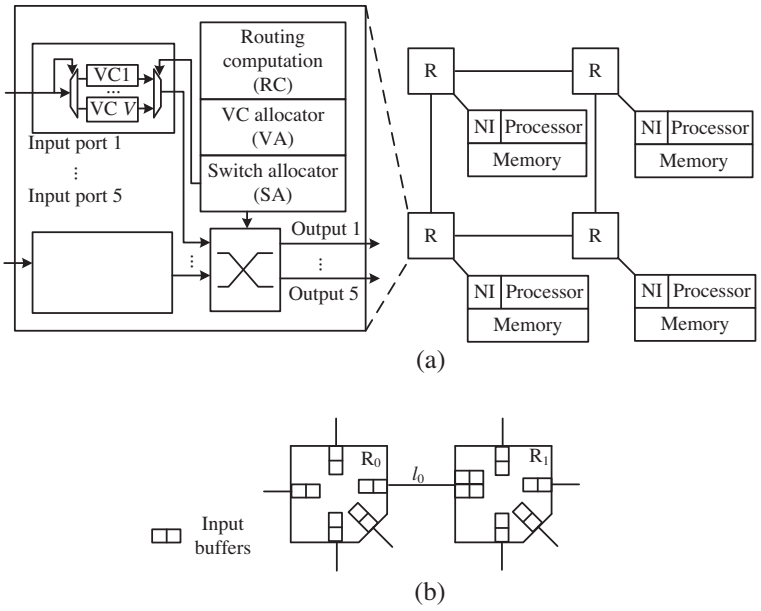

Fig. 2. (a) The NoC architectural model. A rectangle box represents a router and a circle represents a processing node. (b) Non-uniform VC example. The link l0 has two virtual channels while the other links has only one virtual channel.

All tasks are scheduled in a self timing manner [10] as follows. A task can start a new iteration if (a) it receives the data (messages) from all its predecessors and (b) its output data buffers are empty. The data (messages) are consumed (cleared) from its input edges of a task after it starts a new iteration.

3.2. Architectural model

The architectural model considered in this paper is shown in Fig. 2. Application specific NoC architectures are assumed. The NoC system under consideration is composed of $N \times N$ tiles interconnected by a 2-D mesh network. Each tile, indexed by its coordinate (x, y), where $0 \leqslant x \leqslant N-1$ and $0 \leqslant y \leqslant N-1$, is associated with one router and one local processing node.

3.2.1. Router

Fig. 2(a) shows the router architecture with wormhole switching [18]. Each message consists of a number of fixed-size packets with each carrying the needed routing information. Each packet will be further decomposed into fixed-size flits. Each router is composed of input buffers, routing computation unit, VC allocator, and switch allocator. Each input port has into one or more input VC buffers. The flits are first stored in one of these input VC buffers. If a flit at the head of an input buffer is deemed as a header flit, the routing calculation unit finds the output port that this flit shall go. The VC allocator (VA) then selects the VC within the selected output port for the flit. The switch allocator (SA) arbitrates the requests made by all the VCs competing for the switching fabric (i. e., a crossbar). The winner flits will be given the permission to transmit over the crossbar to the corresponding output link.

3.2.2. Local processing node

As shown in Fig. 2, each processing node is composed of three components: a processor, a local memory unit and a network interface (NI). The local memory unit holds the input/output data buffers (see Section 3.1) that can be used by the task allocated to this processor. The NI provides the interface between the router and the processor.

Definition 2. An *Architecture Characterization Graph* (ACG) is a directed graph $G' = (T, L)$. Each vertex $t_i \in T$ represents a tile (Fig. 2). A tile T is composed of a local processing node and a router, i.e., $T = P \cup \mathrm{R}$, where P is the set of processors and R is the set of routers. Each edge $l_i = (t_k, t_j) \in L$ represents the link between adjacent t_k and t_j. For each link l_i,

- $bw(l_i)$ defines the bandwidth provided between tiles t_k and t_j.
- $CH(l_i)$ gives the number of virtual channels of link l_i. Fig. 2(a) shows a non-uniform VC allocation [11].
- $c(l_i)$ defines the link cost of l_i, i.e., power consumption for transmitting one bit data from t_k and t_j with the number of VCs equal to $CH(l_i)$.

Definition 3. A mapping function M: $A \rightarrow P$ maps each vertex in CTG to a processor in ACG. $M(a_i)$ gives the processor which vertex a_i is mapped to. For simplicity, we assume that each processor is allocated with one task.

In this paper, we limit our discussion to regular NoC architectures which have $bw(l_i) = B$ for each $l_i \in L$, where B is a constant. $h_{k,j}$ is the set of links forming one of the shortest paths from tile t_k to tile t_j ($h_{k,j} \subseteq L$). $dist(h_{k,j})$ determines the number of elements in $h_{k,j}$, which corresponds to the hop count of the shortest path between tile t_k and tile t_j.

4. Message-dependent deadlock in applications using streaming protocols and a motivating example

In this section, we provide an example to illustrate how a request–request type message-dependent deadlock can occur, followed by important observations how such deadlock can be avoided.

Assume the CTG has three tasks a, b, and c, as shown in Fig. 3(a), has already been mapped to the ACG with a 1×3 mesh using the mapping algorithm in [6]. The three tasks (with a and b as the producers and c as the consumer) form a pipeline. Assume the pushing protocol [8] is applied; that is, a producer task is allowed to continuously push all its generated data into the network until the network is saturated. In this example, no virtual channel is used. Here tasks a and b send their generated data to c after they complete one iteration. Task c has two input data buffers in its local memory and each holds one message from one of its predecessor tasks, a or b. The NI of each tile has one input buffer which can hold a few flits. As indicated in Section 3.1, for all three tasks to operate properly, c cannot start a new iteration (say the i-th iteration) until it receives the data generated in the i-th iteration of a and b in its corresponding input data buffers. Once c starts a new iteration, its two input data buffers must have been cleared.

A problem may happen when, for example, b produces data at a higher rate than a and c. Fig. 3(b) illustrates such a situation. Suppose c has finished its $(n-1)$-th iteration and it is now waiting for $(b \rightarrow c)_n$ and $(a \rightarrow c)_n$. In the following, we will show how a cyclic dependency among the messages can be developed.

(1) Flits of $(b \rightarrow c)_{n+1}$ reserve link (1, 2), the west input buffer of R_2 and the receiving buffer of NI_2 (marked with $*$ in Fig. 3(b)).
(2) Flits of $(b \rightarrow c)_{n+1}$ cannot proceed to the input data buffer dedicated for b in c's local memory since the buffer is occupied by $(b \rightarrow c)_n$ which has not been consumed (cleared) yet.

