GCA:GLOBAL CONGESTION AWARENESS
FOR LOAD BALANCE IN NETWORKS-ON-CHIP

A Thesis

by

MUKUND RAMAKRISHNA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2012

Major Subject: Computer Engineering

GCA:GLOBAL CONGESTION AWARENESS FOR LOAD BALANCE IN NETWORKS-ON-CHIP

A Thesis

by

MUKUND RAMAKRISHNA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Paul V. Gratz
	Alexander Sprintson
Committee Member,	Eun-Jung Kim
Head of Department,	Costas Georghadis

August 2012

Major Subject: Computer Engineering

ABSTRACT

GCA:Global Congestion Awareness

for Load Balance in Networks-on-Chip. (August 2012)

Mukund Ramakrishna, B.Tech., International Institute of Information Technology,

Hyderabad

Co–Chairs of Advisory Committee: Dr. Paul V. Gratz
Dr. Alexander Sprintson

As modern CMPs scale to ever increasing core counts, Networks-on-Chip (NoCs) are emerging as an interconnection fabric, enabling communication between components. While NoCs are easy to implement and provide high and scalable bandwidth, current routing algorithms, such as dimension-ordered routing, suffer from poor load balance, leading to reduced throughput and high latencies. Improving load balance, hence, is critical in future CMP designs where increased latency leads to wasted power and energy waiting for outstanding requests to resolve. Adaptive routing is a known technique to improve load balance, however, prior adaptive routing techniques either use local, myopic information or mis-informed, regionally-aggregated information to form their routing decisions. This thesis proposes a new, light-weight, adaptive routing algorithm for on-chip routers based on global link state and congestion information, Global Congestion Awareness (GCA). GCA leverages unused bits in existing packet header flits to "piggyback" congestion state information around the network and uses a simple, low-complexity route calculation unit, to calculate optimal packet paths to their destination without the myopia of local decisions, nor the aggregation of unrelated status information, found in prior designs. In particular GCA outperforms local adaptive routing by up to 82%, Regional Congestion Awareness (RCA) by up to 51%, and a recent competing adaptive routing algorithm, DAR, by 8% on average on realistic workloads.

To My Parents and My Sister

ACKNOWLEDGMENTS

It is a pleasure to thank the many people who made this thesis possible.

I owe my deepest gratitude to my advisers, Dr. Paul Gratz and Dr. Alexander Sprintson, for their support and guidance over the two years of my graduate education. Their extensive knowledge about my area of research and unwavering confidence in my abilities were important contributors to ensuring the successful completion of this work.

I would also like to thank Dr. Kim for serving on my committee and for being an excellent teacher.

I am indebted to the members of the CAMSIN research group for cultivating an exciting and inspiring research environment. The sharing of thoughts and ideas with them helped me achieve clarity and make progress in my research. I would like to thank Mark Browning for all his help with the simulator used for this thesis.

All of my colleagues and friends in College Station made the stay here memorable. I would like to thank Garima for all her support during the times that I needed it the most.

Lastly, and most importantly, I wouldn't have been able to compile this work without the unconditional love and support of my family.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

In quest of better performance, today's computer architectures are moving towards multi-core designs in order to address the concerns of rising power envelopes and wire delays. With technology scaling allowing more and more transistors on chip, we are slowly witnessing the emergence of designs with a substantial number of cores being introduced into the market. Current day designs include the recently announced Intel Many Integrated Core(MIC) Knights Corner chip [2] and the Single Chip Cloud Computer [3]. As multi-core designs continue to grow, there is a critical need for a scalable and high-bandwidth on-chip communication fabric to serve as an interconnection network for these components. Traditional solutions like bus-based interconnects and point-to-point links cannot scale beyond a certain number of network elements [4, 5]. This is because naïve scaling of traditional interconnects for large multi-core designs, exponentially increases the latency of program data accesses and sharing. High interconnect latencies translates into idle processor core cycles and wasted power, eschewing the primary motivation for multi-core designs. Reducing interconnect latency, therefore, is critical to achieving scaled performance in future Chip-MultiProcessor (CMP) designs. To address these issues, Networks-on-Chip (NoCs) are emerging as a new and popular solution because of their scalability and relative simplicity of implementation for CMPs and application specific systems-on-chip (SoCs).

The journal model is *IEEE Transactions on Automatic Control.*

A. Networks-on-Chip (NoCs)

Any device which incorporates multiple components on the same die uses some sort of interconnect architecture in order to move data and control signals amongst the components. The propagation of these signals is ultimately responsible for converting the individual blocks into one single cohesive processing unit. The traditional interconnect architectures include bus based and point-to-point links.

