Resolving Deadlocks for Pipelined Stream Applications on Network-on-Chips

Xiaohang Wang[1,2], Peng Liu[1]
[1]Department of Information Science and Electronic Engineering, Zhejiang University
Hangzhou, Zhejiang, P. R. China, 310027
e-mail: [1]baikeina@yahoo.com.cn,
[1]liupeng@isee.zju.edu.cn

Mei Yang[2], Yingtao Jiang[2]
[2]Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, NV 89154
e-mail: [2]{meiyang, yingtao}@egr.unlv.edu

***Abstract*—When a stream application that demands real-time processing over continuous data streams 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. In this paper, we focus on the request-request type message-dependent deadlocks, the most devastating deadlocks in stream applications, and show that this type of deadlocks can be avoided by a proper inclusion of virtual channels (VCs). We first prove a sufficient condition that determines the minimum number of VCs needed to completely avoid request-request type message-dependent deadlocks. We then show that the problem of finding the minimum number of such VCs for a given application is NP-complete, and subsequently, a mixed integral linear programming (MILP)-based algorithm, referred as Min_VC algorithm, is introduced to solve this problem. This Min_VC algorithm can literally be integrated with any existing application mapping algorithm to provide deadlock-free mapping results. The experiments results shown that for typical stream applications, such as multimedia applications, the number of VCs needed to avoid deadlocks is fairly modest, typically just 1 or 2 depending on applications. That is, with a modest price paid in terms of area and power, stream applications can run in an NoC-based system completely free of deadlock concerns, which is necessary to deliver the quality of service (QoS) guarantee required by these applications.**

Keywords: network-on-chip (NoC), message-dependent deadlock, virtual channel

I. 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] tailored for stream applications. The increasing importance of stream processing lies in the fact that many emerging applications involve real-time processing over continuous data streams, such as VoIP telephony, playback audio/video, IPTV, and sensor data analysis [2].

There are unfortunately two types of deadlocks that may occur to a pipelined stream application running on an NoC-based architecture: 1) the routing-dependent deadlocks [1] and 2) the message-dependent deadlocks [3, 4]. A message-dependent deadlock is created when some of these messages can never be consumed by the consumer task as the consumption of these messages is mutually dependent on each other's arrival. For example, message *X* may block message *Y* from arriving at the end processor/NI while *Y* is simultaneously required by the consumer task to consume *X*. This request-request type message-dependent deadlock can cause devastating effects on a stream application as it may put the whole system into a complete stall [3, 4].

To avoid or resolve message-dependent deadlocks in a network, generally two classes of methods, reactive methods and proactive methods, are used [3, 4, 7]. However, these existing methods are not suitable for the stream applications running on an NoC-based architecture, for reasons given below. For reactive methods, typified by a deadlock recovery mechanism referred as mDisha [7], they cannot deliver the QoS guarantee required by stream applications, since resolving deadlocks will cause unpredictably long delays [4].

There exist three possible proactive methods that may help avoid deadlocks. However, none of these methods will be suitable for stream applications which have stringent latency and throughput requirements.

(i). Use of network buffers with an extremely large size to avoid deadlocks. This approach is not suitable for NoC designs, as it involves unbearably high area and power costs. In addition, this method, as pointed out in [4], cannot help avoid the request-request type message-dependent deadlocks.

(ii). Creation of multiple virtual networks with one for each message type. This approach, unfortunately, cannot help avoid the request-request type message-dependent deadlocks [4].

(iii). Use of end-to-end flow control. The end-to-end flow control protocols, such as CTC protocols [3], may significantly increase the network latency and communication power due to the usage of additional reply messages for flow control [4].

Virtual channels (VCs) have been long employed in many NoC designs to help improve network routing performance. In this paper, we show that proper inclusion of VCs can also provide a practically feasible solution to completely avoid the request-request type message-dependent deadlocks in an NoC design. In particular, 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. Following this theory, we propose an MILP-based algorithm that can help quickly find this number, and this algorithm shall be integrated with existing application mapping algorithms, like the ones reported in [5], to get deadlock-free

978-1-4244-5539-3/10/$26.00 ©2010 IEEE

mapping results. Our experiments have revealed that for many popular stream applications, such as networking and multimedia applications, the number of VCs needed to avoid deadlocks is fairly modest, typically just 1 or 2 depending on applications. That is, with a modest price paid in terms of area and power, stream applications can run in an NoC-based system completely free of deadlock concerns.

