Adaptive Routing in Network-on-Chips Using a Dynamic-Programming Network

Terrence Mak, *Member, IEEE*, Peter Y. K. Cheung, *Senior Member, IEEE*, Kai-Pui Lam, and Wayne Luk, *Member, IEEE*

***Abstract*—Dynamic routing is desirable because of its substantial improvement in communication bandwidth and intelligent adaptation to faulty links and congested traffic. However, implementation of adaptive routing in a network-on-chip system is not trivial and is further complicated by the requirements of deadlock-free and real-time optimal decision making. In this paper, we present a deadlock-free routing architecture which employs a dynamic programming (DP) network to provide on-the-fly optimal path planning and network monitoring for packet switching. Also, a new routing strategy called k-step look ahead is introduced. This new strategy can substantially reduce the size of routing table and maintain a high quality of adaptation which leads to a scalable dynamic-routing solution with minimal hardware overhead. Our results, based on a cycle-accurate simulator, demonstrate the effectiveness of the DP network, which outperforms both the deterministic and adaptive-routing algorithms in average delay on various traffic scenarios by 22.3%. Moreover, the hardware overhead for DP network is insignificant, based on the results obtained from the hardware implementations.**

***Index Terms*—Adaptive routing, Bellman equation, dynamic programming (DP), DP network, network-on-chip (NoC).**

I. Introduction

INTERCONNECT performance is rapidly deteriorating with the continuous scaling in technology processes. As predicted by the International Technology Roadmap for Semiconductors (ITRS) in Fig. 1, there is a significant performance gap between interconnection RC delay and the gate delay, and this gap will be increasing exponentially (9:1 with the 65-nm technology, according to ITRS 2005 report [1]). The gap will continue to grow even with the help of new interconnect materials and aggressive interconnect optimization [2], [3]. Furthermore, because of the tightly packed wires, capacitances that are attributed to interconnect parasitic also increase drastically. As a result, multilevel interconnect networks have become the primary limit on the productivity, performance, energy dissipation, and signal integrity of gigascale integration [4].

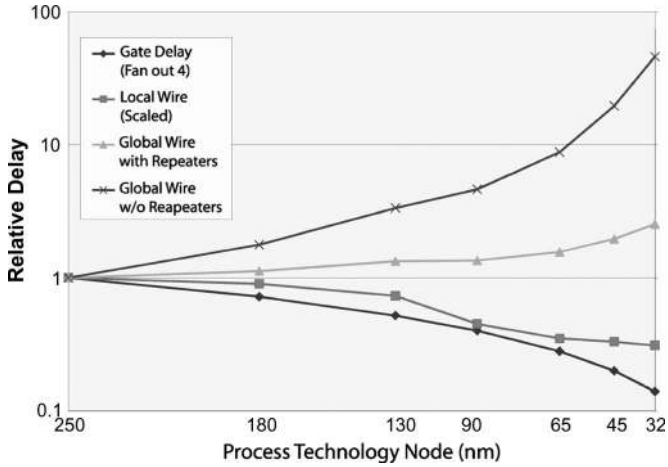


Fig. 1. Projected relative delay for local and global wires and for logic gates in technologies of the near future. [1].

Recently, network-on-chip (NoC) has been proposed as a promising solution to the increasingly complicated on-chip communication challenges [5]–[7]. Such architectures consist of a network of regular tiles where each tile can be an implementation of general-purpose processors, DSP blocks, memory blocks, and embedded reconfiguration modules, *etc*. Communications among these tile-based modules are following a packet-switch or circuit-switch scheme where messages are transmitted among the processing elements. The NoC architecture would be an ideal solution to provide effective integration for multiple modular blocks [8] and can potentially mitigate the gigascale integration challenge [9], [10].

In such an NoC environment, the routing of flits (or packets) becomes a critical issue, which determines the interprocessor communication performance. Routing provides a protocol for moving data through the NoC infrastructure and also determines the path of data transport. The selection of communication pathway would greatly affect the latency of packets transmitted from the source to the destination and, therefore, can have significant impact on the overall traffic flow in the network. An intelligent routing mechanism is required to utilize the communication bandwidth and minimize transportation latency.

Dynamic routing (or adaptive routing) has been widely used in computer and data network design. Utilizing the online communication patterns and real-time information, dynamic

Manuscript received December 31, 2009; revised April 28, 2010 and June 7, 2010; accepted September 6, 2010. Date of publication September 30, 2010; date of current version July 13, 2011.

T. Mak is with the School of Electrical, Electronic and Computer Engineering, Newcastle University, NE1 7RU Newcastle upon Tyne, U.K. (e-mail: terrence.mak@ncl.ac.uk).

P. Y. K. Cheung is with the Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ London, U.K. (e-mail: p.cheung@ic.ac.uk).

K.-P. Lam is with the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: kplam@se.cuhk.edu.hk).

W. Luk is with the Department of Computing, Imperial College London, SW7 2AZ London, U.K. (e-mail: wl@doc.ic.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2010.2081953

0278-0046/$26.00 © 2010 IEEE

routing can effectively avoid hot spots or faulty components and can reduce the possibility of packets being continuously blocked. Several partially adaptive-routing algorithms within the context of NoC were proposed, and the evaluations of their performances were reported. For example, implementation of wormhole-adaptive odd–even routing was described in [8], [11], and [12]. In [13], a minimal routing mechanism with partially adaptive protocols was proposed. However, implementation of adaptive routing in an NoC system is not trivial and is further complicated by the requirements of deadlock-free and real-time optimal decision making. Also, the previously proposed adaptive approaches only exploit local traffic which lead to a moderate improvement in packet latency and traffic load balancing. Optimal path planning and routing adaptations, which were considered as hardware expensive as their counterparts in computer networks, are rarely studied.

In this paper, we introduce a novel methodology to enable dynamic routing in an NoC. A massive parallel and high-throughput network architecture, namely, dynamic programming (DP) network, that provides real-time computation for shortest path problems is presented. This network couples with the NoC to enable optimal traffic control based on the online network status and, thus, provides optimal path planning and dynamic routing with novel routing mechanics. The DP network presents a simple, reliable, and efficient methodology to enable adaptive routing in NoCs. The major contributions of this paper are as follows.

1) A novel DP network for high speed and parallel shortest path computation is presented. The characteristics of the DP network, such as discrete and continuous-time formulations, network dynamics, and convergence, are discussed, and two numerical examples are presented to exemplify the high-gain and versatility properties. (Section III)
2) Integration of DP network and NoC architecture as a dual network is introduced. Routing mechanics and routing-table updating strategies, such as fully optimal and sub-optimal k-step look ahead (KLSA), are presented. The dual network enables a tradeoff between the routing optimality and memory consumption. Network scalability and deadlock issues are also discussed. (Section IV)
3) Performances and merits of the DP network are investigated thoroughly through experimental studies based on SystemC cycle accurate simulator. The new method is compared with other popular routing schemes, such as XY and odd–even, in different traffic benchmarks and large-scale NoC architectures. (Section V). The proposed DP-network architecture is realized using Xilinx field-programmable gate array (FPGA) device, and hardware overhead and performances are evaluated. (Section VI)

II. Preliminaries

A. Routing in NoC

NoC is an architecture inspired by data-communication networks, such as Internet, communication [14], and wireless networks [15], with interprocessor communication supported by a packet-switched and circuit-switched networks [5], [6]. The basic idea of NoC is to communicate across the chip in a way similar to that of messages transmitted over the Internet as the methods and architectures from the computer network could be borrowed and adopted to the on-chip communication and can potentially resolve the interconnect scaling challenges. It has been reported that the NoC architecture can effectively overcome the long-wire disadvantages from bus architectures as on-chip switches are connected in a regular topology with point-to-point basis, and long wires can be eliminated from the architecture [10]. Also, the architecture is decoupled into different layers, such as transaction and physical layers. Thus, the layered architecture enables independent optimization and design for each independent abstract layer.

Given an NoC architecture, routing becomes the most important design strategy to consider, which determines the overall system performance. Routing strategies can be categorized into deterministic and adaptive schemes. In a deterministic routing strategy, source and destination determine the traversal path. Popular deterministic routing schemes for NoC are source routing and XY routing, which are also referred to as 2-D dimension-order routing [16]. In source routing, the source core specifies the route to the destination. In XY routing, the packet follows the rows first then moves along the columns toward the destination, or vice versa. XY routing can be implemented using algorithmic routing logic but is limited to regular network topologies.

In an adaptive-routing strategy, the traversal path is decided on a per-hop basis. Adaptive schemes involve dynamic arbitration and next-hop selection mechanisms, i.e., based on local link congestions. There are several adaptive-routing algorithms that have been proposed within the context of NoC [17]. For example, a methodology that focuses on deadlock-free adaptive routing has been proposed in [18], which provides a framework to design routing tables that can outperform the turn-model-based deadlock-free routing algorithm. Other schemes, such as the adaptive odd–even [8], [12] and adaptive selection node-on-path (NoP) [13], also provide routing adaptability but only exploit local traffic or conditions of neighbors. There is a great potential to improve communication efficiency by considering the global traffic at runtime using adaptive routing, such as global traffic monitoring [19] and adaptive global routing [20]. However, these approaches employ either a rule-based approach or heuristics for traffic adaptation. Utilizing an on-demand shortest path computation could improve the routing optimality and adaptability effectively.

Minimal-cost (or shortest path) computation is fundamental among different dynamic-routing strategies. The basic idea is that the routing algorithm always chooses the least congested path toward the destination through optimal path planning. The least congested route can be found based on the shortest path computation where the path cost is obtained at runtime. Since the network status, such as traffic intensity and conditions, is changing at runtime, the dynamic-routing algorithm should be able to discover the congestions and perform shortest path computation at the same time. A novel DP-network architecture that provides real-time shortest path computation and optimal path planning is proposed in this paper. The background of shortest

TABLE I
NOTATIONS USED IN THIS PAPER

Variables for the DP problem formulation	
$\mathcal{V}$	A set of nodes in network $\mathcal{G}$
$\mathcal{A}$	A set of edges in network $\mathcal{G}$
$C_{u,v}$	Cost of the edge $n_u \rightarrow n_v$
N	Number of nodes in the network
n_i	The i-th node in the network
$d(u,v)$	the expected path cost from n_u to n_v
$V(u,v)$	The cost-to-go function or DP-value for node n_u and destination n_v
$V^*(u,v)$	The optimal cost-to-go for node n_u
$W^{(k)}(u,v)$	The expected cost to go for node n_u with k-step look ahead
$\mathcal{P}^k_{u,v}$	The set of paths from n_u to n_v, all of which have k edges
$\mu(u,v)$	The decision variable (routing direction) at node n_u
$\mathcal{N}(i)$	A set of neighbour nodes that can be directly accessed from node n_i
Variables for the DP-network system	
$S_k(u,v)$	The k-th site function of a DP unit where n_u is the current node and n_v is the destination.
$g(u,v)$	The output of a DP unit where where n_u is the current node and n_v is the destination.

path computation and the parallel computation architecture are described in the following.