Bus based designs connect multiple on chip devices to a common data transmission medium through the interface of an arbiter. As all the users of the bus share the same medium, the purpose of the arbiter is to fairly allot bus bandwidth to requesting sources and synchronize communication. Such a topology is very easy to implement and eats up minimal die area. However, as the number of the devices on the bus increases, we run into problems of scalability. Bus wires which are too long introduce excessive amounts of parasitic capacitance and intrinsic resistance which in turn contribute towards increasing the propagation delay of these wires. Therefore, as we keep increasing the number of blocks connected to the bus the clock period of the design will degrade and we will reach a point beyond which no more blocks can be added without adversely affecting the clock frequency [6].

As transistor technology scales at an incredible pace dictated by Moore's Law, the gate delays also scale down enabling greater density and higher frequencies of operation. However, the wire delays on the chip do not scale at a similar rate. According to the International Technology Roadmap for Semiconductors (ITRS) in Fig. 1, the already substantial gap between the interconnection RC delay and gate delay will only increase exponentially (9:1 with the 65-nm technology as per the ITRS 2003 report [1]) as process technology moves to smaller transistor sizes. Also, because of increasing wire density the parasitic capacitance associated with interconnect wires

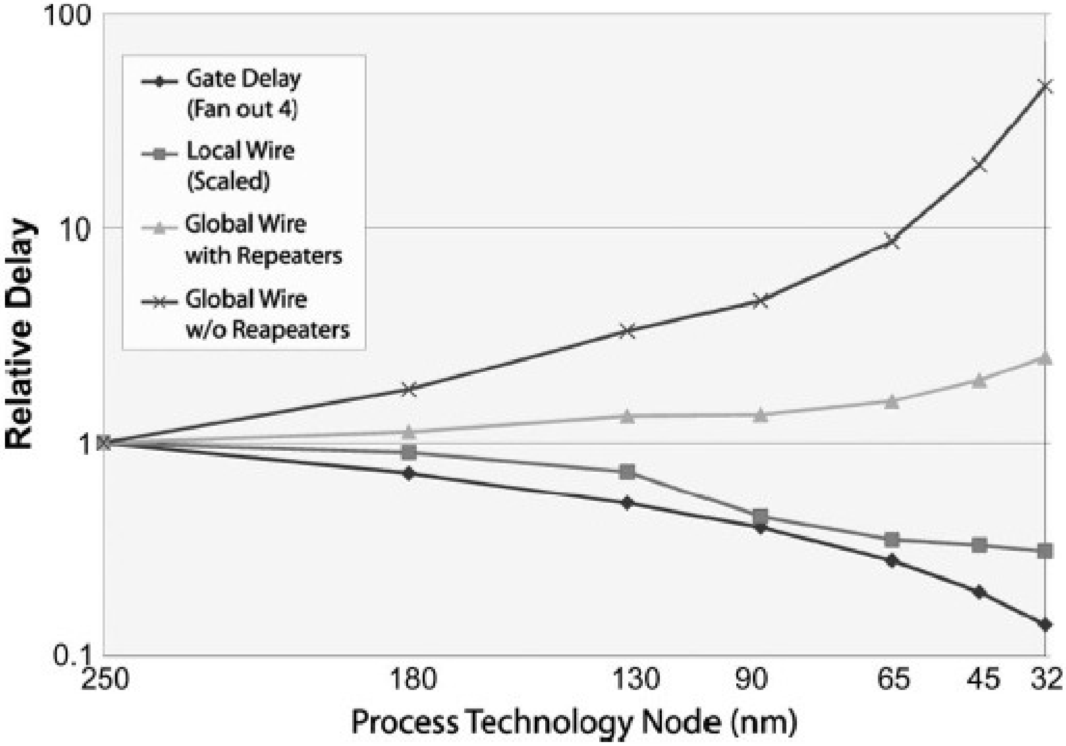


Fig. 1.: Projected relative delay for local and global wires and for logic gates in technologies of the near future [1].