II. Target Application and Architecture Model

A. Application model

The stream applications can be modeled as a synchronous data flow graph.

Definition 1[2] A synchronous data flow graph (SDFG) is a directed graph SDFG(A, EC), where each vertex $a_i \in A$ represents a task (to be consistent, an actor that was originally defined in [2] is also referred as a task in this text), and a directed edge $ec_i=(a_k, a_j)\in$EC represents the data communication from a_k to a_j. For task a_k, the following notations are defined,

- $EX(a_k)$ gives the worst execution time of task a_k.
- $IN(a_k)$ is the set of all a_k's predecessor tasks in SDFG.
- $OUT(a_k)$ is the set of all a_k's successor tasks in SDFG.

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 (packed in messages) from a_k to a_j after a_k finishes its m-th iteration. In this paper, we assume that all the tasks have already been bound to IP cores.

Definition 2 A stream *communication graph* (SCG) is a directed acyclic graph SCG(P, E), where a vertex $p_k\in P$ represents an IP core, and an edge $e_i=(p_k, p_j)\in E$ represents the communication between vertices p_k and p_j. $\omega(e_i)$ defines the amount of data sent from p_k to p_j in bits per second (bps).

Without loss of generality, we assume that each IP core is allocated with one task. If multiple tasks are allocated to a single IP core, we can treat these tasks as one macro task.

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 [2]. 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 [2]. That is, the number of output data buffers is the same as that of the task's successors.

B. Architectural model

Fig. 1 shows the architecture model used in this paper. The NoC system under consideration is composed of NxN tiles interconnected by a 2-D mesh network. Each tile, indexed by its coordinate (x, y), where $0\le x\le N$-1 and $0\le y\le N$-1, has one router and one processing node.

A router, as shown in Fig. 2, implements wormhole switching. Each message will be broken into a number of fixed size packets with each carrying the needed routing information. Each packet will be further decomposed into flits. Assume each physical channel is split into V VCs, realized by V buffers at each input port with each buffer holding several flits.

As shown in Fig. 2, each processing node is composed of four components: a processor, a bus, a local memory unit and a network interface (NI). The local memory unit holds the input/output data buffers that can be used by the task allocated to this IP core. The NI holds a number of receiving buffers each dedicated to holding several flits of the message from one of the task's predecessors. Hence, the number of receiving buffers configured at the NI is the same as the number of the task's predecessors.

Definition 3 An *Architecture Characterization Graph* (ACG) $\check{G}=(T, L)$ is a directed graph, where each vertex $t_i\in T$ represents a tile (Fig. 1), and each edge $l_i=(t_k, t_j) \in L$ represents the link between adjacent t_k and t_j. For link l_i,

- $bw(l_i)$ defines the bandwidth provided between tiles t_k and t_j.
- $c(l_i,V)$ defines the link cost of l_i, i.e., power consumption for transmitting one bit data from t_k and t_j with a total of V VCs.

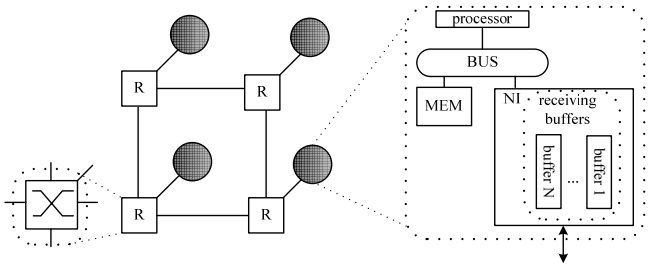

Figure 1 NoC architecture model. A rectangle box represents a router and a circle represents a processing node.

III. Message-dependent Deadlock in Stream Application and 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 SCG has three IP cores on which three tasks a, b, and c are allocated, as shown in Fig. 2(a), has already been mapped to the ACG with a 1x3 mesh using the mapping algorithm in [3]. The three tasks (with a and b as the producers and c as the consumer) form a pipeline. Assume the pushing protocol [4] 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. As

indicated in Section 2.1, 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 their corresponding input data buffers. Once *c* starts a new iteration, the two input data buffers must have been cleared.