B. Shortest Path Computation

DP is a powerful mathematical technique for making a sequence of interrelated decisions. Bellman formalized the term DP and used it to describe the process of solving problems where one needs to find the best decision one after another [21], [22]. It provides a systematic procedure for determining the optimal combination of decisions which takes much less time than naïve methods [23]. In contrast to other optimization techniques, such as linear programming (LP), DP does not provide a standard mathematical formulation of the algorithm. Rather, DP is a general type of approach to problem solving, and it restates an optimization problem in recursive form, which is known as Bellman equation [21], [22]. The Bellman equation for optimal-value function $V(\cdot)$ is unique and can be defined as the solution to the recursive equation [22], [24].

The shortest path problem can be described as follows: Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ with $n = |\mathcal{V}|$ nodes, $m = |\mathcal{A}|$ edges, and a cost associated with each edge $u \rightarrow v \in \mathcal{A}$, which is denoted as $C_{u,v}$, the edge cost can be defined subject to different applications, and the cost is defined as the number of flits or packets in a buffer in this paper. The total cost of a path $p = \langle n_0, n_1, \ldots, n_k \rangle$ is the sum of the costs of its constituent edges: $\text{Cost}(p) = \sum_{i=1}^{k} C_{i-1,i}$. The shortest path of $\mathcal{G}$ from n_i to n_j is then defined as any path p with cost that is $\min \sum_{i=1}^{k} C_{i-1,i}$ for all constituent edges n_i. The notations are summarized in Table I.

The shortest path problem as a linear optimization problem can be formally stated. Suppose that node n_w is the destination node and it aims to compute the shortest path cost $d(v,w)\ \forall v \in \mathcal{V}$. To express this as a linear program, the constraint becomes $d(v,w) \leq d(u,w) + C_{u,v}$ to denote that the cost of the shortest path from any node n_v to destination n_w is less than or equal to the shortest path from node n_u plus the cost of a direct path from node n_u to node n_v. The destination node n_w vertex initially receive a value $d(w,w) = 0$. Thus, the following LP formulation can be obtained:

$$\begin{aligned} \text{minimize} \quad & \sum_{\forall v \in \mathcal{V}} d(v,w) \\ \text{subject to} \quad & d(v,w) \leq d(u,w) + C_{v,u}\ \forall v,u \in \mathcal{V} \\ & d(w,w) = 0. \end{aligned}$$

The previous formulation yields the shortest path from any nodes in $\mathcal{V}$ to destination n_w, which is known as multiple-source–single-destination shortest path problem. Solution of an LP problem can be resolved readily using any standard LP solver [25].

Alternatively, the shortest path problem can be stated in the form of Bellman equation, which defines a recursive procedure in step k and can lead to a simple parallel architecture to speed up the computation. To find the cost of the shortest path from n_v to n_w, it requires the notion of DP value or, namely, cost-to-go function, which is the expected cost from n_v to n_w. This expected cost is being updated recursively based on the previous estimates until it reaches its optimality criteria. This algorithm is known as *DP*. We denote the DP value for n_v to n_w at the kth iteration as $V^{(k)}(v,w)$, and $V^*(v,w)$ is the optimal DP value, which is equal to the resolved variable $d(v,w)$ from the aforementioned LP formulation. The Bellman equation becomes

$$V^{(k)}(v,w) = \min_{\forall u \in \mathcal{V}} \left\{ V^{(k-1)}(u,w) + C_{v,u} \right\} \tag{1}$$

where $V(w,w) = 0$. If the recursion is expanded from n_0 to n_k, the DP value can be expressed as the total cost of the path from node n_0 to node n_k

$$V^*_k(n_0,n_k) = \min_{\{n_0,n_1,\ldots,n_k\} \in P^k_{n_0,n_k}} \left\{ \sum_{i=1}^{k} C_{i-1,i} \right\} \tag{2}$$

where destination node $n_w = n_k$ and $P^k_{i,j}$ are the set of paths from n_i to n_j, all of which have k edges. In addition, the optimal decisions at each node n_i that lead to the shortest path can be readily obtained from the argument of the minimum operator at the Bellman equation as follows:

$$n_v = \arg\min_{\forall u \in \mathcal{V}} \{ V^*(u,w) + C_{v,u} \} \tag{3}$$

where the optimal decision becomes $\mu(v,w)$. Both the LP and DP can yield the optimal solution for shortest path problems. However, the DP approach presents an opportunity for solving the problem using a parallel architecture and can greatly improve the computational speed.

III. SHORTEST PATH COMPUTATION USING DP NETWORK

A. General Architecture

Mapping Bellman recursive DP to a parallel computation platform can be realized with the introduction of a DP-network architecture. The network has a parallel architecture and can be

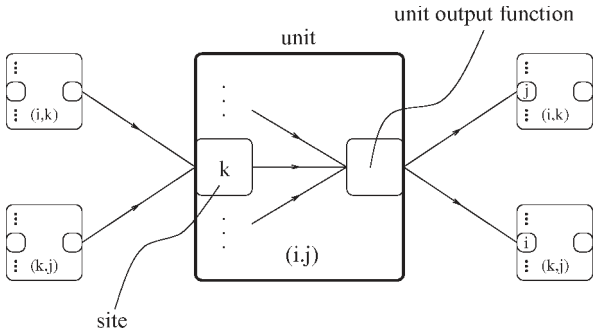


Fig. 2. Unit interconnection in a general DP-network architecture, where $1 \leq i, j, k \leq n$; $k \neq i, j$. Unit (i, j) outputs the cost-to-go value $V(i, j)$, which will be the input of other units according to the problem network structure. At each unit, there are $|\mathcal{N}(i)|$ sites, which correspond to the total number of neighboring nodes of i, to carry out the inference operations as defined in the site function.

used to derive DP solution through the simultaneous propagation of successive inferences. Originally, it provides an efficient platform for checking data inconsistency due to results from different inference paths [26]. In [26], with close resemblance to the deterministic-type DP formulation on closed semiring, Lam and Tong introduced a continuous-time ordinary differential equation (ODE) network to solve a set of graph optimization problems with an asynchronous and continuous-time computational framework. This new class of inference network is inherently stable in all cases, and it has been shown to be robust and with arbitrarily fast convergence rate [26]. A similar parallel computational network for DP has also been proposed in [24]. The network was proven to converge to optimal solution even under an asynchronous network.

A DP network is formed by the interconnection of self-contained computational units. Fig. 2 shows the structure of a unit and the connections in a general inference network. Each unit is to represent a binary relation $\langle i, j\rangle$ between two objects i and j and is denoted by $U(i, j)$. At each unit, there are $|\mathcal{N}(i)|$ sites, which correspond to the total number of neighboring nodes of i, to carry out the inference operations, as defined in the site function. The value of the corresponding relationship between i and j is then determined by resolving the conflict among all of the site outputs. In essence, if $S_k(i, j)$ represents the site output at the kth site and $g(i, j)$ stands for the unit output of unit (i, j), then

$$S_k(i, j) = g(i, k) \otimes g(k, j) \tag{4}$$
$$g(i, j) = \oplus_{\forall k \in \mathcal{N}(i)} S_k(i, j) \tag{5}$$

where $\otimes$ is the inference operator for the site function (which is usually the same at all of the sites) and is the conflict-resolution operator for the unit function. Also, the computational unit $\oplus$ denotes the unit which resolves the binary relation (i, j).

The shortest path problem can be mapped to the DP network. For the original problem graph, each node refers to a processor unit. However, in the DP network, each computational unit $U(i, j)$ represents the binary relation, i.e., the expected distance between node i and j. When the network has converged, the solution of the problem would be found at the output of each computational unit. In general, if there are m nodes in the original graph, then the DP network (based on the Bellman equation) will have $m - 1$ functional units with $U(i, j)$, where $i = 1, 2, \ldots, j - 1, j + 1, \ldots, m$.. By supposing that the interconnection network has a fixed topology, the multiple-source–multiple-destination solutions can be obtained by applying the DP network m times for computing the shortest paths for m different destinations.

Let $g(i, k) = C_{i,k}$ and $g(k, j) = V(k, j)$. The architecture of the DP network can then be defined as follows. A DP network for the shortest path problem can be stated in terms of network structure as $\otimes$ is substituted by "+" and $\oplus$ is substituted by "min" as

$$S_k(i, j) = g(i, k) + g(k, j) \tag{6}$$
$$g(i, j) = \min_{\forall k \in \mathcal{N}(i)} S_k(i, j). \tag{7}$$

The computational units are interconnected and resemble the shortest path problem structure. Each unit represents a node, and an interconnection represents an edge. With the realization, the network converges, and the optimal solution can be readily implemented using a distributed network. Note that when the network resolves, the optimal cost-to-go function can be obtained as $V^*(i, j) = g(i, j)$. Also, this network architecture encompasses the advantage of simplicity and parallelization, which presents a great opportunity to be applied for on-chip routing and optimization.

B. Discrete and Continuous-Time Formulations

The recursive formulation of the Bellman equation only specifies the mechanism to update value $V(u, v)$, as can be found from the classical Value Iteration algorithm [23], [27]. Therefore, the priority and order of the updating process are not relevant, and the value $V(u, v)$ can be computed asynchronously. This allows an opportunity to design distributed computation system to realize the DP network with distributed computational units without synchronous control. Furthermore, the asynchronous property can be further exploited to consider a continuous-time framework of the DP network, as opposed to the discrete-time DP network. The continuous-time formulation provides an analytical framework to study the network properties, such as network convergence. In the following, both the discrete and continuous-time formulations are discussed.

1) Continuous-Time Formulation: Consider a DP network that is constructed based on the original shortest path problem. Computational unit i is interconnected with adjacent node j, $\forall j \in \mathcal{N}(i)$, where $C_{i,j}$ is finite. Assume that the min and $+$ operators require an infinitesimal time δt; the output of the operator at time $t + \delta t$ can be expressed as [26]

$$g_{t+\delta t}(i, j) = \min_{\forall k \in \mathcal{N}(i)} \{g(k, j) + C_{i,k}\} \tag{8}$$

Assuming that the transition costs between the current node and the nonadjacent nodes are infinite, minimizing only over the set of neighboring nodes in (8) is equivalent to minimizing over all nodes. Also, minimizing only over the adjacent nodes leads to a hardware realization with smaller cost. Suppose that the cost function $C_{i,j}$ is a constant and the min and $+$ operators require an infinitesimal time, each computational unit $U(i, j)$ could

then behave dynamically as a first-order system. The whole network can be described by a set of differential equations

$$\frac{dg(i,j)}{dt} = -\lambda_i g(i,j) + \lambda_i \min_{\forall k \in \mathcal{N}(i)} \{g(k,j) + C_{i,k}\}, \forall i \quad (9)$$

where λ_i is the system pole for unit $U(i,j)$, which controls the rate of how $g(i,j)$ may change. If $\lambda_i = 0$, then $|dg(i,j)/dt| = 0$, and $g(i,j)$ becomes a constant, and the unit is said to be fully constrained and has a fixed memory. Whereas, for a memoryless unit with $\lambda_i = \infty$, it has an infinite power to change because $|dg(i,j)/dt|$ can be made arbitrarily large. Also, the units are interconnected based on $\mathcal{N}(i)$, which defines the set of adjacent nodes of unit $U(i,j)$. Therefore, $g(k,j)$ is the output of unit $U(k,j)$, which is an adjacent unit of $U(i,j)$ in $\mathcal{N}(i)$.