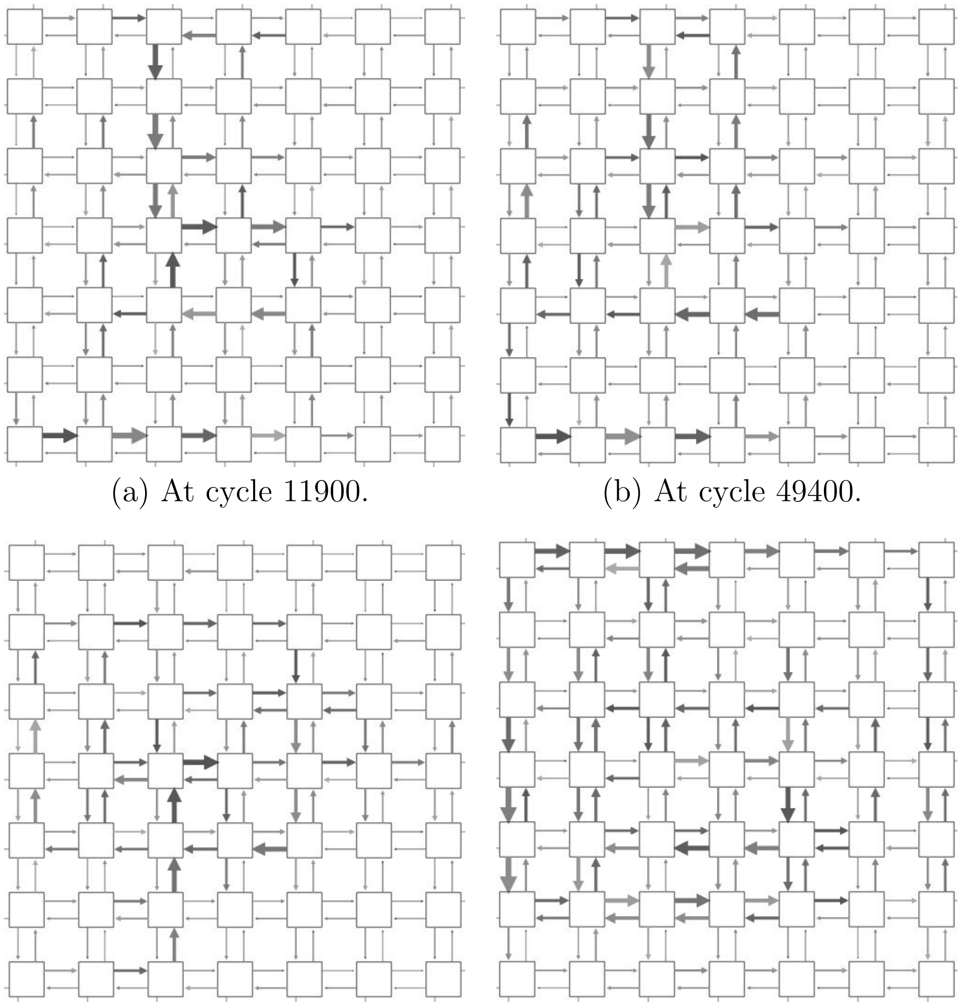
also increases. As a result, interconnects have become the primary limit on productivity, performance, energy dissipation and signal integrity of giga-scale integration [7].

The most popular solution to overcome these issues is that of a packet-switched on-chip communication fabric [5, 8] which is similar in a lot of ways to the current-day off-chip networks. Each processing element is equivalent to a node in the network and the interconnect links are arranged in some form of a network topology like a mesh, ring etc. Data is exchanged in the form of packets and every node consists of a lightweight router to switch these packets from the source node to the destination node.

B. Routing in NoCs

Since multiple paths between source and destination exist, NoCs must implement a routing algorithm to route packets to their destination. This algorithm has implications on the latency and throughput that the traffic experiences. Many current NoC proposals implement oblivious routing algorithms, such as dimension ordered routing (DOR), which route packets irrespective of the state of the congestion in the network. Although these algorithms have low complexity and are deadlock free, they often perform poorly because they produce load imbalances in the network, particularly when under realistic workloads which are often inherently imbalanced. Fig. 2 illustrates the load imbalance due to oblivious routing. The figure shows snapshots of the state of an NoC implementing DOR when executing the *fft* benchmark from the SPLASH-2 suite [9]. In the figure, link utilization is shown for a 100 cycle window during a highly congested program phase. The size and darkness of the arrows indicates the degree of link utilization during the time window. We see here that workload imbalances and the DOR routing algorithm conspire to direct a significant fraction of the traffic to a subset of the nodes in the network, overloading those links while the links which lie on the periphery of the mesh remain underutilized. This disproportionate loading of the links causes both latency and throughput to suffer.

Adaptive routing defines a class of routing techniques which exploit the path diversity of the topology to select a path of least congestion for every source-destination pair. Such techniques are efficient at routing traffic around congested hotspots and balance the load across the network thereby providing better performance over obliv-



(a) At cycle 11900.

(b) At cycle 49400.

(c) At cycle 51700.

(d) At cycle 146200.

Fig. 2.: Network link utilization for SPLASH-2 benchmark, *fft* under dimension-order routing.

ious routing. Typically, adaptive routing techniques pick the path that is least congested from amongst the options present for a source-destination pair. The method that is used to annotate a given path as congested depends on the degree of congestion awareness that a router has as per the routing algorithm. In other words, the degree of congestion awareness specifies how visible the rest of the network is to a particular node. Intuitively, more visibility of the network would enable the routers to make a better decision of selecting the better path. Traditional adaptive routing algorithms use locally available information to approximate the congestion status of the broader network.