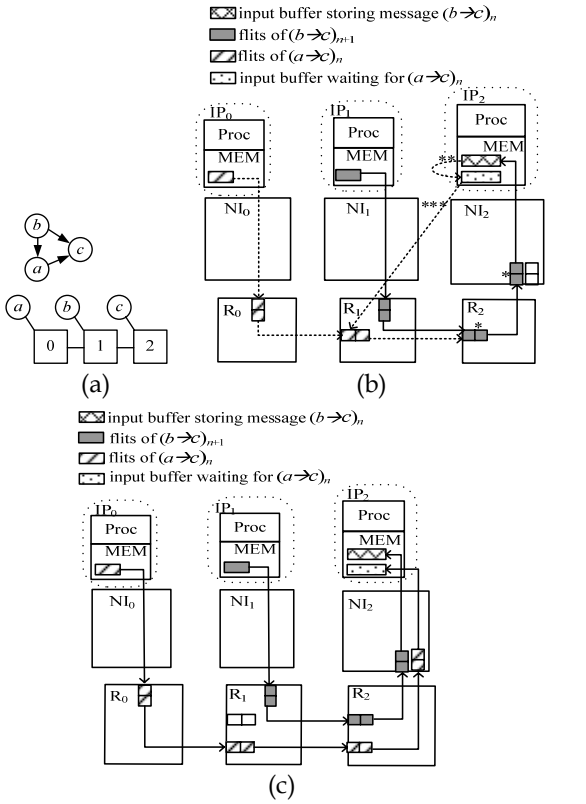

Figure 2 (a) An example of SCG with three processors mapped with three tasks. (b) Deadlock configuration due to request-request type message-dependency. (c) Increasing the number of virtual channels can help avoid request-request type message-dependent deadlocks.

A serious problem, however, may happen when, for example, *b* produces data at a higher rate than *a* and *c*. Fig. 2(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 $(b\rightarrow c)_{n+1}$ reserves link (1, 2), the west input buffer of R_2 and the receiving buffer of NI_2 (marked with * in Fig. 2(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.
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. 2(b)).
4) In Fig. 2(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 devastating deadlock can be formed. In another word, to avoid any deadlock like the one shown in Fig. 2, some network control mechanism should be in place to either proactively prevent any messages generated in different iterations from blocking others to reach their destinations, or act reactively to resolve the deadlocks once they are detected. As a matter of fact, all the existing deadlock avoidance methods [3, 4, 7], can actually be viewed as a variation of such control mechanism.

As a viable alternative to the existing deadlock avoidance methods, the request-request type message-dependent deadlocks can be completely avoided by adding a number of virtual channels. For the example shown in Fig. 2(a), one can see that the cycle that causes a deadlock in Fig. 2(b), is completely avoidable by employing two virtual channels at each router (Fig. 2(c)). Since in many NoC designs, virtual channels are used for helping improve network routing performance [2], the hardware cost of this approach for avoiding message-dependent deadlocks is well justifiable.

IV. Deadlock Avoidance Using Virtual Channels

A. *Sufficient condition for deadlock-avoidance with virtual channels*

Assume a stream application given as an SCG (Section 2) has already been mapped to an NoC architecture modeled by an ACG (Section 2). We also assume that (i) the routing algorithm used is deadlock-free, and (ii) messages are delivered in-order.

Lemma 1 For any task in a stream application allocated to an IP core in a given SCG, 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.

Lemma 2 For a stream application given as a SCG, 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 mapped IP core in the SCG.

Theorem 1 Consider an NoC architecture in which (i) *V* VCs are used at each router and (ii) the number of receiving buffers at the NI of each tile is set to be equal to the number of its predecessors. If each link is shared by no more than *V* communications, no request-request type message-dependent deadlock can ever be created.

The proofs of Lemmas 1-3 and Theorem 1 are omitted due to space limit.

B. *Minimum virtual channel algorithm*

MinVC problem: Given an ACG(*T*, *L*) that a SCG(*P*, *E*) is mapped onto, for each communication e_i in SCG, find a

routing path in ACG among all the possible minimal routing paths such that the number of virtual channels needed at each router is minimized, i.e.,

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

satisfying,

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

$$\forall\, l_m,\ B \geq \sum_{e_i=(p_k,p_j)\in E} \omega(e_i)\times g(l_m, h_{M(p_k),M(p_j)}) \quad (2)$$

where $M(p)$ is the tile that p is mapped to, and $g(l_i, h_{M(p_k),M(p_j)}) = \begin{cases} 1 \text{ if } l_i \in h_{M(p_k),M(p_j)} \\ 0 \text{ if } l_i \notin h_{M(p_k),M(p_j)} \end{cases}$.