2) Discrete-Time Formulation: The equivalent discrete formulation can be obtained based on (9). Let $\delta t = 1$. The system of differential equations (9) then becomes

$$g_{t+1}(i,j) = \lambda_i \min_{\forall k \in \mathcal{N}(i)} \{g_t(k,j) + C_{i,k}\} \forall i \quad (10)$$

where λ_i defines the converging time constant, which controls the convergence speed of the system, as will be shown in the next section.

C. Convergence of the Network

There are two important considerations in using a DP network. First, will the network always converge to the desired solution? Second, what are the parameters or conditions that affect the convergence rate of the network? The answer to the first question is a "yes" because it follows directly from the principle of the Bellman optimality equation which states that the constituent optimal expected value of all states are optimal. The local minimization based on the Bellman equation performed at each distinct unit, in fact, is driving the network to a global optimal state, which is the desired solution. To measure the "distance" of the network from this global minimum and in line with Hopfield's energy modeling in [28], the computational energy $E(t)$ can be defined as the root-mean-square (rms) error if the system deviates from the optimal solution. From (9), the energy function for the continues-time ODE can be stated as

$$E(t) = \sum_{\forall i} \left(-\lambda_i g(i,j) + \lambda_i \min_{\forall k \in \mathcal{N}(i)} \{g(k,j) + C_{i,k}\} \right)^2 \quad (11)$$

where $E(t) = 0$ when the network has converged. To determine the convergence rate of the network, an explicit expression for $dE(t)/dt$ has to be evaluated. By differentiating the energy function in (11), the following expression is obtained:

$$\begin{aligned} \frac{dE(t)}{dt} &= \frac{dE(t)}{dg(i,j)} \cdot \frac{dg(i,j)}{dt} \\ &= \sum_{\forall i} \left[\frac{d}{dg(i,j)} \left(-\lambda_i g(i,j) + \lambda_i \min_{\forall k \in \mathcal{N}(i)} \{g(k,j) + C_{i,k}\} \right)^2 \cdot \frac{dg(i,j)}{dt} \right]. \end{aligned} \quad (12)$$

By evaluating the first term in (12), the following expression is obtained:

$$\begin{aligned} \frac{dE(t)}{dt} &= \sum_{\forall i} \left[-2\lambda_i \left(-\lambda_i g(i,j) + \lambda_i \min_{\forall k \in \mathcal{N}(i)} \{g(k,j) + C_{i,k}\} \right) \cdot \frac{dg(i,j)}{dt} \right] \quad (13) \\ &= \sum_{\forall i} \left[-2\lambda_i \left(\frac{dg(i,j)}{dt} \right)^2 \right]. \quad (14) \end{aligned}$$

Note that in order to establish the aforesaid expression, it is assumed that all outputs of units $g(i,j)$ do not provide a feedback to the unit itself. Thus, in the set of neighboring nodes, $\forall k \in \mathcal{N}(i)$, $k \neq i$. Hence, all the factors that make up the sum of the right-hand side of (14) are nonnegative. In other words, the energy function $E(t)$ defined in (11) is a monotonically decreasing function of time as

$$\frac{dE(t)}{dt} \leq 0. \quad (15)$$

From the definition of (11), note that the function $E(t)$ is bounded. The time evolution of the continuous DP-network model described by the system of first-order differential equations in (9) represents a trajectory in the station space, which seeks out the minima of the energy function $E(t)$ and comes to a stop at such fixed point. From (14), note that the derivative $dE(t)/dt$ vanishes only at the point that satisfies the Bellman optimal criterion

$$\frac{dg(i,j)}{dt} = 0 \ \forall i. \quad (16)$$

D. Numerical Examples

Example 1: Computing the Expected Costs in a Ten-Node Array: A ten-state random-walk problem can be solved by a ten-unit continuous-time DP network. The ten states are indexed by S_i, $i = 1, 2, \ldots, 10$. The outputs of the ten units of the network, signifying the expected costs to the destination, are described by a vector $V(S_i, S_{10})$, $i = 1, 2, \ldots, 10$, which has a semantic meaning of the expected reward of $V(S_i, S_{10})$, $i = 1, 2, \ldots, 10$. Also, the transition cost is defined as $C_{i,i+1} = 1$ and $C_{i+1,i} = 1$ for all i, $j = 1, 2, \ldots, 9$, and $C_{i,j} = \infty$ for all $j \neq i + 1$ and $j \neq i - 1$. The continuous-time DP network can be modeled by a set of differential equations on the ten nodes S_i. The expected rewards $V(S_i, S_{10})$ evolve as first-order lag controlled by λ, which is the reciprocal of the network-convergence-time constant. In particular, it relates to the computational delay of each computational unit in a network implementation, and the latency of information propagates throughout the network. The discount factor γ is a problem-related parameter, which defines the discount factor for multistage cost. The value is independent of the network

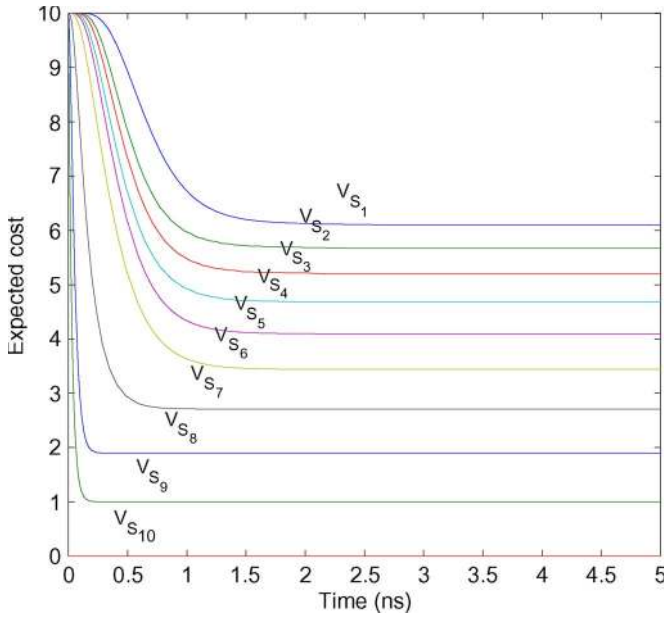


Fig. 3. Convergence of a DP network for the ten-node array random-walk problem where the time constant $1/\lambda = 1$ ns. Each curve corresponds to the output of each unit and represents the cost-to-go value from that node to the destination node S_{10}.

implementation and is subject to the requirements of the objective function

$$\frac{dV(S_1, S_{10})}{dt} = -\lambda V(S_1, S_{10}) + \lambda\gamma V(S_2, S_{10}) \tag{17}$$

$$\frac{dV(S_i, S_{10})}{dt} = -\lambda V(S_i, S_{10}) + \lambda \min\{C_{i,i-1} + \gamma V(S_{i-1}, S_{10}), C_{i,i+1} + \gamma V_{S(i+1)}, S_{10})\} \; \forall i = 2, 3, \ldots, 9 \tag{18}$$

$$V(S_{10}, S_{10}) = 0. \tag{19}$$

Equation (17) describes the boundary node S_1 which has a single "right" action. For nodes S_i, $i = 2, 3, \ldots, 9$, they have both left and right actions and can be readily shown to follow the equations as typified in (18). A destination-node value $V(S_{10}, S_{10})$ is defined to be zero, as in (19).

Given arbitrary positive initial values of $V(S_i, S_{10})$ $\forall i$, the converged values of the respective differential equations [(17)–(19)] can be verified to be identical with the optimal values governed by the Bellman equations. Fig. 3 shows the convergence results obtained by using Matlab ODE solver[1] for the differential equations. The converged values are found to be [6.10, 5.67, 5.20, 4.68, 4.10, 3.44, 2.71, 1.90, 1.00, 0]. The results are verified correctly against the results computed using the well-known Bellman–Ford algorithm for shortest path problems. Also, note that node S_9 is the quickest to converge, whereas S_1 is the slowest. This is because there is a dependence on the expected cost, and it takes the longest time for information to propagate to S_1 from S_{10}.

[1]The differential equation solver is based on ode45, which is provided in the Matlab. The ode45 is based on an explicit Runge–Kutta formula, the Dormand–Prince pair.

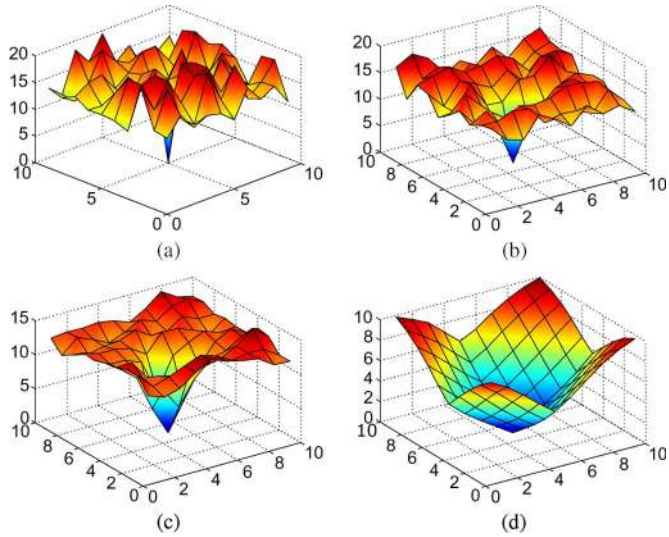


Fig. 4. Convergence of the cost-to-go values of all the nodes from a 10×10 mesh network. (a) $t = 1$ ns. (b) $t = 5$ ns. (c) $t = 10$ ns. (d) $t = 20$ ns.

Example 2: Computing the Expected Costs in a 10 × 10 Mesh: Consider a 100-node network with 10 by 10 mesh interconnection. Each node only connects to a maximum of four adjacent nodes, while each node at the edges connects to three, and each node at the corners connect to two. The nodes are oriented as a perfect square. All transitions would result in a cost of one, and the destination node at the center would have an expected cost of zero.

Similar to Example 1, the continuous-time DP network can be modeled by 100 differential equations on the 100 nodes S_{ij} $\forall i, j = 1, 2, \ldots, 10$. The expected cost $V(S_{ij}, S_{ij})$ $\forall i, j = 1, 2, \ldots, 10$ evolves as first-order lag controlled by λ.

Let $1/\lambda = 2$ ns and the destination node to be $S_{5,5}$. The values of the expected cost are shown in Fig. 4. At time $t = 1$ ns, the expected costs are randomly initialized, and $V(S_{5,5}, S_{5,5}) = 0$ as $S_{5,5}$ is the destination node. The network begins to converge to the optimal solution at time $t = 20$ ns, and the intermediate results are also shown in the figure. The convergence of the DP network in the 2-D mesh depends on λ, which, in this example is equal to 0.5. The network settles to the desired solution at $t = 20$ ns. By increasing λ, the time needed for the network to settle decreases. Also, even if λ is a large value (e.g., $\lambda = 0.9$), the network still converges to the optimal solution.

Fig. 5 shows the convergence of the network with different λ values. The results are rms errors between the V_S output from the network and the values obtained using the Bellman–Ford algorithm, averaged over the 10×10 mesh example. Clearly, λ is the reciprocal of the network time constant, which governs the time required to obtain the optimal solutions.