This thesis proposes a new *globally*-aware routing policy which uses a novel mechanism for the transmission of congestion information through the network and construct a detailed congestion map of the whole network. This map is then used to compute the best output port for every other node by using the shortest path algorithm in an optimized form. In particular, this thesis makes the following primary contributions:

- Proposes a novel globally-aware routing policy which has low implementation complexity and can efficiently react to changing network conditions. This adaptive routing algorithm has the following characteristics:
    - Uses “back-annotated piggybacking” to propagate congestion information across the network in an efficient manner.
    - Contains a light-weight shortest path route computation technique which is optimized to be easily implemented in hardware.
- Explores the performance benefit gained by the new routing policy on realistic workloads both individually and in a consolidated scenario.

The organization of the thesis is as follows: Chapter II examines previous work in this area, followed by a detailed description of the GCA mechanism in Chapter III. Design implementation issues are discussed in Chapter IV. We then present the results obtained on different traffic patterns in Chapter V and conclude in Chapter VI.

CHAPTER II

RELATED WORK

Adaptive routing techniques choose the path that is least congested from amongst the options present for a source-destination pair. The method that is used to annotate a given path as congested depends on the degree of congestion awareness that a router has as per the routing algorithm. In other words, the degree of congestion awareness specifies how visible is the state of the rest of the network to a particular node.

A. Locally Aware Adaptive Routing

Locally aware adaptive routing techniques make decisions based on local congestion information. The metric of congestion may be free buffer slots, output queue length or free virtual channels (VCs) or more exotic metrics such as crossbar demand or combinations of these. Dally and Aoki make use of the number of free VCs as a congestion metric and pick the port with the higher number of free VCs [10]. The technique proposed by Kim et al [11] uses buffer availability at adjacent nodes as a congestion metric while Singh et al [12, 13] use output queue lengths for the same purpose.

Hu et al. propose an scheme which switches between adaptive and oblivious routing algorithms depending on congestion seen in the network [14]. Congestion is measured in terms of the buffer occupancy at every node. When this occupancy crosses a threshold the downstream node asserts a signal to the upstream node. In order to detect congestion and switch routing algorithms, a node needs to receive this assertion on any one of its output ports. The DyXY algorithm also performs adaptive routing using local congestion metrics [15]. This design chooses from amongst either XY or YX dimension ordered routing based on congestion information and is minimal.

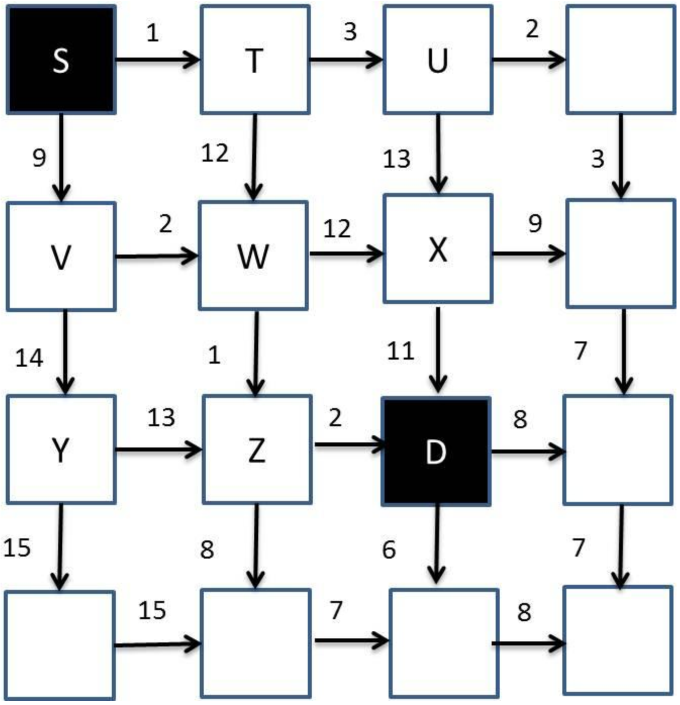


Fig. 3.: Motivation for global awareness.

It is similar to DyAD [14] in the technique employed to gather and measure congestion of a link in the network. By basing the switching of the route selection method on an absolute value rather than a threshold, however, this technique achieves better resolution and would be more sensitive to congestion.

As these techniques use readily available local information only, they have low implementation complexity and have no secondary impact on network traffic due to status information propagation. Such schemes, however, often make sub-optimal routing decisions at a global level because of the myopic view of the network congestion. They are also slower to react to congestion at further links as they rely on network back pressure for information propagation. Fig. 3 highlights these limitations. The figure shows a 4x4 2-D Mesh network with a source node (S) and destination node

(D) for a given packet. The links downstream from node S are marked with a value representing the congestion of that link. If we route a packet from S to D using local adaptive routing, at each hop the link with a lower congestion value will be chosen. In this manner, the path $S \rightarrow T \rightarrow U \rightarrow X \rightarrow D$ on which the sum of congestion values is $1 + 3 + 13 + 11 = 28$ would be selected[1]. For this packet, however, the optimal path is $S \rightarrow V \rightarrow W \rightarrow Z \rightarrow D$ as the total sum of the congestion values on this path is $9 + 2 + 1 + 2 = 14$. Thus, local adaptive picks a more congested path due to greedy decision making based on local lightly loaded links to nodes T and U, such that once it reaches node U, it has only one direction to traverse ($U \rightarrow X \rightarrow D$), given minimal adaptive routing, and that direction is heavily congested.