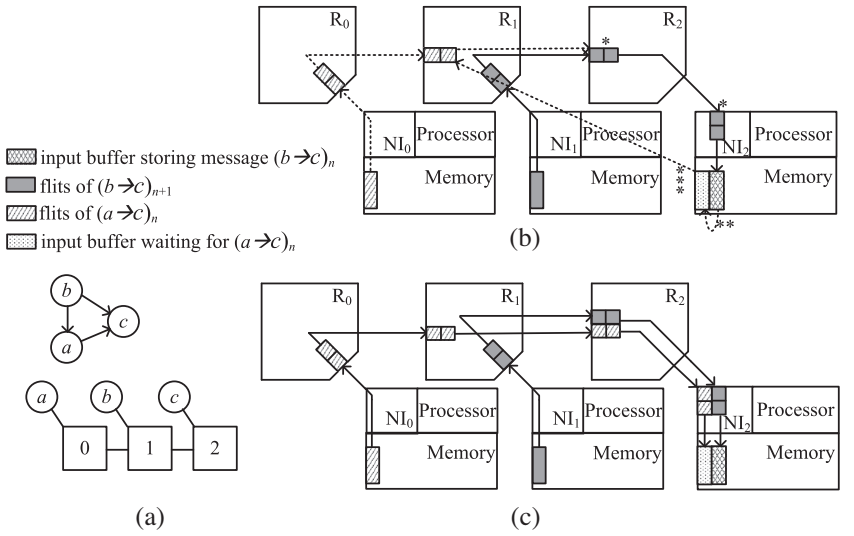

Fig. 3. (a) An example of ACG with three tiles mapped with three tasks. (b) Deadlock configuration due to request–request type message-dependency. (c) Increasing the number of virtual channels can help avoid request–request type message-dependent deadlocks.

(3) To consume $(b \rightarrow c)_n$, $(a \rightarrow c)_n$ needs to arrive at the input data buffer dedicated for a in c's local memory to allow c to start the n-th iteration. (This dependency is marked as $**$ in Fig. 3(b))
(4) In Fig. 3(b), the arrow marked with $***$ denotes that the input data buffer in c's local memory dedicated for a waits for the arrival of flits of $(a \rightarrow c)_n$.
(5) However, flits of $(a \rightarrow c)_n$ cannot proceed since the west input buffer of R_2 is already reserved by flits of $(b \rightarrow c)_{n+1}$.
(6) Up to this point, a deadlock is already formed.

Above example has revealed that a deadlock can be formed when some data flows may take all the network resources, like network buffers, consequently, depriving others from fair access to these resources. However, due to the nature of the applications using streaming protocols where tasks are iteratively executed (Section 1), mutual dependence of the consumption/arrival of the data/control messages generated across two iterations may be created. As a result, a deadlock is undesirably formed.

As a viable alternative solution, the request–request type message-dependent deadlocks can be completely avoided by adding a number of non-uniform virtual channels [11] and NI buffers. For the example shown in Fig. 3(a), one can see that the cycle that creates a deadlock in Fig. 3(b) is completely avoidable by adding (1) one virtual channel at the West direction of R_2 and (2) one input buffer at NI_2 (Fig. 3(c)). In this regards, only 2 additional buffers is needed and the size of each buffer is set to hold several flits. Also, it shall be noted that these virtual channels are not dedicated to specific types of messages. Actually, they can be shared by any communication flow.

Since in many NoC designs, virtual channels are used for helping improve network routing performance [6,7], the hardware cost of this approach for avoiding message-dependent deadlocks is well justifiable. in a simple term, to avoid request–request type message-dependent deadlocks, it is required that (1) the number of the receiving buffers of each NI is set to be the number of predecessors of the task allocated on the local processor and (2) a few additional VC buffers should be used. As proved in Section 5, these two requirements constituent a sufficient condition to avoid deadlocks of interest.

5. Deadlock avoidance using virtual channels

In this section, the sufficient condition for avoiding request–request type message-dependent deadlocks is proved. Then, we show that finding the minimum number of virtual channels needed for deadlock-freeness is NP-complete. Thus an MILP-based approximation algorithm is proposed to solve this sub-problem. The second sub-problem finds the minimum number of non-uniform VCs for each links.

5.1. Sufficient condition for deadlock-avoidance with virtual channels

Assume an application which uses streaming protocols given as an CTG (Section 3) has already been mapped to an NoC architecture modeled by an ACG (Section 3). We also assume that (i) the routing algorithm used is deadlock-free, and (ii) messages are delivered in-order and lossless.

Lemma 1. *For any task in an application which uses streaming protocols allocated to a processor, after it finishes the execution of its n-th iteration, its input data buffers are either empty or they are holding messages from its predecessors' (n + 1)-th iteration.*

Proof (Proof by contradiction). According to the application model (Section 3.1), after the task finishes its n-th iteration, the messages from its predecessors' $(n+i)$-th iteration, $i \leqslant 0$, must have been consumed.

Assume to the contrary that there exists a task, say task a, which after completing its n-th iteration, holds the message generated by one of its predecessors, say a_j, from a_j's $(n+i)$-th iteration with $i \geqslant 2$ in one of its input data buffers. Since the messages are delivered in-order, message $(a_j \rightarrow a)_{n+1}$ must have arrived at the input data buffer of a. This contradicts to the application model assumed in Section 3.1 in which there is only one input data buffer holding one message from each predecessor. Hence, the lemma holds. □

From Lemma 1, we can derive the following corollary.

Corollary 1. *For any task in an application using streaming protocols allocated to a processor, if it has not started its (n + 1)-th iteration, any message generated from its predecessors' (n + i)-th iteration (i ⩾ 2) cannot reach the task's input data buffer.*

Lemma 2. *There exists one virtual path between any two communicating tasks so that all messages will arrive at their destination tasks, provided that (1) each link is shared by communications less than or equal to the number of virtual channels and (2) the number of receiving buffers at the NI of each tile is equal to the number of predecessors of the tasks which are mapped onto this tile.*

Proof. Each message initiated from the source tile (that the source task is mapped to) has to travel through a path to reach the destination tile (that the destination task is mapped to). For the sake of this proof, this path can be divided into two segments: (i) from the router at the source tile to the router at the destination tile, and (ii) from the router at the destination tile to the input data buffer of the destination task. We will show that a virtual path always exists in these two segments.

