Research Journal of Applied Sciences, Engineering and Technology 8(21): 2211-2219, 2014
DOI:10.19026/rjaset.8.1220
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2014 Maxwell Scientific Publication Corp.
Submitted: September 03, 2014 Accepted: November 13, 2014 Published: December 05, 2014

Research Article

Hybrid Adaptive Routing in Network-on-chips Using KLSA with Dijkstra Algorithm

M. Muthulakshmi and A. James Albert
Karpagam University, Coimbatore, Tamil Nadu, India

Abstract: The aim of this study is to analyse dynamic programming in large scale, complex networks is more important in the fields of scientific and engineering. Recent applications needs the analysis of scale-free networks with many millions of nodes and edges; presenting a huge computational challenge. Employing distributed networks on-chip infrastructure presents a unique opportunity of delivering power efficient and massive parallel accelerations. Dynamic Programming (DP) network is a massive parallel and high throughput network architecture, which provides real-time computation for shortest path problems. This network combines with the NoC to enable optimal traffic control based on the online network status and, provides optimal path planning and dynamic routing with proposed novel routing mechanics heuristic K-Step Look Ahead (KLSA) in deadlock free architecture. K-step look ahead routing algorithm based calculating the Manhattan distance has some disadvantages and it affects the overall performance of the routing algorithm. In order to overcome aforementioned disadvantages of manhattan distance and improving the efficiency of K-step looks ahead algorithm proposing a dijkstra algorithm for calculating the distance between two nodes. Here in implementation, the results are compared with existing routing schemas or algorithms like XY, DyAD, odd-even, odd-even routing with an NoP selection scheme. The DP network presents a simple, reliable and efficient methodology to enable adaptive routing in NoCs.

Keywords: Dijkstra algorithm, dynamic programming, K-step look ahead, network-on-chip

INTRODUCTION

The shortest path problem is a classic problem in mathematics and computer science with applications. In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized. An example shown in Fig. 1 for finding the quickest way to get from one location to another on a road map; in this case (Cherkassky *et al*., 1996), the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel that segment.

Formally, given a weighted graph (that is, a set Vof vertices, a set E of edges and a real-valued weight function f : E → R) and one element vof V, find a path P from v to a v′ of V so that is minimal among all paths connecting v to v′.

The problem is also sometimes called the shortest path problem, to distinguish it from the following generalizations (Abraham *et al*., 2012):

The single-source shortest path problem, in which we have to find shortest paths from a source vertex v to all other vertices in the graph.

The single-destination shortest path problem, in which we have to find shortest paths from all vertices in the graph to a single destination vertex v. This can be reduced to the single-source shortest path problem by reversing the edges in the graph.

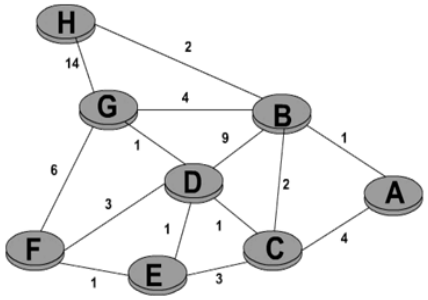

Fig. 1: Example for shortest path algorithm

The all-pairs shortest path problem, in which we have to find shortest paths between every pair of vertices v, v' in the graph.

These generalizations have significantly more efficient algorithms than the simplistic approach of running a single-pair shortest path algorithm on all relevant pairs of vertices.

The most important algorithms for solving the shortest path problem are:

- Dijkstra's algorithm solves the single-source shortest path problems.

Corresponding Author: M. Muthulakshmi, Karpagam University, Coimbatore, Tamil Nadu, India
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

- Bellman-Ford algorithm solves the single-source problem if edge weights may be negative.
- A* search algorithm solves for single pair shortest path using heuristics to try to speed up the search.
- Floyd-Warshall algorithm solves all pairs shortest paths.
- Johnson's algorithm solves all pairs shortest paths and may be faster than Floyd-Warshall on sparse graphs.

Additional algorithms and associated evaluations may be found in Cherkassky *et al*. (1996).

For shortest path problems in computational geometry, some of the practical applications such as travelling salesman problem are the problem of finding the shortest path that goes through every vertex exactly once and returns to the start (Sergey *et al*., 2012). Unlike the shortest path problem, which can be solved in polynomial time in graphs without negative cycles, the travelling salesman problem is NP-complete and, as such, is believed not to be efficiently solvable for large sets of data (see P = NP problem). The problem of finding the longest path in a graph is also NP-complete.