Global congestion awareness enables complete visibility of the network state and alleviates the issue of greedy sub-optimal decisions. Also, as the decisions are based on a global state, there is no reliance on network back pressure for information of network congestion.

B. Regionally Aware Adaptive Routing

Regional Congestion Awareness (RCA) was the first work to explore adaptive routing based on congestion visibility beyond the adjacent nodes [16]. RCA gathers the congestion information from nodes beyond the adjacent one by propagating them over a light-weight monitoring network. At each hop, the incoming congestion information is aggregated with the local congestion information and then propagated further into the network. Thus every node has a view of the congestion status of a region of the network downstream from each output port for use in selecting the optimal adaptive route for a given packet. RCA weights the congestion values by distance from the

[1] In this work we only consider minimal path adaptive routing, due to the complexity and generally higher latencies of non-minimal adaptive routing.

local node, such that distant links have less impact upon route selection.

Given its aggregated regional view, however, RCA can aggregate information from beyond a given packet's destination node, causing interference in the congestion estimate on an individual packet basis. This interference impacts performance of regional congestion aware routing algorithms when routing packets which have more localized routes. DBAR is a regional adaptive routing algorithm which aims to mitigate this issue by factoring in information about the destination node while selecting the route, thereby reducing the noise aggregation adds [17]. DBAR exploits the high degree of locality that most traffic patterns exhibit by propagating congestion information in the X and Y dimensions without aggregation. As a result, each node has greater state information resolution at the cost of greater implementation complexity. This technique is particularly useful for partitioned many-core architectures, where distant partition congestion information is not useful for routing decisions in the local partition. In unpartitioned networks, however, DBAR's performance is very similar to RCA and hence we do not explicitly compare GCA's performance against it.

Although regional awareness provides a better view of the network than locally aware adaptive schemes, it still falls short of the complete picture. Due to the aggregation of the congestion weights along the whole dimension, the resolution of the acquired information is quite poor, leading to degraded performance. This is because in minimal adaptive routing, the relevance of congestion of a downstream link is highly dependent on whether or not that link will eventually be traversed by the packet. Fig. 3 illustrates the limitations with regionally-aware routing. Again, consider a packet routed from S to D. Here the network makes use of a regionally adaptive routing with visibility along one dimension (e.g. RCA-1D [16]). Instead of basing decision on congestion one hop away, the packet's route is determined by the lesser congested of the two admissible dimensions of traversal. At node S, it compares between the aggregate congestion of all links in the EAST direction against the aggregate congestion of all links in the SOUTH direction. This comparison leads it to pick the EAST direction. Following a similar principle at downstream nodes, it picks EAST at T to reach U from where it has only one admissible dimension to traverse. The path here is same as the one picked by local adaptive, $S \rightarrow T \rightarrow U \rightarrow X \rightarrow D$, and the sum is 28. This sub-optimal route is chosen because RCA considered links in the SOUTH and EAST direction which were 2 hops away from the current node. With minimal routing, for a packet from S to D, these links would never be used as they fall out of the minimal region, however, their congestion status adversely affects the route selection.

Global awareness overcomes the issue of aggregated information by maintaining fine-grained congestion information about the network, helping it avoid considering links outside the minimum path range for a given source-destination pair.

C. Globally Aware Adaptive Routing

Two recent works have proposed globally aware adaptive routing techniques. Adaptive Toggle Dimension Ordered Routing (ATDOR) is a NoC architecture which implements a secondary network to transmit congestion information by each node to a dedicated node, using it to pick between XY or YX DOR for every source destination pair in the network [18]. DAR adds a separate network which the nodes use to communicate their local congestion information with the rest of the network [19]. Every node then determines the amount of traffic that it has to split for a particular destination amongst the candidate output ports based on this congestion information.

Both these approaches introduce a sideband network for gathering and disseminating congestion information. This adds implementation overheads on the design.

In the case of DAR, the monitoring network's congestion information updates are very slow. As a result DAR has a high response time to changing congestion status of the network. Our proposed technique removes the need for a sideband network by embedding ("piggybacking") status information in packet headers. We also implement a low-complexity, low-latency route computation unit, providing a much lower response time to the changing network congestion.