(i) Each message can arrive at the router at the destination tile.
If a link has V VC's and is shared by no more than V communication flows, no message belonging to one communication flow can be blocked by messages from other communication flows (Fig. 4). Hence, there exists a virtual path from the router at the source tile to the router at the destination tile. Through this virtual path, each message can arrive at the router connecting to the destination tile.

(ii) Each message can proceed to the input data buffer of the destination task.

According to our architectural model defined in Section 3.2, at each tile that a task is mapped onto, the number of input data buffers at the local memory and the number of receiving buffers at the NI are the same as the number of the task's predecessors. This will ensure that each message (generated from the source task) once reaching the router at the destination tile will arrive at the input data buffer dedicated for its predecessor, if there is free buffer space.

Putting these two segments together, there exists a dedicated virtual path between two communicating tasks. This will guarantee that all messages will arrive at their destination tasks.

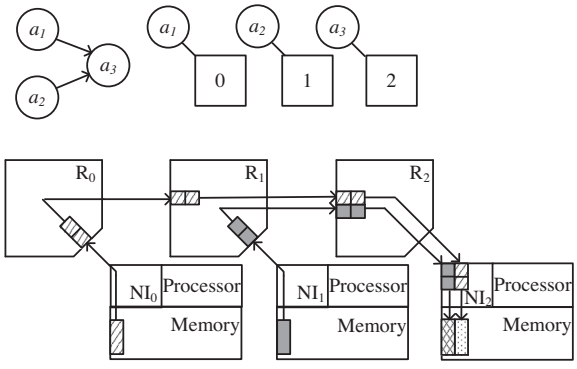

Fig. 4. An example illustrating Lemma 2. The channel between routers R_0 and R_1 has two VCs and the NI_2 has two input buffers. There is a virtual path either from a_1 to a_2, or from a_1 to a_3.

For example, in Fig. 4, the channel between routers R_0 and R_1 has two VCs. Task a_3 has two predecessors, a_1 and a_2. Thus the NI_2 has two input buffers. For each message from a_1 to a_3 and from a_2 to a_3, there exists a dedicated virtual path. With Lemma 2, we can have the following lemma. □

Lemma 3. *For an application which using streaming protocols given by a CTG with tasks allocated to the processors in ACG, each task can start a new iteration as long as the input data are continuously admitted into the system and the output data are continuously extracted from the system.*

Proof. (*Proof by contradiction).* We first consider the processor allocated with the sink task (the task with no successor task) $a_k \in A$ which has N predecessors $(a_k^1, \ldots, a_k^N)$. If there are multiple sink tasks, a dummy task node can be added which has all sink tasks as its predecessors. Assume to the contrary that a_k has finished its n-th iteration ($n \geqslant 0$, $n = 0$ refers the state before the first iteration is executed) but it cannot start its $(n+1)$-th iteration. Since a_k is a sink task from which output data are continuously extracted, the reason that it cannot start its $(n+1)$-th iteration must be the missing of at least one message generated from one of its predecessors in the $(n+1)$-th iteration.

Assume a_k^m is one of a_k's predecessors whose message $(a_k^m \rightarrow a_k)_{n+1}$ has not arrived at a_k's input data buffer. According to Lemma 2, if message $(a_k^m \rightarrow a_k)_{n+1}$ is generated, it will arrive at a_k's input data buffers. Thus, the only possible reason is that a_k^m has not started its $(n+1)$-th iteration yet. Since the sink task a_k has finished its n-th iteration, all other tasks (including a_k^m) must have finished their n-th iteration. Hence, the output buffers of a_k^m are empty. The reason that a_k^m cannot start its $(n+1)$-th iteration must be the missing of at least one message generated from one of a_k^m's predecessors in the $(n+1)$-th iteration.

Following the similar reasoning, at least one reverse path from the processor allocated with the sink task to the processor allocated with a source task (say a_s) can be found and all the processors on the path are allocated with tasks which have finished their n-th iteration but cannot start their $(n+1)$-th iteration. Particularly, a_s has finished its n-th iteration but cannot start its $(n+1)$-th iteration. However, the output data buffers of a_s must be empty because all a_s's successor tasks have finished their n-th iteration. With the input data continuously provided to the system, a_s can start its $(n+1)$-th iteration. This is contradicting to the above reasoning.

Therefore, the sink task a_k can start its $(n+1)$-th iteration after finishing its n-th iteration.

Based on this result, following similar approach we can progressively prove that the predecessor tasks of a_k and other tasks can start their $(n+1)$-th iteration after finishing their n-th iteration. Hence, the lemma holds. □

Theorem 1. *Consider an application represented as an CTG and mapped to an NoC architecture represented as an ACG in which (i) the number of communications allocated on each link is not larger than the number of the VCs of the link and (ii) the number of receiving buffers at the NI of each tile is set to be equal to the number of the predecessors of the tasks which are mapped to the tile. No request–request type message-dependent deadlock can ever be created.*

Proof (Proof by contradiction). Assume to the contrary that a request–request type message-dependent deadlock as described in Section 4 is created. The deadlock will prevent some messages from reaching their destinations. As such, these destination tasks cannot receive all the input data generated from their predecessors. Consequently, none of these tasks can start a new iteration. This is a contradiction to Lemma 3. Hence, the theorem holds. □

5.2. Minimum non-uniform virtual channel algorithm

Once an CTG is mapped to an ACG, the routing path allocation step follows, which determines how each communication flow is routed through the physical links. According to Theorem 1, to avoid message-dependent deadlocks, the number of communications allocated on each physical link cannot exceed the number of VCs used by each link. On the other hand, the number of VCs used shall be minimized to reduce the overhead in terms of power consumption and area. Hence, it is vital to find the minimum number of non-uniform VCs needed to satisfy Theorem 1.

According to Theorem 1, the number of VCs needed for each link depends on both the mapping result and the routing path selection for each communication. Assume the mapping result is given, we propose the path selection and minimum VC allocation method to find the minimum number of non-uniform VCs for each link. The PSMV method consists of two major steps aiming to solve two sub-problems: (P1) find the minimized maximum (min_max) number VCs needed among all the links (shortened as the MinVC problem), (P2) based on the solution to the MinVC problem, determine the minimum number of VCs for each individual link. In the following, the MinVC problem is first defined and solved first.