The Canadian traveler problem and the stochastic shortest path problem are generalizations where either the graph isn't completely known to the mover, changes over time, or where actions (traversals) are probabilistic. The shortest multiple disconnected path (Kroger, 2005) is a representation of the primitive path network within the framework of Reptation theory. The widest path problem seeks a path so that the minimum label of any edge is as large as possible. The shortest path algorithm is implemented in the wireless sensor networks also. In this study shortest path algorithm is used in dynamic programming network for routing technique is used.

The Network-on-Chip (NoC) concept replaces design specific global on-chip wires with a generic on-chip interconnection network realized by specialized routers that connect generic Processing Elements (PE). Adaptive routing in a network-on-chip system is unimportant which primarily deals on the deadlock-free and real-time optimal decision making strategy. Dynamic Programming (DP) network is a massive parallel and high throughput network architecture (Mak *et al*., 2011), which provides real-time computation for shortest path problems. This network combines with the NoC to enable optimal traffic control based on the online network status and, provides optimal path planning and dynamic routing with novel routing mechanics K-Step Look Ahead (KLSA) (Lai *et al*., 2009).

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. If the destination is within the k-step region, an optimal decision is readily available in the routing table. Otherwise, a transition node nu is selected such that the sum of the DP value to the transition node and the Manhattan distance fromthat node to the destination is smallest. Mostly Manhattan distance metric is mainly based time series analysis (Miśkiewicz, 2010). In practical implementation cannot be found optimal distance between the nodes because of the time series analysis and it will affect the overall performance of the routing algorithm. A new technique called dijkstra's algorithm is proposed and the objective of the study is to improve the performance of the KSLA algorithm.

LITERATURE REVIEW

Most of the existing routing algorithms are used to find the shortest path from source to destination. Each of these algorithms has unique technique to find shortest path. Some of related routing algorithms are discussed below.

In a dynamic network environment (Wang and Crowcroft, 1992) under heavy traffic load, shortest-path routing algorithms, particularly those that attempt to adapt to traffic changes, frequently exhibit oscillatory behaviors and cause performance degradation. In this study we first examine the problems from the perspective of control theory and decision making and then analyze the behaviors of the shortest-path routing algorithms in details.

The proposed algorithm adopts the enhanced Dijkstra's algorithm for searching the shortest route from the gateway to each end node for first route setup (Zuo *et al*., 2013). A virtual pheromone distinct from the regular pheromone is introduced to realize pheromone diffusion and updating. In this way, multiple routes are searched based on the ant colony optimization algorithm. The routes used for data transmission are selected based on their regular pheromone values, facilitating the delivery of data through better routes. Link failures are then handled using route maintenance mechanism. Simulation results demonstrate that the proposed algorithm outperforms traditional algorithms in terms of average end-to-end delay, packet delivery ratio and routing overhead; moreover, it has a strong capacity to cope with topological changes, thereby making it more suitable for industrial wireless mesh networks.

Dynamic Programming (DP) is a fundamental algorithm for complex optimization and decision-making in many engineering and biomedical systems (Mak *et al*., 2010). However, conventional DP computation based on digital implementation of the Bellman-Ford recursive algorithm suffers from the "curse of dimensionality" and substantial iteration delays which hinder utility in real-time applications. Previously, an ordinary differential equation system was

proposed that transforms the sequential DP iteration into the network is realized using a CMOS current-mode analog circuit, which provides a powerful computational platform for power-efficient, compact and high-speed solution of the Bellman formula.

Experiment of adaptive routing in an NoC system is not negligible and is further tangled by the deadlock-free and real-time optimal decision making requirements and the previously proposed partially adaptive routing approaches utilize local traffic only which directs to a moderate improvement in traffic load balancing and packet latency. Routing adaptations and optimal path planning, which were considered as hardware expensive in computer networks, are rarely studied. In this study, we discussed Dijkstra's algorithm for optimal path in routing.

METHODOLOGY

Shortest path computation: The graph representing all the paths from one node to all the other nodes G = (V, A) where V is set nodes and A is set of edges. With $n = |V|$ nodes, $m = |A|$ edges and a cost associated with each edge$u \to v \in A$, which is denoted as $C_{u,v}$. The total cost of a path $p = < n0, n1, \ldots, nk >$ is defined by:

$$\text{Cost}(p) = \sum_{i=1}^{k} C_{i-1,i} \quad (1)$$

The shortest path of G from n_i to n_j is then defined as any path p with cost as for all constituent edges ni:

$$\text{Cost}(p) = \min \sum_{i=1}^{k} C_{i-1,i} \quad (2)$$

To find the cost of the shortest path from nvto nw, it needs the notion of DP value which is the expected cost from nv to nw. This expected cost is being recursively updated based on the previous estimates until it reaches its optimality condition. This algorithm is known as DP. We denote the DP value for nv to nw at the kth iteration as $V(k)(v, w)$ and $V*(v, w)$ is the optimal DP value.