E. Summary

In this section, the characteristics of the DP network have been discussed. The DP network can be formulated in discrete and continuous-time forms. The monotonic property of the

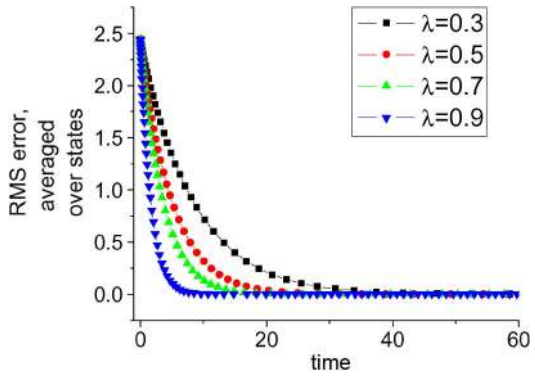


Fig. 5. RMS error of the DP network for computing shortest paths in a 10 × 10 mesh network with different λ values, where λ is the reciprocal of the network time constant.

continuous-time network has been shown, and the network convergence has been discussed. The convergence rate of the network depends on λ, which is the time constant that varies based on the different implementation platforms. In the following, the embedding of the DP network in NoC, to provide shortest path computation on-the-fly, and the dynamic routing, to enhance the network utilization, are discussed.

IV. NoC Routing With DP Network

A. Routing Architecture

An interesting feature of an on-chip communication network or NoC is that the communication network itself defines the graph of the shortest path problem. This provides an opportunity to compute the optimal path by embedding a DP unit at each node. Unlike the general computer network, the shortest path routing computation is solely attributed to the processors at each node. The NoC environment demands tighter timing and performance constraints as well as more flexible implementation methodologies, which can be achieved by implementing a DP-network architecture.

The DP network shown in Fig. 6 consists of distributed computational units and links between the units. The topology of the network resembles the defined graph topology, which is the communication structure of an NoC. At each node, there is a computational unit, which implements the DP unit equations in (10). The numerical solution of the unit will be propagated to the neighboring units via the neighborhood interconnects. The DP network is tightly coupled with the NoC, and each computational unit locally exchanges control and system parameters with the tile or core. The DP network quickly resolves the optimal solution, as will be shown later in this paper, and will pass the control decisions to the router or other controllers in the tile, while the real-time information, such as average queuing time, will be inputted to the computational unit.

The DP network presents several distinguishing features to an on-chip communication system. First, the distributed architecture enables a scalable real-time monitoring functionality for the NoC. Each computational unit acquires local information, and, through communication with neighboring units, a global optimization can be achieved. Second, because of the simplicity of the computational unit, the dedicated DP network provides a

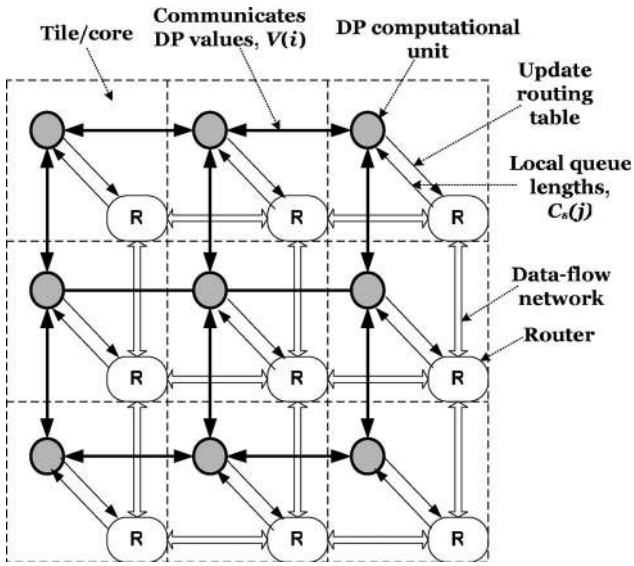


Fig. 6. Example of a 3 by 3 mesh network coupled with a DP network.

real-time response and will not consume any data-flow network bandwidth. Third, because of the convergence property, as discussed in Section III-C, the DP network provides an effective solution to optimal path planning and dynamic routing.

1) DP Routing Mechanics: Consider a node–table-routing architecture in which the routing table is stored at each router. The destination of the header flit will be checked, and it will decide the routing direction based on the routing-table entries. In contrast to the table-based routing in which a routing algorithm computes the route or next hop of a packet at runtime, algorithmic routing is more restrictive to simple routing algorithms and can only be applied on regular topologies, such as a mesh topology. The routing-table approach enables the use of per-hop network state information, such as queue lengths, to select among several possible next-hop at each stage of the route.

Algorithm 1 presents an algorithm for updating the routing table with a DP network. At each node unit, there are k inputs from the k neighbor nodes for the expected costs. The output of the unit at node n_i is the updated expected cost $V(i,j)$ and is sent to all adjacent nodes. The main algorithm is outlined in lines 4–10. For each destination j and direction k, the expected cost will be computed, and the minimum cost will be selected, as stated in line 8. The optimal direction for routing is selected and used to update the routing table, as stated in line 9. Although the algorithm consists of two for loops, this can be realized in a hardware with a parallel architecture, and the computational-delay complexity can be reduced to linear.

Algorithm 1 Update routing table for destination n_j

1: **Inputs**: $V(i,j)$, $i \in \mathcal{N}(i)$, where $\mathcal{N}(i)$ returns all neighbor nodes of n_i, and $i = 1, 2, \ldots, N$
2: **Outputs**: $V^*(i,j)$
3: **Definitions**:
n_i is the current node;
$C_{i,k}$ is input queue-length node i from direction k
4: **for all** i such that $n_i \in \mathcal{V}$ **do**

5: **for all** k directions such that $k \in \mathcal{N}(i)$, where $\mathcal{N}(i)$ return all neighbor nodes of n_i **do**
6: $V'(i,k,j) = V(k,j) + C_{i,k}$
7: **end for**
8: $V^*(i,j) = \min_{\forall k} V'(i,k,j)$
9: $\mu(i,j) = \arg\min_{\forall k} V'(i,k,j)$ {Update routing table}
10: **end for**

Many routers use routing tables either at the source (source routing) or at each hop (node–table routing) along the route to implement the routing algorithm. In adaptive routing, the routing table is updated dynamically or periodically, such that the communication traffic can be altered subject to the choice of switching mechanisms. The DP network does not interact or interfere with the packet-switching mechanisms but alter the routing table at runtime. Also, a mesh-network topology will be used throughout this paper for illustrating the idea. However, the proposed methodology is not limited to the mesh topology, and simple modifications can be made for tackling network of different forms, such as torus, butterfly fat tree, and other custom-designed topology, based on the flexible routing-table-based design. Also, the numerical accuracy of the cost estimates might affect the network performance. Due to the nature of the DP-based decision making, the absolute cost is not crucial to the decisions but the difference between the costs. A reasonable bit width is adopted, e.g., 8 b, to be allocated to realize the DP computation throughout this paper.

Deadlock can effectively be avoided by adopting one of the deadlock-free turn model. In this paper, the west-first [29] turn model is used. It prohibits all turns to the south–west and north–west direction. The dynamic-routing scheme will be switched to XY routing whenever the destination node is within these directions. In this case, the north–west and south–west turns are removed, and thus, the routing dependences will never form a cycle in the network. Alternatively, other turn models, such north-last, can also be applied in the DP network to avoid deadlock, with a similar performance, at the designer's disposal.

2) DP Network Computational Complexity: The delay of the DP network converges to an optimal routing solution depending on the network topology, which determines the delay information propagates within the network and the delay of each computational unit. It can be seen that each unit involves $\mathcal{O}(|\mathcal{A}|)$ additions and comparisons, where $|\mathcal{A}|$ is the number of edges. Note that the number of additions corresponds to the number of adjacent nodes, and $|\mathcal{A}|$ is an upper bound, which corresponds to the configuration of a fully connected network. Hence, the worst case solution time is $\mathcal{O}(k|\mathcal{A}|)$, where k is the number of iterations evaluated by each unit. In software computation, k is equal to the number of nodes in the network; thus, $k = |\mathcal{V}|$, which guarantees that all nodes have been updated [23]. However, in hardware implementation with parallel execution, k is determined by the network structure, and $\mathcal{A}$ additions can be executed in parallel. Each computational unit can simultaneously compute the new expected cost for all neighboring nodes. Therefore, the solution time becomes the time for the updated value to be distributed to every other node, and the computational complexity becomes $\mathcal{O}(1)$. Also, it is assumed that the comparator delay is transient in time and is independent of the network size. For a more conservative estimation of computational delay, we can assume a binary-tree comparator to be implemented, and the computational complexity becomes $\mathcal{O}(\log_2(|\mathcal{A}|))$. Consider a mesh network with N nodes with $\sqrt{N}$ rows and $\sqrt{N}$ columns. The longest path in this network is $2\sqrt{N}-1$, which is the minimum time required for updating the expected costs at all nodes. Therefore, the network convergence time is proportional to the network diameter, which is the longest path in the network. The DP-network convergence time for some of the network topologies are summarized in Table II.

TABLE II
CONVERGENCE ANALYSIS OF A DP NETWORK FOR DIFFERENT NETWORK TOPOLOGIES

Size	Topology	DP Convergence (cycle)
N, k	binary k-cube	$n-1$
N	2D Mesh	$2\sqrt{N}-1$
N, k	N-dimension k-ary mesh	$Nk-1$
N	2D Torus	$\sqrt{N}-1$
N	N-node k-ary tree	$2\log_k N - 1$
N, k	k-ary N-cube	$Nk/2$
N, k	k-ary N-flies	$N-1$

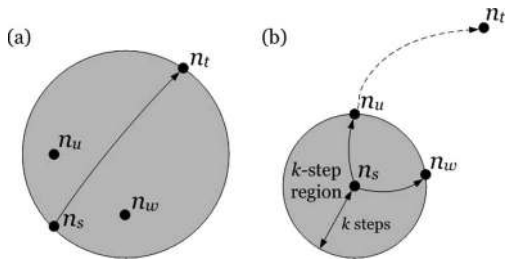


Fig. 7. Comparing the two routing strategies, where the shaded area represents the nodes covered in the routing table and n_s is the source and n_t is the destination. (a) Optimal decision can be made at n_s. (b) Since $V(s,u) \leq V(s,w)$ and the Manhattan distances for n_u to n_t and n_w to n_t are the same, n_u is selected as transition node in the suboptimal path to n_t.

B. Optimality and Memory Tradeoff

One concern for the table-based routing mechanics is the routing-table size, which requires allocation of memory or registers. Even though the adaptive routing brings in substantial advantage in routing delay and throughput, the memory requirement could sometimes become a hindrance for the system to scale up [16]. In this Section, a new method, namely, KSLA, is introduced. This method yields a suboptimal solution in dynamic routing but can substantially reduce the memory requirement.

Instead of storing routing decisions for all destinations in a routing table, storing a table that provides optimal decision to local premises can enable a suboptimal path to the destinations with a substantial reduction on the storage requirement. The idea is that each router computes the routing decisions for nodes that are k steps away from the current node. A k-step region is shown in the shaded area in Fig. 7(b). If the destination is within the k-step region, an optimal decision is readily available in the routing table. Otherwise, a transition node n_u is selected

such that the sum of the DP value to the transition node and the Manhattan distance from that node to the destination is the smallest. These procedures repeat at each hopping step, and eventually, the packet arrives at the destination in a suboptimal route. Fig. 7 shows the two strategies graphically.