The conditions given by (1) ensure that each link l in ACG is shared by at most V communication flows, while the condition given by (2) ensure that the total bandwidth requirement of all communication flows sharing each link l does not exceed the capacity of link l.

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

Theorem 2 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 communications in the SCG, while all the resource constraints are satisfied and no more than V VCs are needed at each router.

We will prove this theorem restricting the decision version of the MinVC problem to its instances with $\omega(e_i)$=1 for all ei, and V=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 [8] for a set of |E| communication flows {1, …, |E|}, each flow sending from a source node s_i to its destination node d_i with demand $\omega(e_i)$ for i∈{1, …, |E|}. The knapsack [9] problem can thus be viewed as a special case of the restricted MinVC problem, as shown in Fig. 3.

In Fig. 3, for each item in knapsack problem [9], there is a corresponding flows i (i.e., an edge e_i in the SCG) whose demand $\omega(e_i)$ is equal to the size of each item. The cost of each direct edge from node s to d_i is set to w_i/s_i where w_i and s_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 other edges have infinite capacity. Therefore, the minimum cost flows from all s_i to d_i, i∈{1, …, |E|}, upon satisfying all the demands, lead to an optimal solution to the knapsack problem and vice versa.

The knapsack problem is NP-complete [9], and so is the MinVC problem which can be transformed to the knapsack problem.

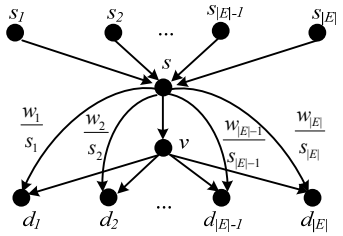

Figure 3 Formulation of the Knapsack problem [9] as the restricted MinVC problem.

The MinVC problem can be approximated by formulating it as a mixed integer linear programming problem given below. Here, MP_i is the set of all minimal paths between the two tiles mapped by the two IP cores of edge e_i, and $PATH_{i,j}$ is the j-th minimal path in MP_i. $f_{i,k} = \begin{cases} 1 \text{ if } e_i \text{ takes the } k-th \text{ minimum path} \\ 0 \text{ otherwise} \end{cases}$. Eqn. (2) sets a constraint that only one of the minimal paths will be allocated for each communication. Eqn. (3) sets that each link will not accommodate more than V communications. Eqn. (4) represents the bandwidth constraint of each link.

Our experiment has shown that when lp_solve [10] is used to solve Min_VC, the running time is less than 0.1sec for a 6×6 mesh-based NoC (obtained from a PC with one Intel Core2 P8600 2.4GHz processor and 2GB RAM).

Min_VC(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 minimal paths for all e_i
Procedure body:
{
call lp_solve to solve the following equations:
Objective: min V (1)
Constraints:

$$\sum_{k\in|MP_i|} f_{i,k} = 1, \forall e_i \quad (MP_i \text{ is the set of all minimal path set for } e_i); \quad (2)$$

$$\sum_{e_i} \sum_{l\in PATH_{i,j}, k\in|MP_i|} f_{i,k} \leq V, \forall link\ l; \quad (3)$$

$$\sum_{e_i} \sum_{l\in PATH_{i,j}} \omega(e_i)\times f_{i,k} \leq B, \forall link\ l; \quad (4)$$

}

V. EXPERIMENTS

In our experiments, the mapping process is based on an existing mapping algorithm [5].

A. Simulation results

Fig. 4 shows a synthetic application. The execution time of each stage is modeled by a random process with the mean and variance are chosen randomly. For the mapping result in Fig. 4(b), it has been found that 2 VCs are needed to completely avoid any deadlocks.

The CMMS system in Fig. 5 is composed of both a video/audio encoder and a video/audio decoder. The video and audio encoders are synchronized, and so are the video and audio decoders. The tasks in the video codec are

synthesized as follows. The mean, variance and the Hurst parameter of the execution time of each stage are from [11] We assume that the target platform runs QCIF format video which has 99 macro blocks in one frame. The video processing part is pipelined at the macro block level and is synchronized with the audio processing at the end of each frame. For this application, 1 VC is found sufficient to avoid deadlocks.

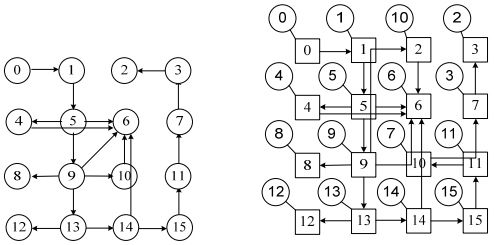

Figure 4 (a) The SCG of synthetic application. (b) The mapping result.