The Bellman equation becomes:

$$V(k)(v, w) = \min, V(k-1)(u, w) + Cv, u \quad (3)$$

a continuous-time parallel computational network. Here, where, $V(w, w) = 0$. If the recursion is expanded from n_0 to n_k, the DP (Bellman, 1957) value can be expressed as the total cost of the path from node n_0 to node n_k:

$$V_K^*(n_0, n_k) = \min_{\{n_0, n, \ldots, n_k\} \in P_{n_0 i}^k, \; n_k} \left\{ \sum_{i=1}^{k} C_{i-1,i} \right\} \quad (4)$$

where, destination node $n_w = n_l$ and $P_{i,j}^k$ are the set of paths from n_i to n_j , all of which have k edges.

KSLA: Consider the table based routing mechanism in adaptive routing is routing table size. It requires the memory allocation or register for storing process. This requirement becomes difficulty for the system to scale up (Dally and Towles, 2004). The method called KSLA is introduced. This method reduces the memory requirements and provides suboptimal solution. It stores a table that gives optimal decision to local locations. It can facilitate a suboptimal path to the destinations. In KSLA shown in Fig. 2, each router calculates the routing decisions for nodes that are k-steps far from the current node. If the destination is inside the k-step region, best decision is available in the table. Else, a transition node is chosen where the sum of the DP value to node and the Manhattan distance from that node to destination is the smallest. These processes go over at each hopping step. Finally, the packet reaches at the destination in suboptimal route.

Manhattan distance: The Manhattan distance between two nodes is the sum of the differences of their corresponding weights.

The formula for this distance between a node $X = (X1, X2, \text{etc.})$ and a point $Y = (Y1, Y2, \text{etc.})$ is:

$$D(i, n) = \sum_{i=1}^{n} |X_i - Y_i| \quad (5)$$

where,

n : The number of node

X_i and Y_i : The values of the i^{th} node, at node X and Y

Algorithm: KSLA routing algorithm:

1: Inputs: Destination node d_n

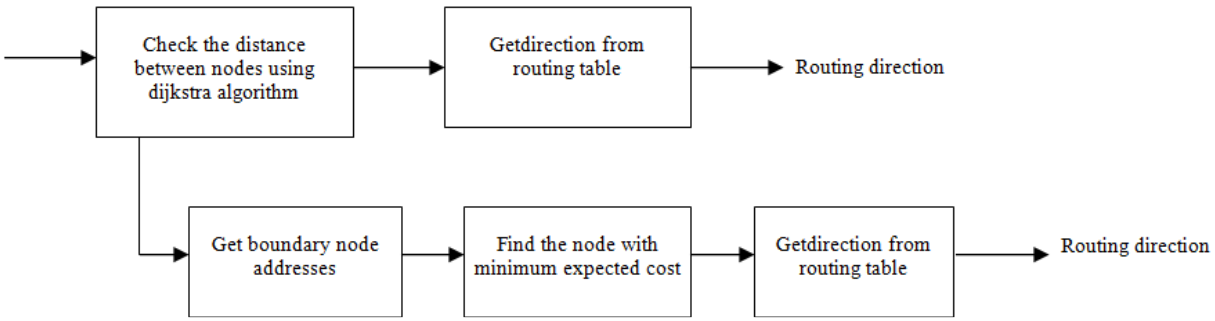

Fig. 2: Data-flow routine for the KSLA algorithm

2: Outputs: Routing direction μ (s, n)
3: Definitions: d_s is the current node
S (s, n) returns the number of steps from s to n
μ (s, n) returns the routing direction of destination n at node s; k (s) returns a set of nodes that are k steps away from s; D (i, j) returns the Manhattan distance from d_i to d_j
4: if S (s, n) ≤k then
5: return μ (s, n)
6: else
7: for all nodes i such that iϵ k (s) do
8: V (s, i, d) = V (s, i) +$D\ (i, n)$
9: end for
10: μ (s, n) = $argmin_{\forall i \epsilon k(s)}$ V (s, i, n)
11: end if
12: return μ (s, n)

Disadvantages of manhattan distance: Values of MD depend on the time series length. However in some applications of time series analysis.

One has to compare distances between time series in different time windows sizes.

Proposed heuristic KSLA: Extension of KSLA to irregular and other topologies requires implementation of other heuristics. In order to overcome this requirement, the heuristic KSLA is proposed in this study. Instead of calculating manhattan distance, this proposed work use Dijkstra's algorithm (Biswas *et al*., 2013) solves shortest path problem. It provide optimal path from source to destination. This algorithm works from the source node, s, it grows like a graph, G, that spans all nodes reachable from S. Nodes are added to G in order of distance i.e., first S, then the node closest to S, then the next closest and so on. The relaxation method (Relax) updates the costs of all the node, v, connected to a node, u.