The **MinVC problem** (P1) is stated as follows. Given an ACG(T, L) and an CTG(A, E) which is mapped to the ACG, for each non-local communication $e_i = (a_j, a_k)$ in CTG, find a routing path in ACG among all the possible minimal routing paths between $M(a_j)$ and $M(a_k)$ if $M(a_j) \neq M(a_k)$, such that the maximum number of VCs needed among all the links is minimized, i.e.,

$$\text{Min}\{V\}$$

satisfying,

$$\forall\, l_m, \quad V \geqslant \sum_{e_i=(p_k,p_j)\in E} g(l_m, h_{M(a_k),M(a_j)}) \tag{1}$$

$$\forall\, l_m, \quad F \cdot B \geqslant \sum_{e_i=(p_k,p_j)\in E} \omega(e_i) \cdot g(l_m, h_{M(a_k),M(a_j)}) \tag{2}$$

where $M(a_i)$ is the tile that a_i is mapped to, F is the bandwidth factor and $0 < F \leqslant 1$, and $g(l_i, h_{M(a_k),M(a_j)}) = \begin{cases} 1 \text{ if } l_i \in h_{M(a_k),M(a_j)} \\ 0 \text{ if } l_i \notin h_{M(a_k),M(a_j)} \end{cases}$

The condition given by (1) ensures that each link l in ACG is shared by at most V communication flows, while the condition given by (2) ensures that the total bandwidth requirement of all communication flows sharing each link l does not exceed the capacity of link l. The factor F ensures that the link should not be fully reserved to avoid possible congestions.

Next we will first show that the MinVC problem is NP-complete. Then we will present an MILP-based solution.

Theorem 2 [19]. *The knapsack problem defined below is NP-complete.*

Given a finite set U, a size $s_u \in Z^+$ and a weight $v_u \in Z^+$ for each $u \in U$, a size constraint $B \in Z^+$, and a value goal $K \in Z^+$, is there a subset $U' \subseteq U$ such that $\sum_{u\in U'} s_u \leqslant B$ and $\sum_{u\in U'} v_u \geqslant K$.

Theorem 3. *The decision version of the MinVC problem is NP-complete.*

The proof sketch is listed below.

The decision version of the MinVC problem is to decide whether there exist the minimal routing paths in the ACG for all non-local communications in the CTG, while all the resource constraints are satisfied and no more than V VCs are needed at each router.

We will prove this theorem by restricting [19] the decision version of the MinVC problem to its instances with $\omega(e_i) = 1$ for all e_i, and $V = F{\cdot}B$. Thus, conditions (1) and (2) of the MinVC problem are algebraically identical. This restriction, from general MinVC to its restricted version, takes $O(|L| + |E|)$ time, where $|L|$ is the number of links in the ACG. The restricted MinVC problem is equivalent to finding the minimum cost unsplittable flows [20] for a set of $|E|$ communication flows $\{1, \ldots, |E|\}$, each flow sending from a source node s_i to its destination node d_i with demand $\omega(e_i)$ for $i \in \{1, \ldots, |E|\}$. The knapsack [19] problem can thus be viewed as a special case of the restricted MinVC problem, as shown in Fig. 5.

In Fig. 5, for each item in knapsack problem [19], there is a corresponding flow i (i.e., an edge e_i in the CTG which is a non-local communication) whose demand $\omega(e_i)$ is equal to the size of each item. The cost of each direct edge from nodes s to d_i is set to w_i/q_i where w_i and q_i represent the weight and size of the i-th item, respectively. Costs of all the other edge are set to 0. The capacity of the edge from nodes s to v is equal to the capacity of the knapsack while assuming all the other edges have infinite capacity. Therefore, the minimum cost flows from all s_i to d_i, $i \in \{1, \ldots, |E|\}$, upon satisfying all the demands, lead to an optimal solution to the knapsack problem and vice versa.

The knapsack problem is NP-complete [19], and so is the MinVC problem which takes the knapsack problem as a special case.

The MinVC problem can be approximated by formulating it as a mixed integer linear programming problem given in Fig. 6. As such, a linear programming solver, such as the lp_solve [21] can be used.

In Fig. 6, MP_i is the set of all minimal paths between the two tiles mapped by the two terminal vertices of edge e_i, and $PATH_{i,j}$ is the j-th minimal path in MP_i.

$$f_{i,k} = \begin{cases} 1 & \textit{if } e_i \textit{ takes the } k-\textit{th minimum path} \\ 0 & \textit{otherwise} \end{cases}.$$

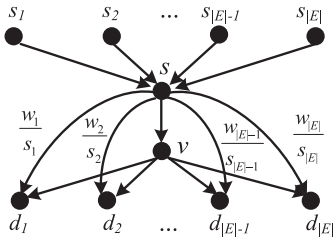

Fig. 5. Formulation of the Knapsack problem [19] as the restricted MinVC problem.

MinVC(M, MP)

Input: (1) M: a mapped result

(2) MP: a set of all minimum paths for all communication flows

Output: (1) V: the number of VCs needed for all routers

(2) $\{ f_{i,k} \}$: the set of variables indicating the minimal paths selected for all e_i

Procedure body:

{

call lp_solve to solve the following equations:

Objective: min (V) (3)

Constraints:

$$\sum_{k \in |MP_i|} f_{i,k} = 1, \forall e_i \ (MP_i \text{ is the set of all minimal path set for } e_i); \quad (4)$$

$$\sum_{e_i} \sum_{l \in PATH_{i,j}, k \in |MP_i|} f_{i,k} \le V, \forall link\ l; \quad (5)$$

$$\sum_{e_i} \sum_{l \in PATH_{i,j}} \omega(e_i) \times f_{i,k} \le B, \forall link\ l; \quad (6)$$

}

Fig. 6. The solution to the MinVC problem.

Eq. (4) sets a constraint that only one of the minimal paths will be allocated for each communication. Eq. (5) sets a limit that each link will not accommodate more than V communications. Eq. (6) represents the bandwidth constraint of each link. Our experiment has shown that when lp_solve [21] is used to solve MinVC, the running time is less than 0.1sec for a 6×6 mesh-based NoC and a little over 0.1 s for an 8×8 NoC (obtained from a PC with one Intel Core2 P8600 2.4GHz processor and 2GB RAM).

The solution to P2 is based on the solution to the MinVC problem which provides the min–max number of VCs among all the links and the minimal routing paths selected for all communications. Based on these selected minimal routing paths, the minimum number of non-uniform VCs for each individual link is determined by accumulating the number of flows that use the link. This sub-problem has a complexity of O($|L||E|$).

The PSMV method is listed in Fig. 7.