Algorithm 2 KSLA Routing Algorithm
1: **Inputs**: Destination node n_t
2: **Outputs**: Routing direction $\mu(s,t)$
3: **Definitions**:
n_s is the current node;
$D(s,t)$ returns the number of steps from s to t;
$\mu(s,t)$ returns the routing direction of destination t at node s;
$k(s)$ returns a set of nodes that are k steps away from s;
$\mathcal{M}(i,j)$ returns the Manhattan distance from n_i to n_j.
4: **if** $D(s,t) \leq k$ **then**
5: **return** $\mu(s,t)$
6: **else**
7: **for all** nodes i such that $i \in k(s)$ **do**
8: $V'(s,i,t) = V(s,i) + \mathcal{M}(i,t)$
9: **end for**
10: $\mu(s,t) = \arg\min_{\forall i \in k(s)} V'(s,i,t)$
11: **end if**
12: **return** $\mu(s,t)$

The KSLA algorithm is presented in Algorithm 2. The inputs are the destination nodes, which are the same as the router designed for the global optimal path planning. For every flit or packet, the algorithm checks whether this destination is within the k-step region. This can be achieved differently for different topologies. For a mesh, this can be checked by analyzing the coordinates and comparing the Manhattan distances. Extension of KSLA to irregular and other topologies requires implementation of other heuristics, which will be studied in our future work. This step is line 8 in Algorithm 2. If the destination is within the k-step region, the optimal routing decision can be readily retrieved from the routing table. If the destination is outside the region, which is not covered in the routing table, the algorithm finds a node within the region that is closest to the destination and with minimal cost. In line 10, the condition ensures that the node chosen is the closest to the destination. Lines 7–10 are aiming to find a node that is leading to the destination node with the minimal expected cost. Finally, in line 11, this node within the region will be output as the next-hop direction.

With the optimal routing scheme, the total cost to go from node n_s to n_t is

$$V^*(n_0,t) = \min_{\forall n_i \in P^m_{n_0,n_m}} \left\{ \sum_{j=1}^{m} C_{i-1,i} \right\} \tag{20}$$

where $i = 0, 1, \ldots, m$ and $n_m = n_t$. In other words, each router is able to look ahead for all possible paths $P^m_{n_0,n_m}$ to the destination and choose the one with minimal delay. For the KSLA approach, the routers can only look ahead for k steps

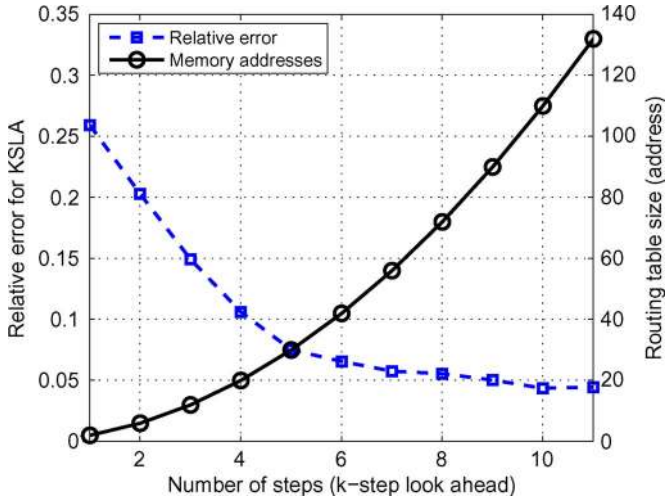


Fig. 8. Theoretical estimates for the approximation error of the KSLA approach with respect to optimal DP values and routing-table size in terms of the address space for the corresponding k values.

at each round. Therefore, the total expected cost $W^k(n_0,t)$ becomes the sum of the $\lfloor m/k \rfloor$ rounds of k-step propagations plus the expected cost of the last round, which requires steps that are less than or equal to k

$$W^k_t(n_0) = \sum_{l=0}^{\lfloor m/k \rfloor} \min_{\forall n_i \in P^k_{n_{lk},n_{(l+1)k}}} \left\{ \sum_{j=lk+1}^{(l+1)k} C_{i-1,i} \right\} + \min_{\forall n_i \in P^m_{n_{\lfloor m/k \rfloor k},n_m}} \left\{ \sum_{j=\lfloor m/k \rfloor k+1}^{m-\lfloor m/k \rfloor k} C_{i-1,i} \right\} \tag{21}$$

where $m \geq k$, $i = 0, 1, \ldots, m$ and $n_m = n_t$. Suppose that the intermediate nodes in the KSLA are the same as those in the optimal path $P^m_{n_0,n_m}$, the path produced by KSLA is the optimal. In this case, the lower error bound for KSLA is zero, with $W^k(n_0,t) = V^*(n_0,t)$. Furthermore, the expected cost between the optimal and KSLA cases have an interesting proven[2] relationship, which can be expressed by the following:

$$W^{k-1}(n_0,t) \geq W^k(n_0,t) \geq V^*(n_0,t) \tag{22}$$

where $m \geq k > 1$. This expression implies that the KSLA approximation error decreases monotonically when k increases. Note that there is no theoretical upper bound for the expected cost for the KSLA approach. If the packet is trapped at a node with a single path to the destination and this path is faulty, the packet will not reach the destination. Similar to other routing algorithms, such as XY and odd–even, backtracking or special rescue routines are required to help the packet to escape from the trapped node. Nonetheless, this situation is rare, and the KSLA can approximate the optimal path in most cases, as shown in the Monte Carlo simulation.

A Monte Carlo simulation has been performed to verify the theoretical results. The relative error of KSLA with respect to the optimal DP values and with different parameter k is

[2]This can be derived using the inequality $\min_{\forall P^m_{n_0,n_2}} \{C_{0,1} + C_{1,2}\} \leq \min_{\forall P^k_{n_0,n_1}} C_{0,1} + \min_{\forall P^{m-k}_{n_1,n_2}} C_{1,2}$, where $C_{i,j} \geq 0, \forall i \to j \in \mathcal{A}$

shown in Fig. 8. For each k, the optimal path cost and the cost using KSLA are obtained and computed. The relative error is equal to the differences between the path costs using the two approaches. The figure presents an average relative error of 1000 networks with randomly generated path costs. The result consistently shows the monotonicity of the parameter k in KSLA. Also, it is interesting to observe that the error decreases drastically between $k = 1$ and $k = 4$. For the case of $k = 4$, the error can be reduced to 10%. Consider the substantial requirement in memory, a relatively small k in KSLA can already provide a good-quality suboptimal routing solution.

For any n-node network, the memory addresses required can be reduced to $k(k + 1)$, where $k \leq \sqrt{n}$ and a 4-ary network topology is assumed. In general, there are $2k(k + 1)$ nodes within the k-step region. Using the west-first turn model to avoid deadlock, only $k(k + 1)$ destination cost are required to be evaluated and stored. Selecting an appropriate k enables a tradeoff between memory consumption and routing optimality at the designer's disposal. Fig. 8 shows the size of the routing-table requirement for each k, as well as the relative error for the KSLA routing. For the case of $k = 4$, the number of memory addresses required is only 20 for the KSLA approach versus 63 for a full routing table.

V. Results and Discussion

A. Simulation Environment

In order to perform a complete evaluation of the proposed routing algorithm, the open Noxim [30], which is an open-source SystemC simulator for NoC of different structures, is employed. The Noxim simulator provides a virtual cycle-accurate NoC architectural model where various performance metrics, including throughput and delay of the on-chip communication methodologies, can be evaluated. In order to evaluate the performance of the proposed DP network, additional ports for communicating the DP values are added to the Noxim NoC router architecture. Routing tables and the table-updating scheme, as described in the previous section, are also introduced to the simulator. A new DP routing function is implemented for realizing both the global path planning and KSLA. Although a mesh topology is considered in our experiments, the Noxim-based NoC architecture can be easily extended to other topological structures by modifying the interconnection of ports of the routers. The traffic-pattern benchmarks embedded in Noxim are used for the routing performance evaluation. These traffic patterns, such as hot-spot random traffic and transpose, provide a comprehensive evaluation for the routing capability, as shown in other related works [13].

By varying the packet injection rate, different routing algorithms produce different average packet-delivery delay and saturation point. The average packet-delivery delay is used as a metric to evaluate the routing algorithm. The DP network provides the shortest path planning, by minimizing the packet-delivery delay at every node. For a mesh topology, the convergence time of the network is $2\sqrt{n} - 1$ cycles. The sampling frequency of the DP network has to be aligned with this convergence time. Therefore, the cost and routing-table updating periods are also the same as the network convergence time, which is $2\sqrt{n} - 1$ cycles. Also, the maximum packet-delivery delay is used to evaluate the routing performance, which is important for NoC real-time applications. The experiments carried out refer to an 8×8 size NoC. Traffic sources generate 8-flit packets with an exponential distribution, the parameters of which depend on the packet injection rate. The first in, first out (FIFO) buffers have a capacity of 16 flits. Each simulation was initially run for 1000 cycles to allow transient effects to stabilize and, afterward, executed for 20 000 cycles. Since it is a mesh topology, the convergence time of the network is $2\sqrt{n} - 1$ cycles, and thus, it is 15 cycles in this experiment. The updating period for individual routing table is then set to be 15 cycles.

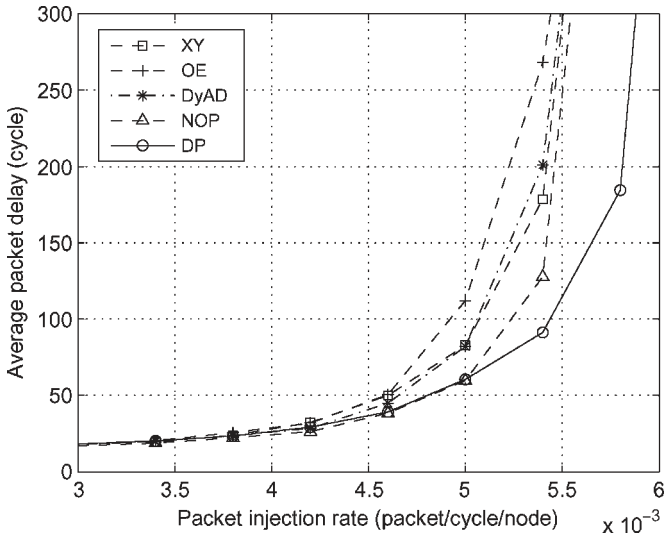


Fig. 9. Average packet delay in random traffic with four hot-spot nodes at the center of an 8×8 mesh network.

B. Results for Average Packet Delay

In order to evaluate the DP-network performance, the average packet delay between the DP and four other well-known routing algorithms, namely, XY [16], DyAD [8], odd–even [12] and odd–even routing with an NoP selection scheme [13], are compared. Each packet is generated randomly from the processors following a traffic pattern and comprises from two to ten flits. A fully optimal DP-network dynamic routing is applied for the experiments in this section. The results for using KSLA will be presented in the next section.