The Dijkstra algorithm is comprised of the following 5 steps:

Step 1: The process starts from node. Since the length of the shortest path from node a to node a is 0, then $d_{aa} = 0$. The immediate predecessor node of node a will be denoted by the symbol + so that $q_a = +$. Since the lengths of the shortest paths from node a to all other nodes $i \neq a$ on the shortest path are unknown, we put $q_i = -$ for all $i \neq a$. The only node which is now in a closed state is node a. Therefore we write that $c = a$.

Step 2: In order to transform some of the temporarily labels into permanent labels, we examine all branches (c, i) which exit from the last node which is in a closed state (node c). If node i is also in a closed state, we pass the examination on to the next node. If node i is in an open state we obtain its first label d_{ai} based on equation:

$$d_{ai} = \min[d_{ai}, d_{ac} + 1(c, i)] \quad (6)$$

In which the left side of the equation is the new label of node i. We should note that d_{ai} appearing on the right side of the equation is the old label for node i.

Step 3: In order to determine which node will be the next to go from an open to a closed state, we compare value d_{ai} for all nodes which are in an open state and choose the node with the smallest d_{ai} Let this be same node j. Node j passes from an open to a closed state since there is no path from a to j shorter than d_{ai}. The path through any other node would be longer.

Step 4: We have ascertained that j is the next node to pass from an open state to a closed one. We then determine the immediate predecessor node of node j and the shortest path which leads from node a to node j. We examine the length of all branches (i, j) which lead from closed state nodes to node j until we establish that the following equation is satisfied:

$$d_{ai} - l\,(i, j) = d_{ai} \quad (7)$$

Let this equation be satisfied for some node t. this means that node t is the immediate predecessor of node j on the shortest path which leads from node a to node j. Therefore, we can write that $q_j = t$.

Step 5: Node j is in a closed state. When all nodes in the network are in a closed state, we have completed the process of finding the shortest path. Should any node still be in an open state, we return to step 2.

Figure 3 shows that calculation of shortest path for node s to node v using dijkstra algorithm. The algorithm described above can also be used to find the shortest path between two specific nodes. In this case, the algorithm is completed when both nodes are in a closed state.

The Heuristic KSLA Routing Algorithm is explained in algorithm 3. This Heuristic method provides a suboptimal solution and considerably reduces the memory requirement in dynamic routing.

Algorithm: Heuristic KSLA routing algorithm:

1: Inputs: Destination node d_n
2: Outputs: Routing direction μ (s, n)
3: Definitions: d_s is the current node; S (s, n) returns the number of steps from s to n; μ (s, n) returns the routing direction of destination n at node s; k (s) returns a set of nodes that are k steps away from s; DA (i, j) returns the shortest path using Dijkstra's algorithm from d_i to d_j

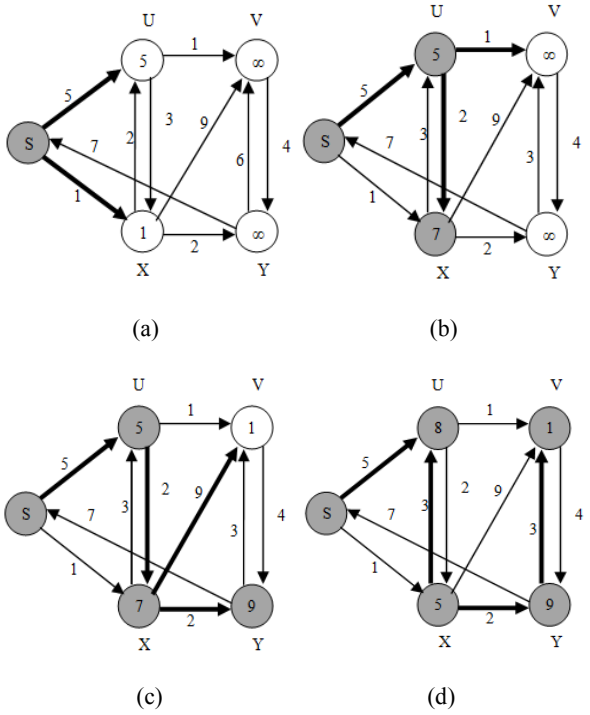

Fig. 3: Example for Dijkstra algorithm

4: if S (s, n) ≤k then
5: return μ (s, n)
6: else
7: for all nodes i such that iϵ k (s) do
8: V (s, i, d) = V (s, i) +DA (i, n)
9: end for
10: μ (s, n) = $argmin_{\forall i \epsilon k(s)}$ V (s, i, n)
11: end if
12: return μ (s, n)

RESULT ANALYSIS

Evaluation methodology: In order to perform a complete evaluation of the proposed routing algorithm, the open Noxim (Noxim, 2008), which is an open source System C simulator for NoC of different structures, is employed. The sample networks for the simulations of the proposed network is shown in the Fig. 4 to 6.