CHAPTER III

GLOBAL CONGESTION AWARENESS(GCA)

Global Congestion Awareness (GCA) is a novel globally aware adaptive routing scheme for NoCs which aims to provide every router a complete picture of the congestion status of every other link in the network. The two main contributions of this technique are in the method of dissemination of the information across the network and the route computation technique that each node employs. A globally aware algorithm can determine the exact amount of congestion that the packet will encounter along the candidate paths instead of a rough estimate of the congestion in its direction of travel. By doing so, it can pick the optimal path as it takes into account each individual link which can be traversed and does not include links which lie outside the minimal routing model. To illustrate, we examine Fig. 3 again to show how GCA picks the optimal path. By utilizing the congestion propagation mechanism, the source node (S) would have complete visibility of the network. Now, it computes the congestion for every possible path to reach destination node (D) by adding up the individual congestion metric of each link on that path. From the set of these admissible paths, it is now left with the simple task of routing the packet along the path whose congestion is least. In the figure, the path $S \rightarrow V \rightarrow W \rightarrow Z \rightarrow D$ is picked as it is determined to have the lowest congestion sum $(9 + 2 + 1 + 2 = 14)$.

A. GCA Model

Ideally, at every hop of the packet's traversal, output port selection would be based upon a perfect view of the network's congestion and thus always pick the best path towards its destination. In a realistic implementation, however, a perfect view is impossible due to the latency incurred in transferring congestion information from one point of the network to another. Furthermore, for packets traversing a considerable number of hops from their source, network congestion close to the destination might change by the time the packet reaches that region. Thus, the decision made at the source might turn out to be sub-optimal. These issues are specifically addressed in our route computation technique.

At a high level, the GCA algorithm consists of the following:

1. At a packet level, a new field in the header flit called the "traffic vector" is introduced which consists of the congestion information as seen by the packet as it traverses the network, along with bits signifying the path taken.

2. At the router level, every node extracts this traffic vector field from the header flit and updates its own current view of the network based upon this new information.

3. Each node maintains a congestion map of the link status of all links in the network. Upon extraction of the traffic vector from a incoming packet, the node performs a route computation of the affected sub-network to determine the best output port for every node in that sub-network. These routes are maintained in a pre-route table.

4. Every node also appends congestion information about its own link into the header before sending out the packet.

5. Packets are routed to the appropriate output port in each node by looking up the pre-route information embedded in the flit. The current node also adds in output port information for the next hop by looking up its routing table. In other words, we employ per-hop routing and every node has a maximum of two output ports for every packet based on a minimal routing model.

In the following Chapters, we present a detailed description of the GCA architecture at a packet-level and router-level. The GCA router-level architecture can be split into two components. The aggregation of incoming congestion values into the local network map of every node and the route computation algorithm.

B. Traffic Vector

One of the primary challenges in implementing a globally-aware adaptive routing algorithm is to employ a scalable and lightweight technique for disseminating the congestion information to every node in the network. Simultaneously, a critical aspect of this technique is providing accurate information about the rest of the network as possible. In all cases, latency of congestion status information present a significant challenge. This latency adversely affects the globally aware algorithm's benefits because if a node is provided with stale information, the routing decisions that it makes would deteriorate the network congestion. Structurally, the routing and congestion information transfer can be considered a form of closed-loop feedback control system and thus it follows that if the feedback is more accurate the routing decision would be better. Furthermore, if there are significant fluctuations in the feedback, the routing decision could potentially thrash, with negative effects on the performance of the system as a whole.

To address these issues, prior globally aware techniques employ a separate sideband network which is dedicated to congestion information. This approach, however, introduces significant extra hardware overhead and increases the complexity of the design. GCA's design goals favor a simpler approach, setting it apart from the prior work. We propose to embed congestion status information in the packet header, "piggybacking" it for propagation in the network.

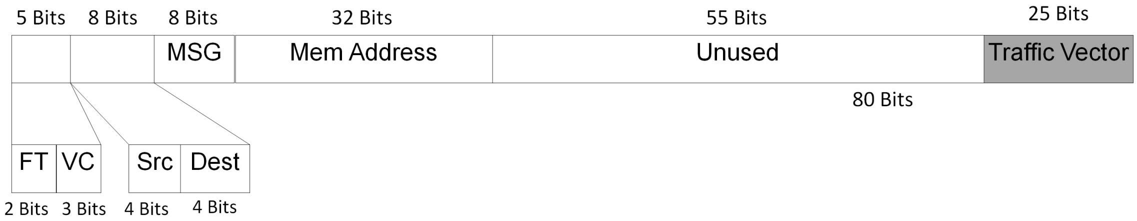


Fig. 4.: Composition of the header flit.