6. Experiments

To evaluate the deadlock avoidance method proposed in previous section, we have conducted experiments on four applications which use streaming protocols, including: (1) a synthetic application, (2) orthogonal frequency-division multiplexing (OFDM) [22], (3) multi-windows display (MWD) [1] and (4) video object plane decoder (VOPD) [1]. To generate the mapping result for each application, we apply a genetic-based mapping algorithm [13] integrated with the power estimation step which calls up the PSMV routine. Based on the mapping result, the Noxim simulator [23] is used to obtain the total communication power. The Noxim simulator is modified, with changes listed below, to make it suitable for simulating applications which use streaming protocols.

- The processing elements of the simulator are modified so that they are in compliance with the model defined in Section 3.1.
- The traffic generator is modified so that it is able to model the dependency of tasks.
- Virtual channels are supported in the simulation.

In the following, we will first describe the mapping and path allocation procedures followed by the simulation settings and the experimental results.

PSMV(M)

Input: M: a mapped result // the SDFG of an application is mapped to the NoC architecture represented by an ACG (Section 3)

Output: (1) $\{CH(l_j)\}$: the set of variables indicating the number of VCs for all links in NoC (Section 3.2)

(2) $\{h_{m,n}\}$: the set of minimal routing paths for the communications in SDFG (Section 3.2)

Procedure body:

{

Var: V: the min_max VCs among all the links returned by **MinVC**

$MP = \varnothing$; // MP is the union of all minimal routing paths

$\{f_{i,k}\}: f_{i,k} = \begin{cases} 1 \text{ if } e_i \text{ takes the } k-th \text{ minimum path} \\ 0 \text{ otherwise} \end{cases}$ (section 5.2)

$CH(l_i)$ is initialized to 0 for all $l_i \in L$

for each edge $e_i = (a_s, a_t)$ in the SDFG { // assume a_s, a_t are mapped to tiles t_m and t_n in ACG ($t_m \neq t_n$)

find all the minimal routing paths between t_m to t_n and insert into MP_i;

}

$MP = \bigcup_{i=1}^{|E|} \{MP_i\}$;

Get V and $\{f_{i,k}\}$ by calling **MinVC**(M, MP);

for each edge $e_i = (a_s, a_t)$ in the SDFG { // assume a_s, a_t are mapped tiles t_m and t_n in ACG ($t_m \neq t_n$)

for $k = 1$ to $|MP_i|$ {

if $f_{i,k}$ is non-zero {

$h_{m,n} = PATH_{i,k}$;//communication flow from tile t_m to t_n takes the k-th minimal routing path in MP_i

}

}

}

for each link l_k, if $l_k \subset PATH_{i,j}$, $CH(l_i) = \sum_i \sum_j f_{ij}$

}

Fig. 7. The PSMV algorithm.

6.1. Mapping algorithm

In our experiments, the mapping process is based on an existing mapping algorithm [13], which uses the genetic framework presented in [24]. The flow of the mapping process is shown in Fig. 8. An intermediate mapped result (named an individual as in genetic algorithms) is generated by mating and crossover on existing intermediate mapped results. After finding an intermediate mapping result, the Calculate_Cost procedure is called (within the procedure of fitness calculation) to estimate the power consumption of the mapping result. The Calculate_Cost procedure (Fig. 9) calls the PSMV procedure to obtain the minimum number of non-uniform VCs needed to avoid message-dependent deadlocks and select the minimal routing paths for all communications.

In order to avoid other types of deadlock, e.g., the routing dependent deadlock, we use the technique in [7] as follows. After getting the routing paths, we can construct an application specific channel dependency graph (ASCDG). If this ASCDG has cycles, we can increase the number of virtual channels to avoid the routing-dependent deadlocks [7].

6.2. Simulation settings

For each application, the mapping algorithm shown in Fig. 8 is applied for all three schemes. The mapping results of all the four applications are shown in Fig. 10.

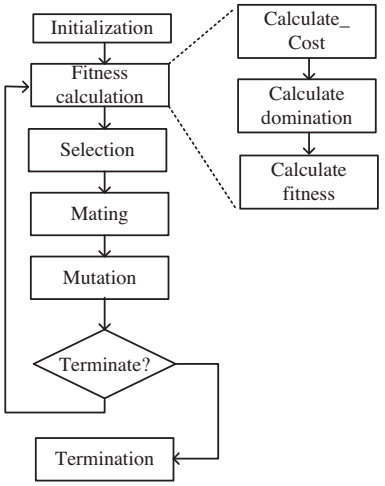

Fig. 8. Overall process of the mapping algorithm [13,24].

Calculate_Cost (M)

Input: M: a mapped result // the SDFG of an application is mapped to the NoC architecture represented by an ACG (section 3)

Output: C: power consumption estimated from the mapping result M

Procedure body:

{

Get the function $CH(l_i)$ and $\{h_{m,n}\}$ (Section 3.2) by calling **PSMV**(M);

// $CH(l_i)$ is a function returns the number of VC for a given link in NoC and

// $\{h_{m,n}\}$ is the set of minimum routing paths for the non-local communications in SDFG

return $C = \sum_{e_i=(a_j,a_k)\in E, M(a_j)\neq M(a_k)} \sum_{l_k \subset PATH_{i,m}} bw(e_i) \times dist(h_{M(a_j),M(a_k)}) \times c(l_k)$;

}

Fig. 9. The Calculate_Cost algorithm used in the mapping process.

In terms of buffer overhead, the number of additional buffers and the ratio of additional buffers for PSMV, CTC [15], and DR [17] over the baseline configuration are compared. The baseline configuration for a given NoC is that, each port of routers has one VC buffer and each NI has one input buffer. The message size is set to be 1024 bits. The flit size is set to be 128 bits and each buffer can hold 4 flits. In PSMV, the number of buffers in each tile's NI is set equal to the number of predecessors of the task allocated to the tile. The number of VCs in each link is found by applying the PSMV method. Each router is configured with one deadlock virtual channel (DVC) and so is each NI. The size of each DVC is set to hold 4 flits. Each router buffer in DR is configured with a timer. The threshold of deadlock detection timer in DR is set to be 100 cycles. In CTC, each NI has a data buffer queue and a request buffer queue. Since the size of elements in the request buffer queue is much smaller than that in data buffer queue, the request buffer queue is ignored in the comparison. We assume that DR and CTC use XY routing and PSMV uses the routing paths shown in Fig. 10. The routing paths in Fig. 10(c) are found by the algorithm in Fig. 7. The average message latency values of the three methods are compared in Fig. 12.