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 Improved KSLA. Although a mesh topology is considered in our experiments, the Noxim-based NoC architecture (Mak *et al*., 2007) 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

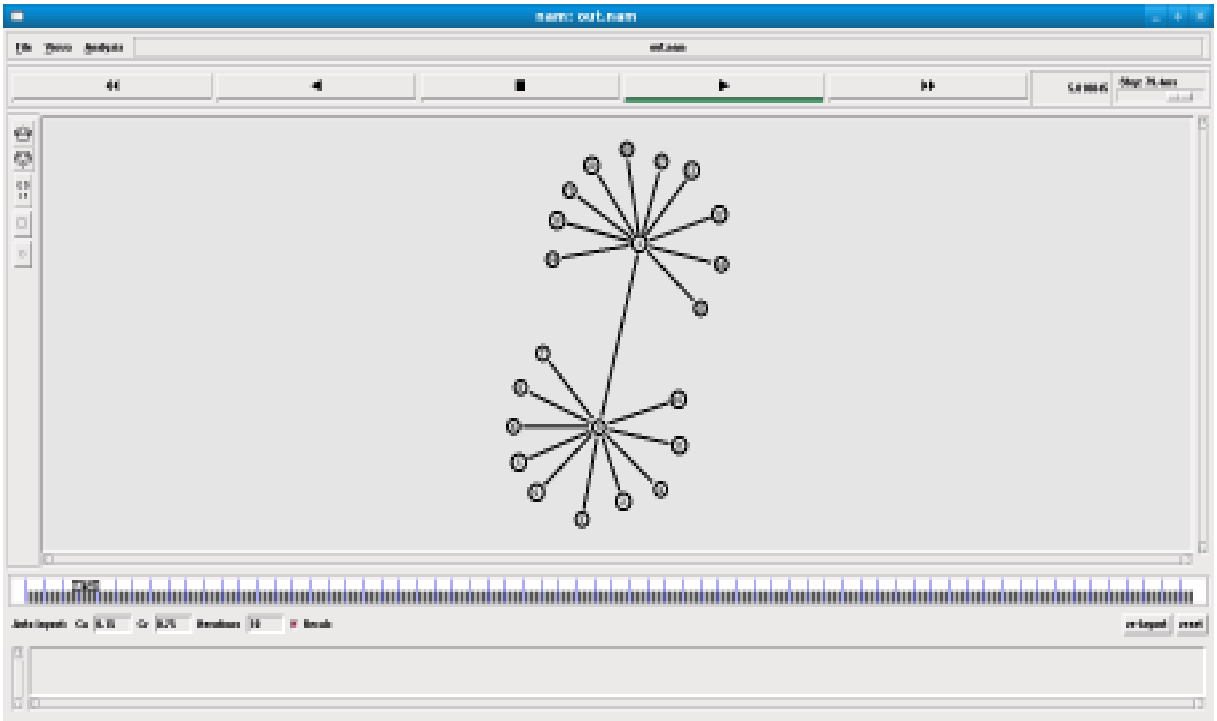