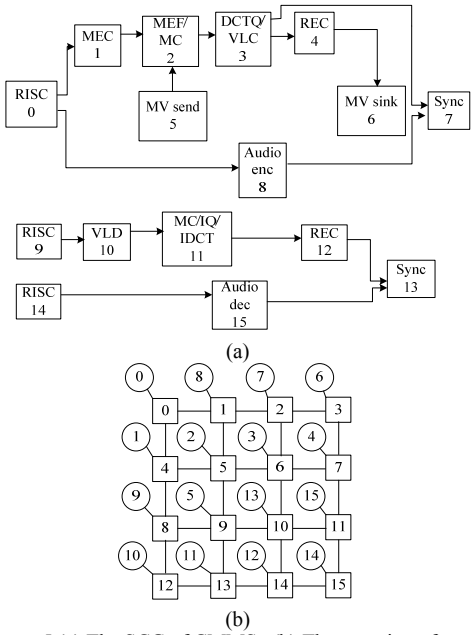

Figure 5 (a) The SCG of CMMS. (b) The mapping of result.

Table I shows the power and area results for each application with the number of VCs needed to avoid deadlocks. The results shown are normalized with respect to a baseline design where no VC is used. Of the three applications, the maximum number of VCs needed to avoid deadlocks is 2. In the worst scenario, the power and area penalties are 16% and 17%, respectively, which is tolerable for gaining the benefit of avoiding message-dependent deadlocks that otherwise will have devastating effects on stream applications. With these numbers of VCs, the applications have no routing dependent and message dependent deadlocks observed.

TABLE I. THE AREA AND POWER OVERHEAD TO AVOID DEADLOCKS FOR THE THREE APPLICATIONS.

Application	VC required	Power	Area
Synthetic	2	1.16x	1.17x
CMMS	1	1x	1x

VI. CONCLUSION

In this paper, we presented a practical method to avoid the request-request type message-dependent deadlock problem that can seriously impact the performance of a pipelined stream application running on an NoC architecture. We showed that the request-request type message-dependent deadlocks can be avoided by adding the right number of virtual channels (VCs) and formally proved a sufficient condition which determines the minimum number of VCs actually needed to obtain such a deadlock-free NoC designs. We further showed the problem of finding the minimum number VCs is NP-complete and thus proposed an ILP-based solution which can be and should be integrated into any mapping algorithms whenever a deadlock-free design is desired. Experiments based on three stream applications confirmed that deadlocks have been avoided with a modest increase of power and area.

REFERENCES

[1] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "Application specific routing algorithms for Networks on Chip," IEEE Trans Parallel and Distributed Systems, vol. 20, no. 3, pp. 316-330, 2009.

[2] N. K. Kavaldjiev, "A run-time reconfigurable Network-on-Chip for streaming DSP applications," Phd thesis, University of Twente, 2007.

[3] N. Concer, L. Bononi, M. Soulié, R. Locatelli, and 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, 2009, pp. 193-202.

[4] A. Hansson, K. Goossens, and A. Radulescu, "Avoiding message-dependent deadlock in network-based Systems-on-Chip," VLSI Design, vol. 2007, no., pp. 1-10, 2007.

[5] G. Ascia, V. Catania, and M. Palesi, "Mapping cores on network-on-chip," Int'l J Computational Intelligence Research, vol. 1, no. 2, pp. 109-126, 2005.

[6] X. Wang, M. Yang, Y. Jiang, and P. Liu, "A power-aware mapping approach to map ip cores onto nocs under bandwidth and latency constraints," to appear in ACM Trans. Architecture and Code Optimization.

[7] Y. H. Song and T. M. Pinkston, "A progressive approach to handling message-dependent deadlock in parallel computer systems," IEEE Trans Parallel and Distributed Systems, vol. 14, no. 3, pp. 259-275, 2003.

[8] M. Skutella, "Approximating the single source unsplittable min-cost flow problem," Mathematical Programming, vol. 91, no. 3, pp. 493-514, 2002.

[9] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness: Freeman San Francisco, 1979.

[10] lp solve 5.5. [Online]. Available: lpsolve.sourceforge.net/5.5/.

[11] C. Lampert, M. Militzer, and P. Ross. XviD MPEG4 core library.