Fig. 9 shows the results of a random traffic with hot spots. This type of traffic pattern is considered to be more realistic than random traffic with uniform distribution. In most of the applications, certain processors or tiles are more frequently accessed than others, such as memory nodes and input/output nodes. In this scenario, there are four hot spots located in the center of the network with 20% hot-spot traffic. When traffic is directed to the center of the network, the central region will be substantially congested. Deterministic routing algorithms, however, would still divert traffic to these regions. Routing algorithms, such as NoP and DP, can slightly outperform other algorithms with deterministic routings. The DyAD routing adopts a scheme that switches between XY and odd–even dynamically and, thus, presents a result in between the two algorithms.

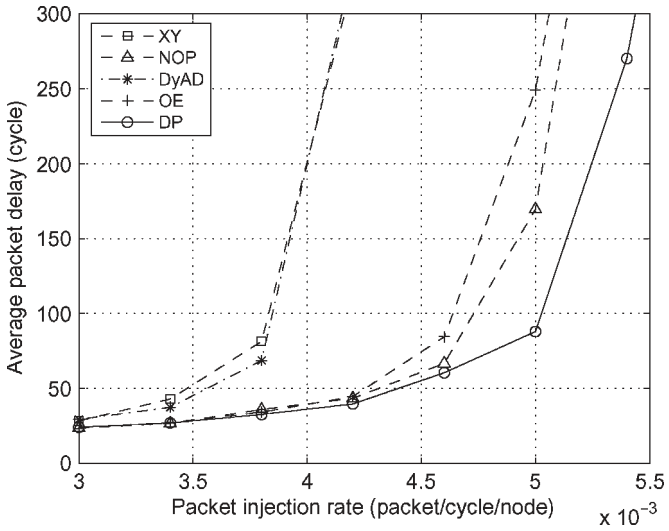


Fig. 10. Average packet delay in random traffic with hot-spot nodes at the four corners of an 8 × 8 mesh network.

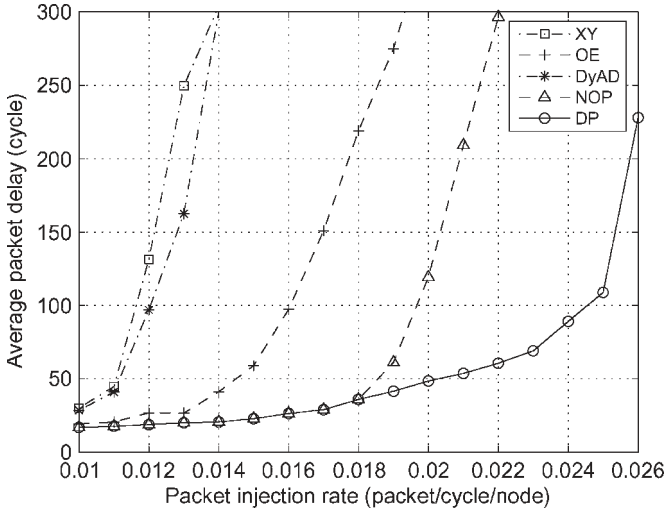


Fig. 11. Average packet delay in matrix-transpose traffic in an 8 × 8 mesh network.

The results are consistent with literature [8]. Fig. 10 shows the results of another hot-spot traffic where the hot spots are located in the corners of the network. In this case, there will be no congested traffic at the center of the network. The dynamic-routing algorithm has a larger degree of freedom to divert the packets to the destination via a potentially smaller delay path. The results demonstrate the performance advantage of adaptive algorithms, such as DP and NoP, with respect to static algorithms, such as XY. These adaptive algorithms provide a larger bandwidth when the network is less congested. The performance advantage from using dynamic routing is more substantial in this case. In particular, DP outperforms the other routing algorithms by 24.7%.

Figs. 11 and 12 show the results for a transpose and butterfly traffic, respectively. The transpose traffic emulates an interesting communication pattern that frequently appears in system-on-chip design, such as traffic in the fast Fourier transform architectures, which is very similar to a matrix transpose [16]. It can be observed that the performances of XY routing and

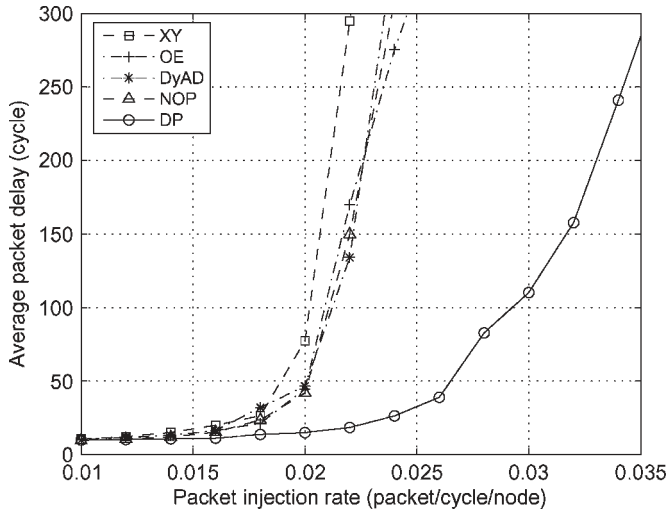


Fig. 12. Average packet delay in butterfly traffic in an 8 × 8 mesh network.

TABLE III
COMPARISONS FOR PACKET INJECTION RATES BETWEEN THE DP AND FOUR OTHER ROUTING ALGORITHMS

Traffic	Packet injection rate (Packet/cycle/node$\times 10^{-3}$) XY	DyAD	Odd-Even	NoP	DP
Random-1	5.06	5.05	4.92	5.24	5.45
Random-2	3.07	3.12	4.52	4.54	5.08
Transpose	10.8	11.1	13.0	14.7	17.3
Butterfly	20.1	20.9	21.0	21.1	29.2
DP Improvement	28.9%	27.5%	18.4%	14.3%	

DyAD are poor due to the congested routes along the horizontal hopping, which coincide with results reported in literature [8], [11], [13]. The DP routing can delay the saturation point significantly because of the optimal path planning, which is able to utilize the throughput of the network effectively. It is interesting to observe that NoP also provides an efficient routing scheme which adapts to the congestions by delaying the saturated packet injection rate to 0.02 in transpose traffic. DP outperforms the other routing schemes by 28.4% and 28.9% for the transpose and butterfly traffic, respectively.

We also compared the maximum packet injection rate for a fixed average delay with different routing algorithms. The results are summarized in Table III. In this scenario, a larger injection rate implies a better utilization of network throughput. The results show that DP outperforms the other routing algorithm by 22.3% with the utilization of real-time traffic information. The other dynamic-routing scheme, odd-even routing with NoP selection, also outperforms the other deterministic routing algorithms, such as XY.

C. Results for KSLA

The recently proposed NoP approach in [13] is a special case of the KSLA. In NoP, each router chooses the routing direction based on the queue information that is two steps away from the current node. A hill-climbing heuristic is implemented for the routing. However, the NoP approach does not compute the DP values for the destination nodes, whereas a score value, which resembles the DP expected delay, is computed on demand. For the DP network, the DP value is computed by the DP network

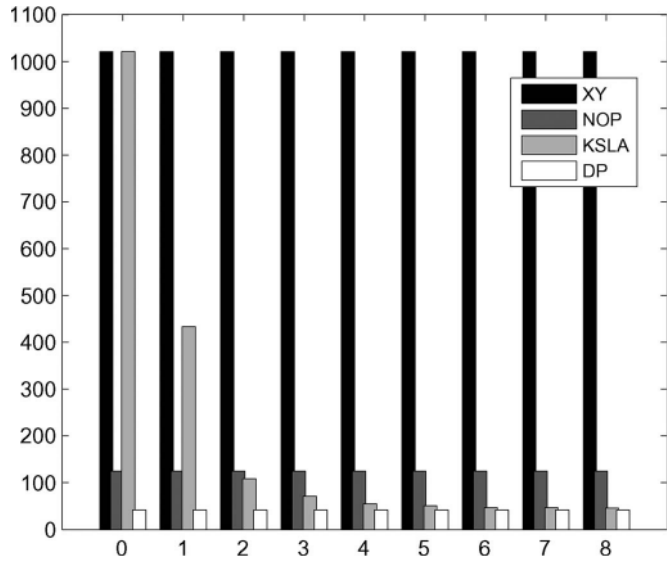


Fig. 13. Comparison of average packet delay between KSLA, odd–even using an NoP selection, XY, and DP routing approaches. An 8×8 mesh network with packet injection rate of 0.02 packet per cycle per node and look-ahead steps for $k = 0, 1, \ldots, 8$ are considered.

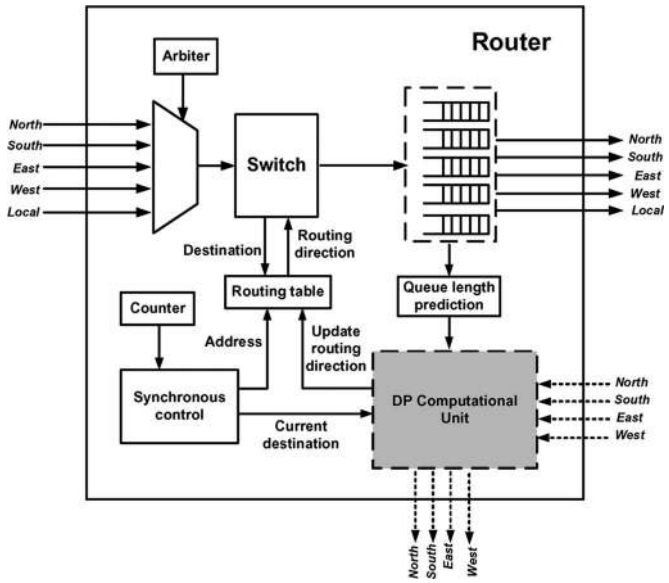


Fig. 14. Schematic design of a standard NoC router except that a DP computation unit is integrated to enable dynamic routing. The "queue-length prediction" block allows realization of different cost-function estimators that provide the cost value for the DP network. The DP computational unit interconnects with other DP units located at adjacent tiles. The DP unit also updates the routing direction in the routing table.

and distributed to all routers. This provides a fast decision time as only a simple lookup table is required when the header flit arrives. In the following, the experimental results of comparing the NoP and KSLA algorithms are discussed.

A special transpose-traffic scenario is considered with a packet injection rate of 0.02 packet per cycle per node. The performances of KSLA with different k, XY, and NoP routings are shown in Fig. 13. When $k = 0$, KSLA has the same performance as XY. This is because the routing table is initialized following the XY routing scheme, and the routing table is never updated. For the case of $k = 2$, KSLA provides a similar performance as NoP (the average delay is equal to 124 for NoP and 108 for DP). This suggests that NoP resembles a special case of KSLA routing, specifically, when $k = 2$. By increasing the k value, the average routing delay is further reduced until it converges to 42 packet delay per cycle per node, where KSLA resembles DP. These results confirm the tradeoff in routing optimality with different k steps, as shown in the earlier Monte Carlo simulation in Fig. 8.