Fig. 4: Sample network 1

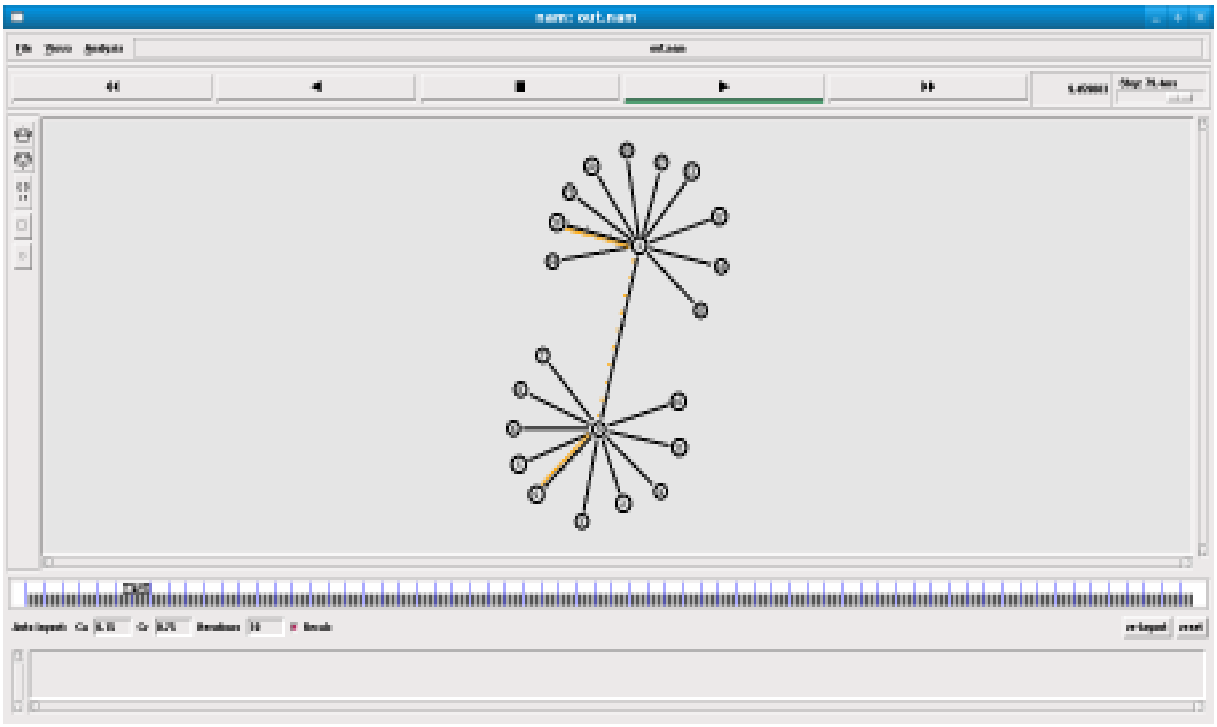

Fig. 5: Sample network 2

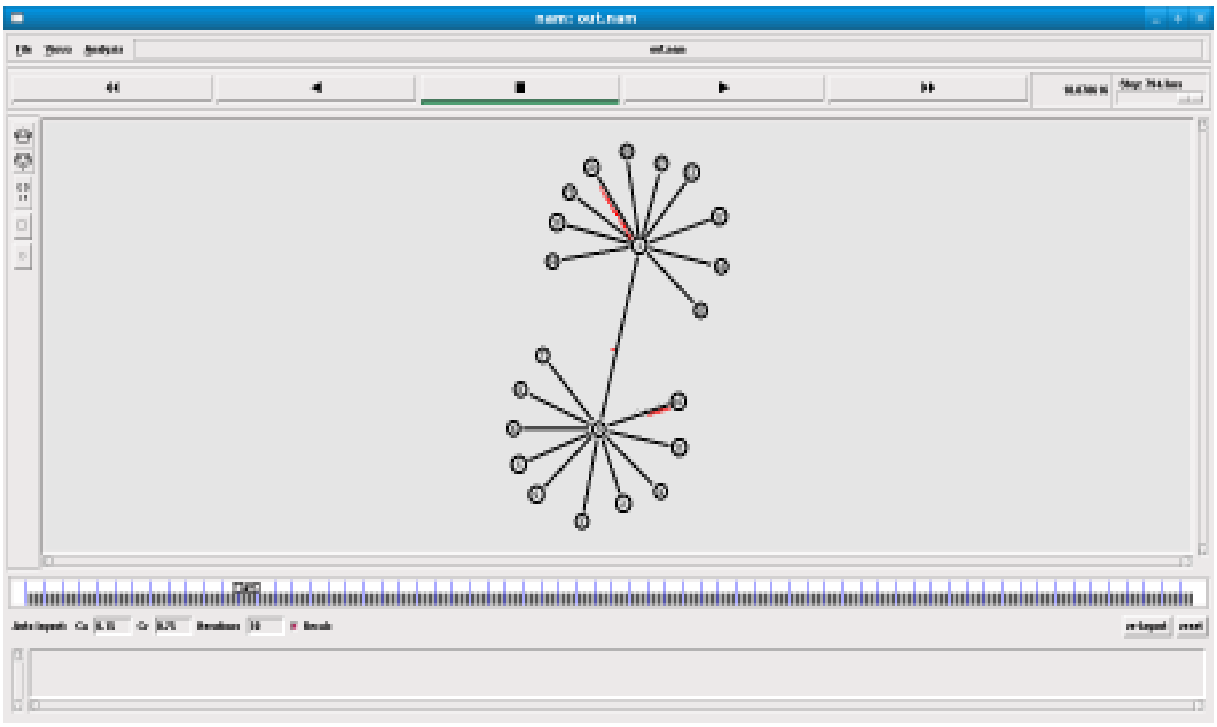

Fig. 6: Sample network 3

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 (Ascia *et al*., 2008).