6.3. Buffer and power overhead

In this section, we report the experimental results of PSMV, CTC and DR in terms of (1) buffer overhead and (2) message latency. Using the PSMV method, the number of non-uniform minimum of VCs is obtained for each application. For the synthetic application shown in Fig. 10(a), links (28, 20), (36, 28), (44, 36) require 4, 3, and 2 VCs, respectively, while all the remaining links require only one VC. For OFDM shown in Fig. 10(b), the link connecting the tiles mapped with tasks *ROTOR* and *MIMO dec* (marked by star) needs two VCs while all the remaining links request only one VC. For MWD and VOPD shown in Fig. 10(c) and (d), no link needs more than one VC if PSMV is used.

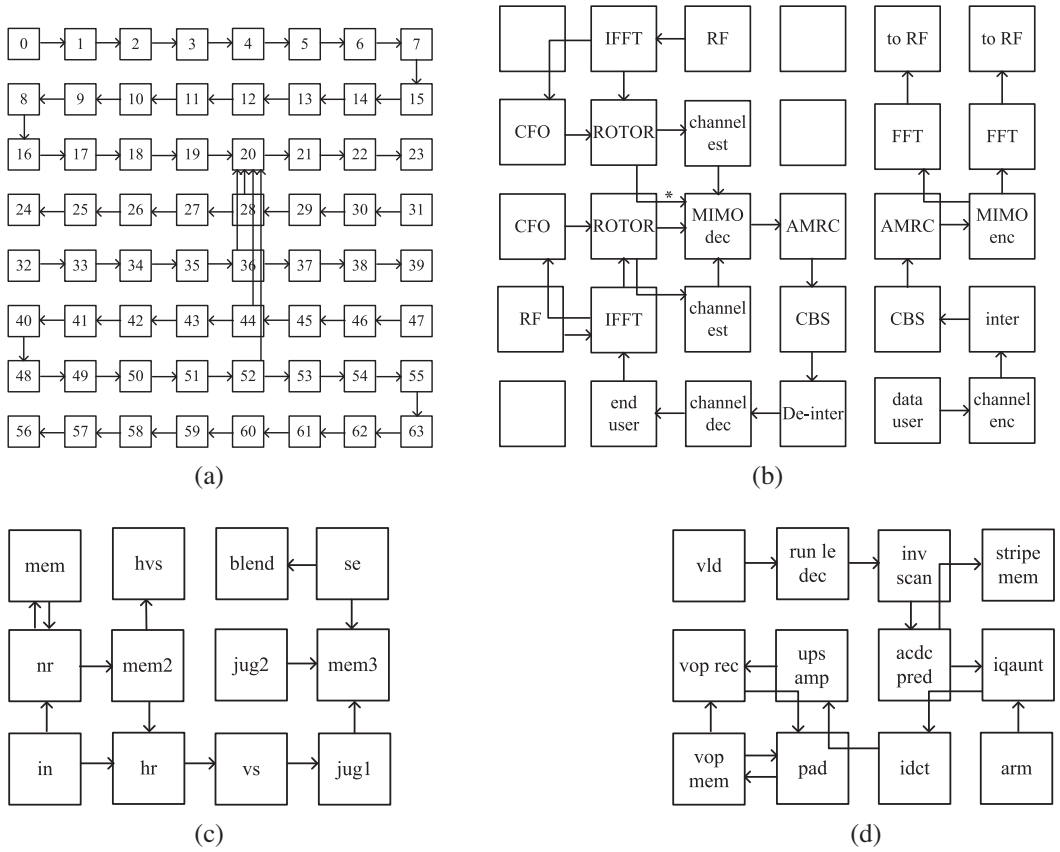

Fig. 10. The CTG and mapping of (a) a synthetic application, (b) orthogonal frequency-division multiplexing (OFDM), (c) multi-windows display (MWD), and (d) video object plane decoder (VOPD).

Table 1 shows the buffer overhead of all three methods over the baseline configuration. Since CTC does not need any additional buffer space, the number of additional buffers in CTC is 0 and thus not included in this table. From the table, we can see that, the ratio of additional buffers of DR over baseline is constant and relatively large as one additional deadlock virtual channel is added to each router and each NI in DR. PSMV, on the other hand, has a maximum of 5% buffer overhead over the baseline configuration for the four applications.

In Fig. 11, the power and area savings of PSMV over DR and using separate virtual networks (VNs) are compared. In Fig. 11, we can see that, the increase in power and area in DR over PSMV is within 7%. This is caused by the additional buffers in deadlock virtual channels. In DR, each router is increased only by one additional deadlock virtual channel. However, if separate virtual networks (VNs) are used, the number of virtual channels should be uniform, which causes the increase of the number of buffers. As in [8], the number of VNs is the same as the number of the communications for request–request type messages.

We can set the number of VNs to be the maximum number of virtual channels. For the synthetic application, four VNs are used because the South input of tile 20 needs four VCs as in Fig. 10 (a). Thus, all links will be configured with four virtual channels. Fig. 11 shows that for the synthetic application, the increase in power and area of VN over PSMV is over 20%. Similarly, OFDM needs two separate VNs as the input channel to *MIMO dec* needs two VCs as in Fig. 10(b). The increase in power and area of VN over PSMV is over 12% for OFDM as in Fig. 11. MWD needs only one VN as no link needs more than one VCs in Fig. 10(c). The power and area of VN are the same with PSMV for MWD as in Fig. 11. VOPD needs two VNs if XY routing algorithm is used. Because if XY routing algorithm is used, *pad* needs two VCs for the West input port. The increase in power and area of VN over PSMV is over 12% for VOPD as in Fig. 11.

6.4. Network performance

Fig. 12 compares the average message latency of PSMV, CTC and DR for all four applications under the injection rate range (in terms of flits/cycle). For this group of experiments, the bandwidth constraint is relaxed [1]. For all four benchmarks, PSMV has the lowest message latency by avoiding deadlocks with additional buffers. As such, PSMV does not incur waiting for flow control feedback (as in CTC [15]), or blocking when deadlock happens (as in DR [17]).

Table 1
The additional buffer number and ratio of PSMV and DR.