D. Summary

This section has presented a novel DP network for adaptive routing in NoC. The DP network provides on-the-fly shortest path computation using distributed DP and enables dynamic routing based on the real-time traffic conditions and congestions. Also, a KSLA routing strategy has been presented. It can provide tradeoff between routing optimality and memory consumption. Experimental results demonstrate the performance and merits of optimal routing over other deterministic and adaptive-routing approaches, which are based on partial or local traffic information. The optimal DP-network-based routing outperforms XY routing by 28.9% and also improves other adaptive-routing strategies, such as adaptive odd–even, by 18.4%. It is interesting to observe that the new KSLA approach is a generalization of other adaptive-routing algorithm, which applies hill-climbing heuristics for route planning.

Also, the DP network provides shortest path computation conditioned on constant inputs of cost function. Given that the hop costs, which are the queue depths, can change faster than the convergence time of the DP algorithm, then the convergence of the network cannot be guaranteed. Additional circuits are required to smooth out the input costs, such that the fluctuation of the cost function does not affect the convergence of the network.

VI. Hardware Implementation

There is a number of different implementation strategies that can be investigated for the proposed DP network. For example, DP network can be realized using analog circuit which could enable high-performance and low-power on-chip adaptation [31], [32]. Alternatively, digital synchronous and asynchronous designs would result to different hardware and timing characteristics. Investigations on the implementation strategies are out of the scope of this paper. In this section, we aim to study the hardware overhead of a DP implementation based on a simple synchronous network.

In this section, an implementation of the DP network and the dynamic-routing-enabled NoC architecture are presented. Comparisons on the utilization of hardware resources and clock frequencies are discussed.

A. Router Architecture

Fig. 14 shows the architecture of a router, which enables dynamic routing. The router design is similar to that used in NoC [8]. An additional block implements the dynamic-routing algorithm. The queue-length prediction unit captures the queue

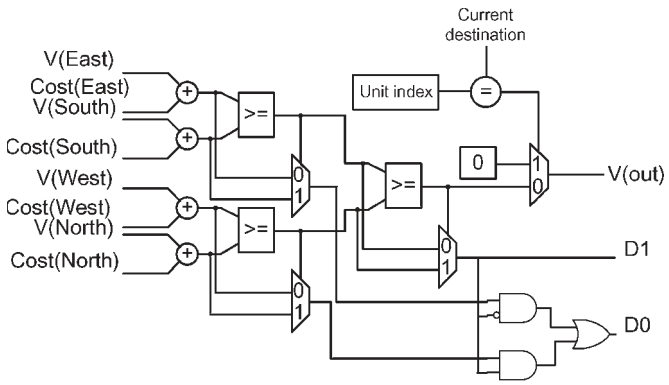


Fig. 15. Realization of the DP computational unit using standard logic. The circuit implements the discrete form of Bellman equation and outputs the updated cost-to-go value. The decision variables can also be obtained via the multiplexer. Since a mesh network is considered here, there are four routing directions that can be encoded using two bits $D0$ and $D1$, where $D1$ is the most significant bit.

length from the input FIFO and evaluates the communication cost for that particular direction. The routing table stores the routing directions, which are constantly updated by the DP network. Successive updating of all entries in the table relies on a synchronous controller, and units in the network are synchronized using counters. The counter provides a reference to indicate which node is regarded as the destination and also provides an address reference to the routing table. The DP unit outputs zero if the current node is the destination; otherwise, it outputs the result of the DP computation. The shortest path computation and optimal routing mechanics are implemented using the DP computational unit, which is shown in Fig. 15. Computation units from different routers are interconnected so as to form a DP network. This figure signifies that the computational network is simultaneously computing the shortest path while the router keeps feeding the new cost estimates into the network.

The shortest path computation requires a minimum operation to evaluate and compare the cost of all actions at each node. Also, adders are required to sum up the costs at the current node and the expected cost associated with the action, as shown in (10). Also, a multiplexer is needed to output the associated action for the minimum expected cost. Therefore, the basic circuit in a DP computational unit comprises four adders, three comparators, and three multiplexers. This circuit can be further extended to provide multiple inputs by increasing the number of adders, minimizers, and operators. The continuous-time formulation of the DP network provides a mathematical framework and convention for convergence analysis that can be applied to study the convergence of the system. The actual implementation can be either analog or digital, which corresponds to the discrete- and continuous-time versions of the network formulation, respectively. The digital network also converges but with a different time constant when compared with the analog realization.

Fig. 16 shows the interconnections between the DP computational units and its neighboring nodes. The interconnections provide a means to deliver the expected values from the neighboring nodes to the DP unit and update the optimal routing direction. The data-flow diagram for the KSLA algorithm is

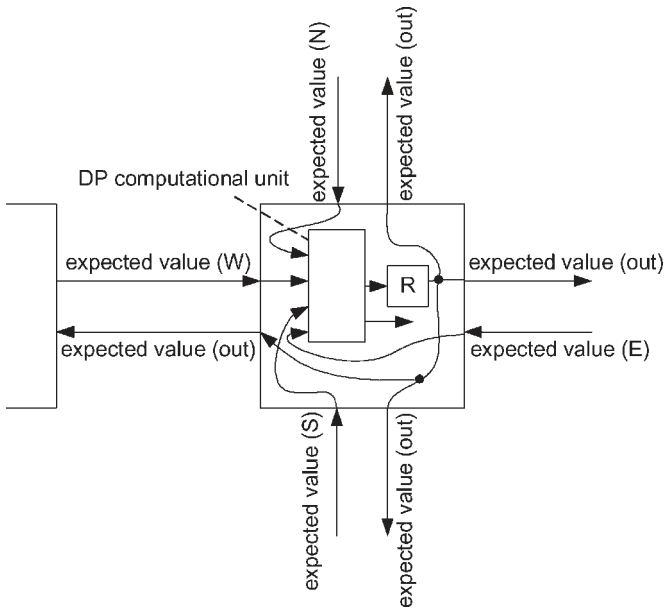


Fig. 16. Interconnecting the computational units with its adjacent nodes.

shown in Fig. 17. When the destination information is obtained from the packet, Manhattan distance to the destination from the current node is calculated. If the distance is smaller than or equal to k, the routing direction to the destination can be directly obtained from the routing table. Otherwise, the nodes within the k-step region are obtained. The nodes in the k-step region are temporary destinations that are k steps away from the current node. For a typical mesh topology, it is relatively trivial to obtain the temporary nodes, which can be done by using the Manhattan distance and lookup tables. One node is selected based on an arbitrary selection scheme. Other selection schemes can be used, such as using the expected costs or traffic. For simplicity, a node is selected randomly in this experiment. The address of this node will be inputted to the routing table to obtain the routing direction.

B. Results of FPGA Implementations

To further evaluate the effectiveness and the hardware cost of the proposed methodology, a DP network is implemented using a Xilinx Virtex-4 XC4LX80 FPGA device. A mesh NoC is implemented using System Generator [33] and synthesis using the Xilinx ISE synthesis tools. The design has been placed and routed to obtain the hardware area-consumption results.

The experiment is designed to evaluate the hardware overhead of the two different routing methods, which are the DP network and KSLA routing. The DP-network routing employs a full routing table, which provides optimal routing directions for all destinations in the network. The KSLA provides routing directions for destinations that are k steps away. The XY routing is also implemented as a reference. Algorithmic routing is employed for computing the routing directions for the XY routing. Similar to other NoC architecture, a wormhole-routing mechanism is implemented.

1) Convergence of DP Network in an FPGA: The performance and network convergence of the DP network in an FPGA realization is studied. DP networks with topologies of 3×3,

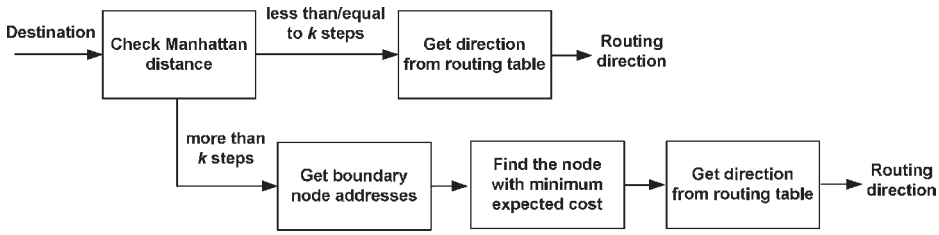


Fig. 17. Data-flow routine for the KSLA algorithm.

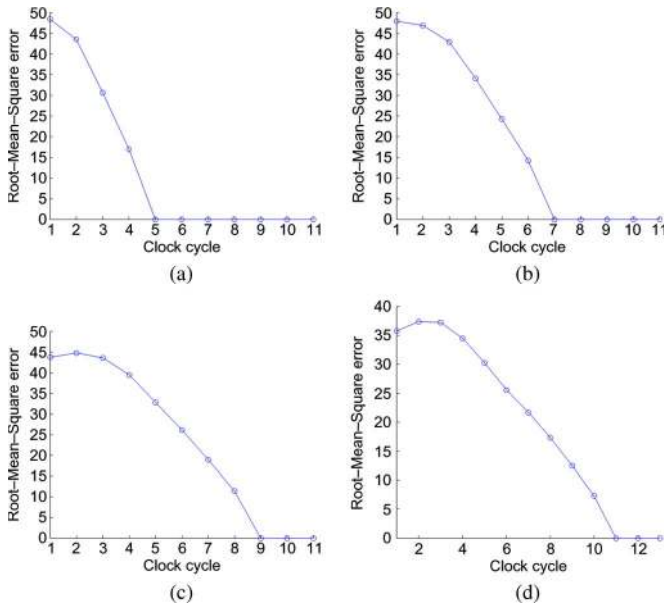


Fig. 18. Convergence of the DP network in an FPGA implementation. The y-axis is the rms error of the optimal values obtained from the value outputs at each computational unit. The x-axis is the clock cycle. The period of the clock cycle is not specified here and varies with different FPGA devices. (a) 3×3 network. (b) 4×4 network. (c) 5×5 network. (d) 6×6 network.

TABLE IV
HARDWARE AREA RESULTS FOR THE DP NETWORKS. RESULTS ARE OBTAINED BASED ON A XILINX VIRTEX-4 XC4VLX40 FPGA

Network format	Area (slice)	Power (mW)	Frequency (MHz)
3×3	337	2.76	100.76
4×4	805	6.84	83.6
5×5	1416	9.06	80.1
6×6	1977	12.4	76.2

4×4, 5×5, and 6×6 are considered. The network-convergence rate for evaluating the shortest path problems can be observed from the outputs of the computational units. These outputs are captured at each time stamp and compared with the optimal values in order for the rms errors to be computed. Fig. 18 shows the errors of the DP network in different clock cycles and in different network topologies. The hardware area and clock-frequency results are summarized in Table IV. The computational units are carefully placed in an FPGA, such that a significant physical separation between the units is introduced. In this case, the operating frequency and power dissipations reasonably indicate the contributions of delay from wires between the units. It can be observed that the network converges to the optimal solution from 5 to 11 clock cycles, depending on the network configurations. The convergence time of the mesh network is bounded by $2\sqrt{n} - 1$, where n is the total number of nodes in a mesh. Suppose that the network is operating at 200 MHz; the convergence time is bounded by $5(2\sqrt{n} - 1)$ ns. The DP network can rapidly evaluate the shortest path and provide optimal path planning for dynamic routing.