The packet rejection rate for existing and proposed method is represented in graphical representation in the Fig. 7. By varying the packet rejection rate, different routing algorithms produce different average packet-delivery delay and saturation point. The average packet-

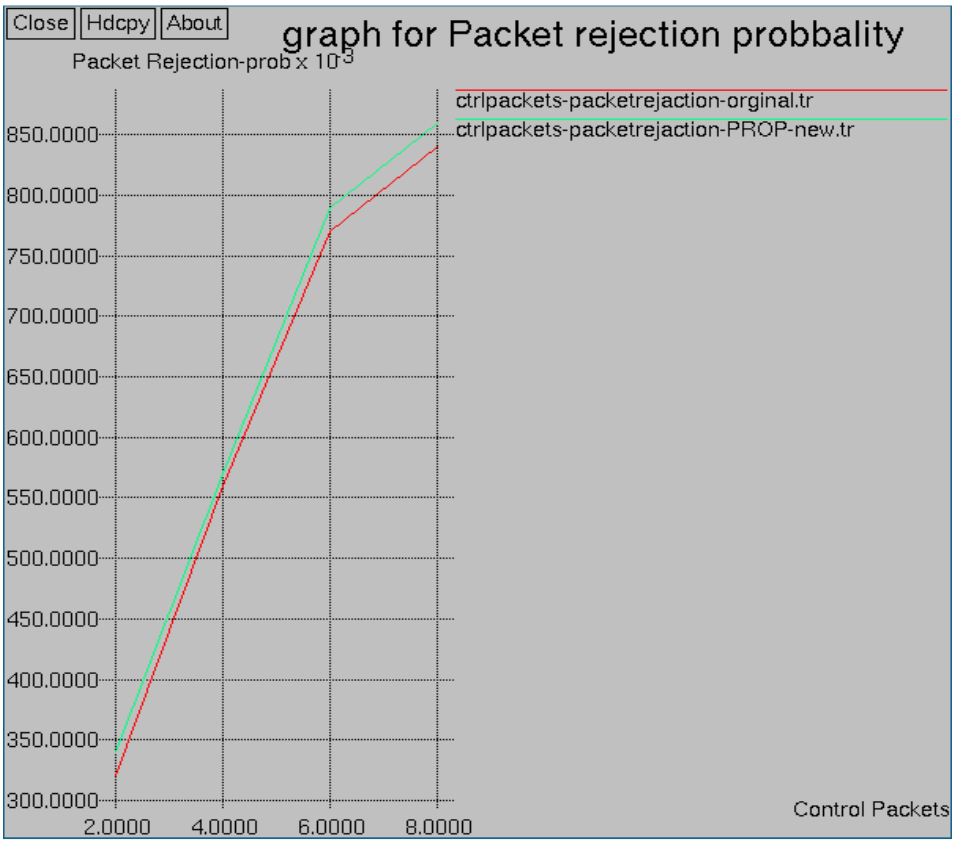

Fig. 7: Packet rejection rate

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 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 20000 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.

The recently proposed NoP approach in Noxim (2008) is a special case of the Improved 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 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 Improved KSLA algorithms are discussed. A special transpose-traffic scenario is considered with a packet injection rate of 0.02 packet/cycle/node. The performances of KSLA with different k, XY and NoP routings are shown in Fig. 2. When $k = 0$, Improved 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 $k = 2$, Improved 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 Improved 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/cycle/node, where Improved KSLA resembles DP.

The average delay comparison between the existing and proposed algorithm is shown in the