Benchmarks	PSMV			DR		
	Number of additional router buffers	Number of additional NI buffers	Total percentage of additional buffers (%)	Number of additional router buffers	Number of additional NI buffers	Total percentage of additional buffers (%)
Synthetic	6	4	2.5	64	64	28
OFDM	1	5	3.3	30	30	28
MWD	0	2	2.7	12	12	28
VOPD	0	4	5	12	12	28

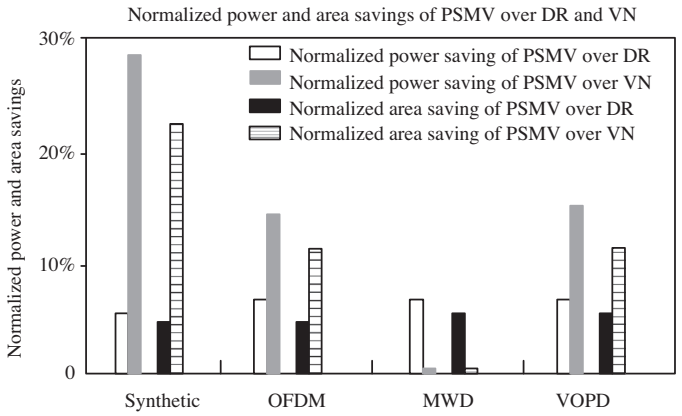

Fig. 11. Power and area savings of PSMV over DR and VN.

For the synthetic application (Fig. 12(a)), when injection rate is relatively low, CTC gives the highest message latency. In CTC, additional response packets are needed to flag the empty slots at the receiver side. However, this method tends to have higher level of congestion particularly when the injection rate is high. Also one can see that when the injection rate is high, DR has the worst performance in terms of latency. The saturation point of DR is reached earlier than PSMV (at injection rate of 0.5 in Fig. 12(a)), as more packets are injected, deadlocks are more likely to happen in DR. Because from Fig. 10(a), we can see that communications from tiles 28, 36 and 44 have contests for the links connecting to tile 20 in DR if no VC is used. In addition, in DR, the message latency is increased due to the latency caused by the deadlock detection timer in detecting deadlocks when the injection rate is high. At the injection rate of 0.6 (flits/cycle), the message latencies of CTC and DR over that of PSMV are 1.4× and 1.6×, respectively.

In Fig. 12(b), for OFDM (Fig. 10(b)), the message latency of CTC is the highest. As observed from the experiments, in case of DR, deadlocks do not occur frequently because only the communications marked by ∗ in Fig. 10(b) contest for the link connecting tile *ROTOR* and *MIMO dec*. The additional flow control packets in CTC contribute to the congestions when the injection rate is high. When the inject rates are larger than 0.6 (flits/cycle), the message latencies of CTC and DR over that of PSMV are 1.5× and 1.3×, respectively.

In Fig. 12(c), for MWD (Fig. 10(c)), the latencies of DR and PSMV are the same because no deadlock is observed in DR and PSMV. The paths resulted from XY routing (in DR) is the same as the path selection in Fig. 10(c). Thus, the performance of DR and PSMV are the same. CTC results in the highest latency due to additional flow control packets needed.

As shown in Fig. 12(d), for VOPD (Fig. 10(d)), the latencies of DR and PSMV are almost the same when the injection rate is small because deadlocks are not frequently observed in DR. When the injection rate is high, no deadlock is observed in PSMV, while deadlocks can trouble DR. The reason is that, the path selection in Fig. 10(d) helps avoid deadlock besides the additional VCs. On the other hand, DR uses XY routing which results in contests for link connecting *acdc pred* and *iquant* (see Fig. 10(d), by the communication from *acdc pred* to *iquant* and from *acdc pred* to *stripe mem*). Thus, when the injection rate is high, the message latency of DR becomes larger than PSMV due to deadlocks in DR. From the comparison of VOPD and MWD, we can see that, path selection can help avoid deadlocks in addition to the inclusion of non-uniform VC buffers.

From the above comparisons, performance of the three deadlock resolution methods is summarized as follows. CTC has the smallest number of additional buffers while DR has the largest number of additional buffers. The number of additional

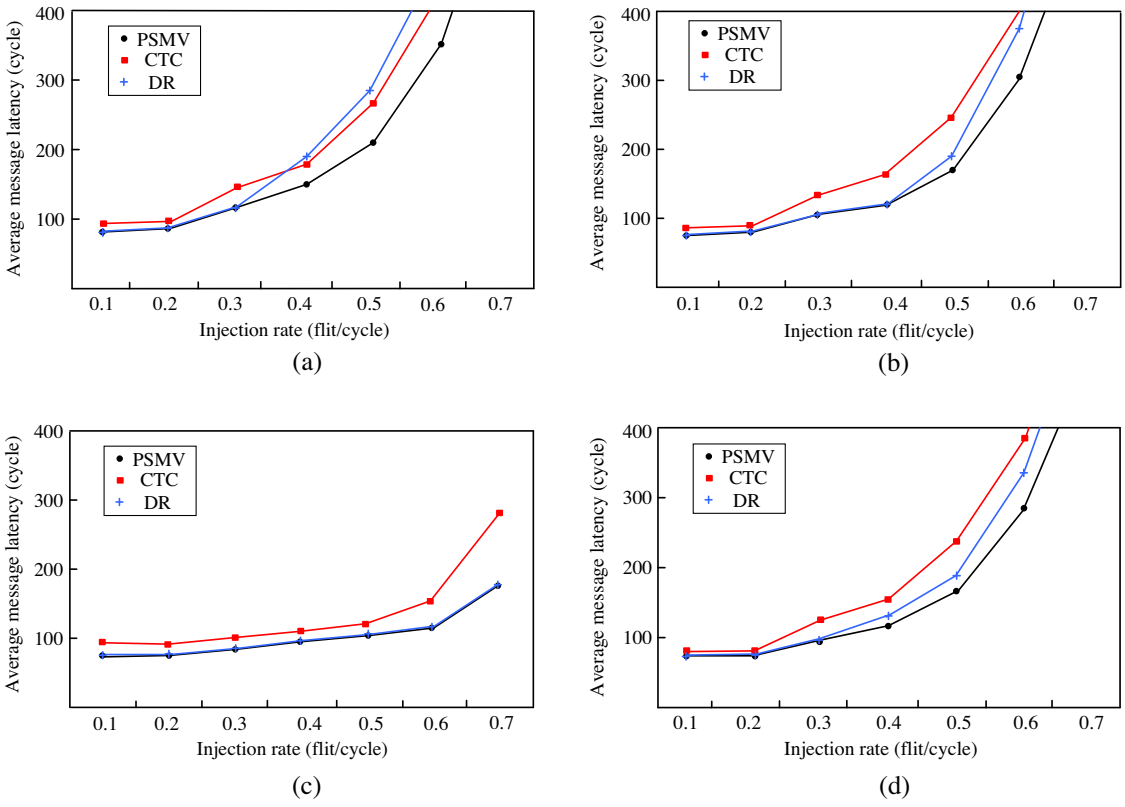

Fig. 12. The message latency of (a) the synthetic application, (b) orthogonal frequency-division multiplexing (OFDM), (c) multi-windows display (MWD) and (d) video object plane decoder (VOPD).

buffers needed in PSMV depends on the path selection result and the buffer overhead is relatively small (within 5% for the four benchmarks). In term of message latency, consistent with our expectation, PSMV shows the lowest message latency among all three methods. When the injection rate is low, CTC has the highest message latency among the three methods because of the additional flow control packets. When deadlocks occur more frequently, e.g., more communications are contesting for a single link, DR has the worst performance in terms of latency at higher injection rate. The reason might be that, when deadlocks occur frequently, each deadlocked message can only be transmitted after the deadlock is detected with the deadlock timer expries. Consequently, longer message latency is resulted.