TABLE V
HARDWARE AREA RESULTS FOR THE XY, DP, AND KSLA ROUTERS. RESULTS ARE OBTAINED BASED ON A XILINX VIRTEX-4 XC4VLX40 FPGA

Routing algorithm	Buffer size	Area (slice)	Device utilization	Frequency (MHz)
XY	16	344	1%	193
DP (Optimal)	16	409	2%	122
KSLA ($k = 3$)	16	384	2%	127
XY	32	664	1%	140
DP (Optimal)	32	740	4%	120
KSLA ($k = 3$)	32	705	3%	126

2) Hardware Results: The hardware-area consumption for routers with five input and output ports are summarized in Table V. The resource consumption for the XY router in this work is similar to that of the implementation reported in [34]. The overhead of a DP network router is small. The overall area is slightly larger than the XY router. The DP router uses 20.6% more slices than the XY router. For the KSLA router, the area overhead is 40.3%. The KSLA employs more hardware resources for the procedures in evaluating the intermediate nodes for suboptimal routing. In order to verify the memory reduction by using KSLA, we synthesize the design to distributed registers, which are located at the reconfigurable tiles. By measuring the logic utilization, the reduction in memory consumption can be demonstrated. Table V compares the logic consumption between DP and KSLA. The approximation scheme can reduce memory consumption up to 6% for the case when the buffer size is equal to 16. Although additional logics are required to realize the KSLA, the reduction in memory consumption outweighs the extra hardware logic. The router area is still dominated by the input FIFO buffers; the area overhead for the DP network can be negligible. As seen in Table V, the DP overhead is only 23% for a typical buffer size. The DP network with the continuous-time formulation can be implemented using an analog circuit as proposed in [31], in which the hardware area and power consumption could be significantly reduced.

VII. CONCLUSION

This paper has presented a novel DP network for fully optimal routing in NoC. The DP network provides on-the-fly shortest path computation by using distributed DP and updating the routing table for optimal path planning based on the real-time network status. The mathematical formulations and

convergence analysis of the network are presented. Two examples are presented to exemplify the robustness of the network and the rapid resolution of shortest path problems in different network structures. The routing mechanics and the KSLA routing strategy are presented which can provide tradeoffs between routing optimality and memory consumption. Experimental results confirm the performance and merits of optimal routing over other deterministic and adaptive-routing approaches, which are based on partial and local traffic information. The optimal DP-network-based routing outperforms the XY routing by 28.9% and is also better than the other adaptive-routing strategies, such as the odd–even, by 18.4%. It has been observed that the new KSLA approach is a generalization of other adaptive-routing algorithm, which applies hill-climbing heuristics for latency minimization. Moreover, the hardware overhead for a DP network has been examined. It was found that a DP network consumes less than 20.6% of extra hardware area when compared with the deterministic routing algorithms for a standard router design. The results suggest that a DP network offers a new and effective solution for dynamic minimal routing in NoC and can greatly enhance the performance of on-chip communication. The DP-network approach can be further enhanced to enable fault tolerance and dynamic power management in NoCs to reduce power dissipation, which will be investigated in our future work.

REFERENCES

[1] D. Edenfeld, A. B. Kahng, M. Rodgers, and Y. Zorian, "Technology roadmap for semiconductors," *Computer*, vol. 37, no. 1, pp. 42–53, Jan. 2002.

[2] J. Cong, "An interconnect-centric design flow for nanometer technologies," *Proc. IEEE*, vol. 89, no. 4, pp. 505–528, Apr. 2001.

[3] J. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S. Souri, K. Banerjee, K. Saraswat, A. Rahman, R. Reif, and J. Meindl, "Interconnect limits on gigascale integration (GSI) in the 21st century," *Proc. IEEE*, vol. 89, no. 3, pp. 305–324, Mar. 2001.

[4] J. D. Meindl, "Interconnect opportunities for gigascale integration," *IEEE Micro*, vol. 23, no. 3, pp. 28–35, May/Jun. 2003.

[5] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. DAC*, 2001, pp. 684–689.

[6] L. Benini and D. Bertozzi, "Network-on-chip architectures and design methods," *Proc. Inst. Elect. Eng.—Comput. Digit. Tech.*, vol. 152, no. 2, pp. 261–272, Mar. 2005.

[7] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Berg, K. Tiensyrj, and A. Hemani, "A network-on-chip architecture and design methodology," in *Proc. Int. Symp. VLSI*, 2002, pp. 105–112.

[8] J. Hu, "Design methodologies for application specific networks-on-chip," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 2005.

[9] R. Marculescu and P. Bogdan, "The chip is the network: Toward a science of network-on-chip design," in *Foundations and Trends in Electronic Design Automation*. College Park, MD: Now Publishers, 2009, pp. 371–461.

[10] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design tradeoffs for network-on-chip interconnect architectures," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[11] G. Chiu, "The odd–even turn model for adaptive routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 7, pp. 729–738, Jul. 1992.

[12] T. Schonwald, J. Zimmermann, O. Bringmann, and W. Rosenstiel, "Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures," in *Proc. Euromicro Conf. Digit. Syst. Des. Archit. Methods Tools*, 2007, pp. 527–534.

[13] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip," *IEEE Trans. Comput.*, vol. 57, no. 6, pp. 809–820, Jun. 2008.

[14] K. Kobayashi, M. Kameyama, and T. Higuchi, "Communication network protocol for real-time distributed control and its LSI implementation," *IEEE Trans. Ind. Electron.*, vol. 44, no. 3, pp. 418–426, Jun. 1997.

[15] Y. Ishii, "Exploiting backbone routing redundancy in industrial wireless systems," *IEEE Trans. Ind. Electron.*, vol. 56, no. 10, pp. 4288–4295, Oct. 2009.

[16] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA: Morgan Kaufmann, 2004.

[17] D. Wu, B. Al-Hashimi, and M. Schmitz, "Improving routing efficiency for network-on-chip through contention-aware input selection," in *Proc. ASP-DAC*, 2006, pp. 36–41.

[18] M. Palesi, R. Holsmark, and S. Kumar, "A methodology for design of application specific deadlock-free routing algorithms for NoC systems," in *Proc. Int. CODES*, 2006, pp. 142–147.

[19] V. Rantala, T. Lehtonen, P. Liljeberg, and J. Plosila, "Hybrid NoC with traffic monitoring and adaptive routing for future 3D integrated chips," in *Proc. DAC*, 2008, pp. 1–4.

[20] S. Bourduas and Z. Zilic, "Latency reduction of global traffic in wormhole-routed meshes using hierarchical rings for global routing," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Process.*, 2007, pp. 302–307.

[21] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[22] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958.

[23] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001.

[24] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Princeton, NJ: Prentice-Hall, 1989.

[25] F. Hillier and G. Lieberman, *Introduction to Operations Research*. New York: McGraw-Hill, 1995.

[26] K. Lam and C. Tong, "Closed semiring connectionist network for the Bellman–Ford computation," *Proc. Inst. Elect. Eng.—Comput. Digit. Tech.*, vol. 143, no. 3, pp. 189–195, May 1996.

[27] D. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 2007.

[28] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sci.*, vol. 81, no. 10, pp. 3088–3092, May 1984.

[29] C. Glass and L. Ni, "The turn model for adaptive routing," *ACM SIGARCH Comput. Archit. News*, vol. 20, no. 2, pp. 278–287, 1992.

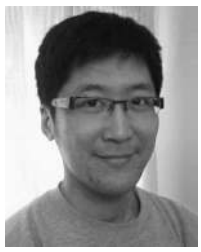
[30] Noxim, Network-on-Chip Simulator, 2008. [Online]. Available: http://sourceforge.net/projects/noxim

[31] T. Mak, P. Sedcole, P. Cheung, W. Luk, and K. Lam, "A hybrid analog–digital routing network for NoC dynamic routing," in *Proc. IEEE Int. Symp. NoC*, 2007, pp. 173–182.

[32] T. Mak, K.-P. Lam, H. S. Ng, G. Rachmuth, and C.-S. Poon, "A current-mode analog circuit for reinforcement learning problems," in *Proc. IEEE ISCAS*, 2007, pp. 1301–1304.

[33] Xilinx System Generator for DSP Version 8.2.02: User Guide, Xilinx, San Jose, CA, 2006.

[34] U. Ogras and R. Marculescu, "'It's a small world after all': NoC performance optimization via long-range link insertion," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 7, pp. 693–706, Jul. 2006.



Terrence Mak (S'05–M'09) received the B.Eng. and M.Phil. degrees in systems engineering from The Chinese University of Hong Kong, Shatin, Hong Kong, in 2003 and 2005, respectively, and the Ph.D. degree from Imperial College London, London, U.K., in 2009.

During his Ph.D., he was as a Research Engineer Intern with the Very Large Scale Integration (VLSI) Group, Sun Microsystems Laboratories, Menlo Park, CA. He was also a Visiting Research Scientist in the Poon's Neuroengineering Laboratory, Massachusetts Institute of Technology, Cambridge. He has been with the School of Electrical, Electronic and Computer Engineering, Newcastle University, Newcastle upon Tyne, U.K., as a Lecturer, since 2010. His research interests include field-programmable gate array architecture design, network-on-chip, reconfigurable computing, and VLSI design for biomedical applications.

Dr. Mak was the recipient of both the Croucher Foundation Scholarship and the U.S. Naval Research Excellence in Neuroengineering in 2005. In 2008, he served as the Cochair of the U.K. Asynchronous Forum, and in March 2008, he was the Local Arrangement Chair of the Fourth International Workshop on Applied Reconfigurable Computing.

Peter Y. K. Cheung (M'85–SM'04) received the B.S. degree with first class honors from the Imperial College of Science and Technology, University of London, London, U.K., in 1973.

He was with Hewlett Packard, Scotland. Since 1980, he has been with the Department of Electrical Electronic Engineering, Imperial College, where he is currently a Professor of digital systems and Head of the department. He runs an active research group in digital design, attracting support from many industrial partners. His research interests include very large scale integration architectures for signal processing, asynchronous systems, reconfigurable computing using field-programmable gate arrays, and architectural synthesis.

Prof. Cheung was elected as one of the first Imperial College Teaching Fellows in 1994 in recognition of his innovation in teaching.

Kai-Pui Lam received the B.Sc. (Eng) degree in electrical engineering from the University of Hong Kong, Shatin, Hong Kong, in 1975, the M.Phil. degree in electronics from The Chinese University of Hong Kong (CUHK), Shatin, in 1977, and the D.Phil. degree in engineering science from Oxford University, Oxford, U.K., in 1980.

He is a Professor with the Department of Systems Engineering and Engineering Management, CUHK. His research is focused on using field-programmable gate array in bioinformatics and neuronal dynamics and on intradaily information in financial volatility forecasting.

Wayne Luk (S'85–M'89) received the M.A., M.Sc., and D.Phil. degrees in engineering and computer science from the University of Oxford, Oxford, U.K.

He is Professor of computer engineering with the Department of Computing, Imperial College London, London, U.K., and a Visiting Professor with Stanford University, Stanford, CA, and Queen's University Belfast, Belfast, U.K. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays. His research interests include theory and practice of customizing hardware and software for specific application domains, such as graphics and image processing, multimedia, and communications.