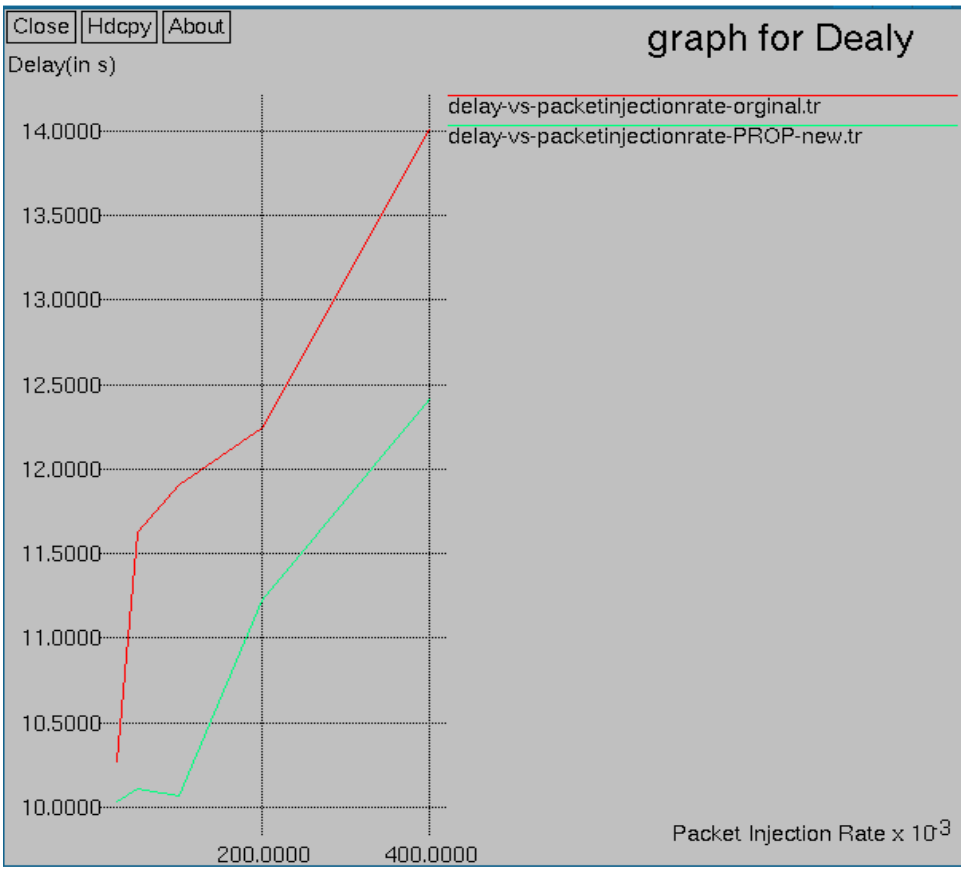

Fig. 8: Average delay comparison between existing and new using a NoP selection, XY and DP routing approaches

graphical representation in Fig. 8. It shows that average delay for the proposed method is low compared to the existing algorithm. The experimental results shows that performance of the proposed algorithm is better than the existing algorithm.

CONCLUSION

This study 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

Abraham, I., A. Fiat, A.V. Goldberg and R.F.F. Werneck, 2012. Highway dimension, shortest paths and provably efficient algorithms. Proceeding of ACM-SIAM Symposium on Discrete Algorithms, pp: 782-793.

Ascia, G., V. Catania, M. Palesi and D. Patti, 2008. Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip. IEEE T. Comput., 57(6): 809-820.

Bellman, R., 1957. Dynamic Programming. Princeton University Press, Princeton, NJ.

Biswas, S.S., B. Alam and M.N. Doja, 2013. Generalization of Dijkstra's algorithm for extraction of shortest paths in directed multigraphs. J. Comput. Sci., 9(3): 377-382.

Cherkassky, B.V., A.V. Goldberg and T. Radzik, 1996. Shortest paths algorithms: Theory and experimental evaluation. Math. Program., 73(2): 129-174.

Dally, W. and B. Towles, 2004. Principles and Practices of Interconnection Networks. Morgan Kaufmann, San Mateo, CA.

Kroger, M., 2005. Shortest multiple disconnected path for the analysis of entanglements in two-and three-dimensional polymeric systems. Comput. Phys. Commun., 168: 209-232.

Lai, D.T., A. Shilton, E. Charry, R. Begg and M. Palaniswami, 2009. A machine learning approach to k-step look-ahead prediction of gait variables from acceleration data. Proceeding of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2009: 384-387.

Mak, T., K.P. Lam, H.S. Ng and G. Rachmuth, 2010. A CMOS current-mode dynamic programming circuit. IEEE T. Circuits Syst., 57(12): 3112-3123.

Mak, T., P.Y.K. Cheung, K.P. Lam and W. Luk, 2011. Adaptive routing in network-on-chips using a dynamic-programming network. IEEE T. Ind. Electron., 58(8): 3701-3716.

Mak, T.S.T., P. Sedcole, P.Y.K. Cheung and W. Luk, 2007. A hybrid analog-digital routing network for NoC dynamic routing. Proceeding of 1st International Symposium Networks-on-Chip (NOCS, 2007), pp: 173-182.

Miśkiewicz, J., 2010. Analysis of time series correlation. the choice of distance metrics and network structure. Proceeding of the 5th Symposium on Physics in Economics and Social Sciences. Warszawa, Poland.

Noxim, 2008. Network-on-Chip Simulator. Retrieved form: http://sourceforge.net/projects/noxim.

Sergey, I., J. Midtgaard and D. Clarke, 2012. Calculating graph algorithms for dominance and shortest path. In: Gibbons, J. and P. Nogueira (Eds.), MPC 2012. LNCS 7342, Springer-Verlag, Berlin, Heidelberg, pp: 132-156.

Wang, Z. and J. Crowcroft, 1992. Analysis of shortest-path routing algorithms in a dynamic network environment. ACM Comput. Commun. Rev., 22(2): 63-71.

Zuo, Y., Z. Ling and Y. Yuan, 2013. A hybrid multi-path routing algorithm for industrial wireless mesh networks. EURASIP J. Wirel. Comm., 2013: 82.