7. Conclusion

In this paper, a practical method was proposed to avoid the request–request type message-dependent deadlock problem that otherwise can severely impact the performance of a pipelined application with streaming protocols on an NoC architecture. First, it was shown in this paper that the request–request type message-dependent deadlocks can be avoided by adding the right number of non-uniform virtual channels (VCs) to specific links based on routing path selection. A sufficient condition was formally proved which determines the minimum number of non-uniform VCs for each link actually needed to obtain such a deadlock-free NoC designs. Next, finding the minimum number of non-uniform VCs for each link was solved by the PSMV method, which can be divided into two sub-problems. The first sub-problem is to minimize the number of the maximum VCs (MinVC) among all the links. We proved the MinVC problem is NP-complete and thus proposed an MILP-based solution. The second sub-problem tries to find the minimum number of VCs for each individual link based on the solution of the first sub-problem. PSMV can be integrated with any mapping algorithm whenever a deadlock-free design is desired. Experiments based on four stream applications have confirmed that deadlocks are avoided with modest number of additional buffers in PSMV. Compared with existing two message dependent deadlock resolution methods, CTC and DR, PSMV shows the lowest message latency.

Acknowledgment

This work was supported in part by NSF under grant no. ECCS-0702168 and Chinese NSF under grant no. 60873112.

References

[1] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, G. De Micheli, NoC synthesis flow for customized domain specific multiprocessor Systems-on-Chip, IEEE Trans. Parallel Distrib. Syst. 16 (2005) 113–129.
[2] R. Rajkumar, L. Sha, J.P. Lehoczky, Real-time synchronization protocols for multiprocessors, in: Proc Real-Time Systems Symp, 1988, pp. 259–269.
[3] A.H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, M. Mousavi, Throughput analysis of synchronous data flow graphs, in: Proc Application of Concurrency to System Design(ACSD), IEEE, 2006, pp. 25–34.
[4] N.K. Kavaldjiev, A run-time reconfigurable Network-on-Chip for streaming DSP applications, Phd thesis, University of Twente, 2007.
[5] M. Ruggiero, A. Guerri, D. Bertozzi, M. Milano, L. Benini, A fast and accurate technique for mapping parallel applications on stream-oriented MPSoC platforms with communication awareness, Int. J. Parallel Program. 36 (2008) 3–36.
[6] J. Hu, R. Marculescu, Energy-and performance-aware mapping for regular NoC architectures, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 24 (2005) 551–562.
[7] M. Palesi, R. Holsmark, S. Kumar, V. Catania, Application specific routing algorithms for networks on chip, IEEE Trans. Parallel Distrib. Syst. 20 (2009) 316–330.
[8] A. Hansson, K. Goossens, A. Radulescu, Avoiding message-dependent deadlock in network-based Systems-on-Chip, VLSI Design 2007 (2007) 1–10.
[9] J. Duato, S. Yalamanchili, L. Ni, Interconnection Networks an Engineering Approach, Morgan Kaufmann, 2002.
[10] M. Bekooij, R. Hoes, O. Moreira, P. Poplavko, M. Pastrnak, B. Mesman, J.D. Mol, S. Stuijk, V. Gheorghita, J. van Meerbergen, Dataflow analysis for real-time embedded multiprocessor system design, Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices 3 (2006) 81–108.
[11] T.C. Huang, U.Y. Ogras, R. Marculescu, Virtual channels planning for networks-on-chip, in: Proc 8th Int'l Symp Quality Electronic Design IEEE, 2007, pp. 879–884.
[12] R. Marculescu, U.Y. Ogras, L.S. Peh, N.E. Jerger, Y. Hoskote, Outstanding research problems in NoC Design: system, microarchitecture, and circuit perspectives, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 28 (2009) 3–21.
[13] G. Ascia, V. Catania, M. Palesi, Mapping cores on network-on-chip, Int. J. Comput. Intell. Res. 1 (2005) 109–126.
[14] X. Wang, M. Yang, Y. Jiang, P. Liu, A power-aware mapping approach to map IP cores onto NoCs under bandwidth and latency constraints, ACM Trans. Archit. Code Optim. 7 (2010) 1–30.
[15] N. Concer, L. Bononi, M. Soulié, R. Locatelli, L.P. Carloni, CTC: an end-to-end flow control protocol for multi-core systems-on-chip, in: Proc 3rd ACM/IEEE Int'l Symp on Networks-on-Chip, IEEE, 2009, pp. 193–202.
[16] Y.H. Song, T.M. Pinkston, A progressive approach to handling message-dependent deadlock in parallel computer systems, IEEE Trans. Parallel Distrib. Syst. 14 (2003) 259–275.
[17] A. Lankes, T. Wild, A. Herkersdorf, S. Sonntag, H. Reinig, Comparison of deadlock recovery and avoidance mechanisms to approach message dependent deadlocks in on-chip Networks, in: Proc ACM/IEEE Int'l Symp Networks-on-Chip, IEEE, 2010, pp. 17–24.
[18] W.J. Dally, B. Towles, Principles and practices of interconnection networks, Morgan Kaufmann, 2004.
[19] M.R. Garey, D.S. Johnson, Computers and Intractability: a guide to the theory of NP-completeness, Freeman, San Francisco, 1979.
[20] M. Skutella, Approximating the single source unsplittable min-cost flow problem, Math. Program. 91 (2002) 493–514.
[21] lp solve 5.5 available: lpsolve.sourceforge.net/5.5/.
[22] T264 available: www.sourceforge.net/projects/t264/.
[23] Noxim available: www.sourceforge.net/Noxim.
[24] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, in: Proc. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, 2001, pp. 95–100.