Typically, a significant fraction of a packet's header goes unused due to the wide bit-widths of typical NoC links, as shown in Fig. 4. The header flit contains source and destination node identifiers, and the physical memory address of the cache block. We assume a flow-control overhead of five bits, 3 bits for virtual channel id (VC) and 2 bits for flit type (FT). Given an assumed link width of 128 bits, the remaining ten bytes in the header are unused. We propose to use some of these bits to encode the congestion information from nodes that the header flit has visited. This eliminates the complex overhead of a sideband network, making better utilization of bits that were being wasted. In Fig. 4, the new field required in the header flit is highlighted in gray. At every hop, each node appends information into the traffic vector field of the header flit and then sends it out on its specified output port. The added information consists of two parts: congestion information and input port information. The congestion information corresponds to the link that is connected to the current node and thus it is readily available. The input port information is appended in order to assist with information extraction that downstream nodes will perform on the traffic vector embedded in this flit. At every hop, the traffic vector field is appended with 5 new bits. The traffic vector shown in Fig. 4 is for a 4x4 2D-Mesh where its maximum size could be 25 bits which still leaves 55 unused bits in a network with 128-bit wide links. Generally the maximum bits required scales as

$2 * (\sqrt{N} - 1)$ where N is the total number of nodes in the network. Thus, the 80 bits available in the header would support networks up to 64 nodes. For larger networks, we find the last 16 hops contain sufficient information for GCA to successfully route packets without the need to further extend the traffic vector.

This method does imply that the visibility of the network is limited to the regions which experience the traffic. Every node's visibility is extracted from the header of the packets that pass thorough it. Most of on-chip traffic, however, is request-response packets, thus the links loaded with "upstream" traffic are candidates for transmission of "downstream" traffic too. To reduce the effect of lack of information for unvisited nodes, links for which a node does not receive direct information are approximated to a nominal congestion level, this trade-off helps in keeping the solution realistic (further details are provided in section III.C).

"Back-annotation" is an important detail of GCA's piggybacking implementation which is subtle yet critical to the effectiveness of the technique. Consider Fig. 5, in which a flit originates at node S and is destined for node D. This packet takes the adaptive route shown by solid arrows on the figure. A naïve implementation of piggybacking would have the flit accumulate congestion information about links in the direction of packet traversal (i.e., the links shown in solid arrows). From the point of view of the nodes which are the consumers of this congestion information (all nodes downstream of S), however, the congestion status of the solid arrows is largely useless. This is because, taking node D for example, node D will never send a packet in a minimally adaptive routing algorithm which uses any of the links in solid black. Instead, node D can only use links in the opposite direction from the solid black links. Therefore we propose to "back-annotate" congestion information, i.e. the packet will have encoded information about the links which are represented by dashed arrows.

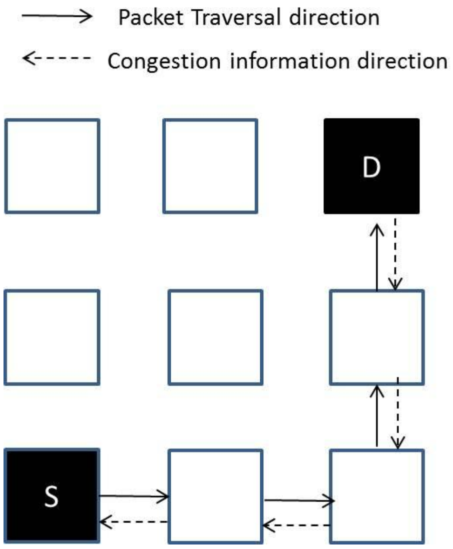


Fig. 5.: Back-annotated piggybacking of congestion information.

C. Congestion Map

The congestion map is at the heart of the GCA routing algorithm. The aim of the map is to provide every node with an accurate enough representation of the congestion of the network's links to utilize in routing decisions. The map contains congestion values denoting the congestion of every link in the network downstream from the current node. These values have been extracted from the header flits of the packets that have traversed the node.

We now describe the steps that take place every time a header flit enters the node. The router reads the traffic vector field off the incoming flits header and inserts this information into its local map. As the vector also has information about the output port that the flit has taken at every previous hop, it is able to trace back its path. For every congestion value in the header, the router overwrites the older value in the map with this new value. In parallel, the router appends the local congestion information about the link in the opposite direction for the port at which the flit entered. It also adds the input port information to the local congestion information and inserts this into the traffic vector field before sending the flit back out into the network.

This technique implies that the information in the congestion map builds up with time. As the node experiences more traffic and can extract the congestion information from all those flits, it fills in more entries in its map. Initially, however, the congestion state of links is unknown before its entries can be filled up through the piggybacked information. If, as in Fig. 2, we signify a congested link as *black* and an uncongested link as *white*, we will assign the initial, unknown state of all links as an intermediate or nominal value which is between the extremes (e.g. *gray*). Over time, as information about links is received, the *gray* links would subsequently be changed to a known state of a value lying between *black* and *white*. By assuming the unknown links to be *gray*, the node will first send traffic through links which are known *white*. If instead links were assigned an extreme value of *white* when state is unknown, packets may be sent to links which are actually currently congested. On the other hand, we could also assume the unknown state of the links to be highly congested or *black*. This could make the algorithm very conservative and would reduce the path diversity presented to the node for routing the packet.

Congestion information staleness is another issue that must be dealt with. It is always possible that some cycles after congestion state information about a given link is received, the condition of the network could change, despite a lack of new information about that link. In this case, the old value present in the local congestion map would be stale, potentially leading to sub-optimal decisions. To address information staleness, the congestion map has a built-in fading mechanism. If a particular entry in the congestion map has not been updated in the last n cycles, we fade (increment or decrement) it towards the *gray* status in steps of x. Intuitively, the link's knowledge is now considered *gray* because we have not received any information about it over a considerable period of time. Discussion of the actual values used for n and x is provided in Chapter IV.

D. Route Computation

So far, we have described how the algorithm proposes to make every node of the network "globally aware" by employing a combination of piggybacking and the congestion map structure. By virtue of these two components, every node now has a picture of the network's congestion at its disposal to route its incoming packets, with the potential to better link utilization and boost performance by reducing the average latency experienced by the traffic or by increasing the saturation throughput. To exploit these potential benefits of global awareness, the router employs a specially tailored shortest path algorithm to compute the best possible output port for the incoming traffic. The underlying idea that we aim to implement is to pick the path with least congestion from the current node to every other node in the network from the set of minimal paths that are admissible.

1. Shortest Path Algorithm

Every link e in the network is assigned a weight w_e which is the congestion metric assigned to that link. A higher weight indicates a higher degree of congestion. Let $\mathbf{P}$ be a set of admissible minimal paths from a source node s to a destination node d. For each path $P \in \mathbf{P}$ we define a congestion estimate $C(P)$ as follows:

$$C(P) = \sum_{e \in P} w_e \tag{3.1}$$

The algorithm will pick a path $P' \in \mathbf{P}$ that satisfies

$$C(P') = \min_{P \in \mathbf{P}} C(P), \tag{3.2}$$

i.e., P' is the path that has the minimum congestion estimate among all paths in $\mathbf{P}$. As the problem is constrained to a 2-D mesh topology we can easily abstract it to that of finding the shortest path in a graph. Each edge weight is a congestion metric and the shortest path would denote the path whose sum congestion is the least. The graph analogy would make sure that all possible options are evaluated. It is important to note, however, that the restriction of minimal routing greatly reduces the size of the graph to be considered. Also, from the perspective of each node, the network can be partitioned into four parts (quadrants), such that each quadrant can be processed in parallel. In addition, we have the added benefit that the graph is an acyclic directed graph with non-negative edge weights.

The problem of finding shortest paths in a graph is well documented with standard solutions such as Dijkstra's algorithm [20] which is a graph search algorithm which produces a shortest path tree for a given graph with non-negative edge costs. For a given source node in the graph, the algorithm finds the minimum weight path to every other node in the graph and is widely used in network routing protocols like IS-IS [21]. As we need to implement route computation in hardware, however, we take advantage of the constraints described above to come up with a light-weight, hardware implementable algorithm. Starting from the source node, which is assigned a cost of zero, our algorithm visits sets of nodes, in order of their distance from the source, and checks if it can find a path with a cost lower than the cost assigned to that node. In the next iteration, it picks another set of unvisited nodes and repeats these steps. A node is marked as visited once we have determined the shortest possible path to that node and all its outgoing links are evaluated. From a programming perspective, this algorithm solves the problem of shortest path calculation by the following property:

Property 1. *If X is a node on the shortest path from node S to node D, then it implies that the path from node S to node X is also the shortest path between these two nodes*

We can also state this as a linear optimization problem in the following manner. Suppose for a destination node w, the algorithm wants to compute the shortest path cost *C(v,w)*$\forall v \in V$ which is the set of all nodes. The constraint for this linear program can be stated as the cost of the shortest path from node v to node w is less than or equal to the shortest path from node u plus the cost of the direct link between node u and node v.

$$C(v, w) \leq C(u, w) + W_{u,v} \tag{3.3}$$

where node u is the source node. As an initial condition all the destination nodes w receive *C(u,w)* to be infinity and as the algorithm visits every node it is assigned a realistic and optimal distance value.

2. Congestion Information Scaling

Due to less frequent and high-latency congestion updates, the congestion map view will be less accurate for distant nodes. Furthermore, the likelihood that congestion status would change before a packet gets to the link increases considerably for distant nodes. To address both these issues, we assume that the choice of optimal path should be more influenced by the congestion status of nearby links than far away links. The links are scaled towards the nominal value $gray(G = (black + white)/2)$, where *black* and *white* denote the two extremes on the congestion scale. We achieve this by scaling every link's congestion value with a scaling factor S_i, where i denotes the distance from the source node in hops and w ($w < 1$) is an empirically determined constant, according to Formula 3.4.

$$S_i = 1 - (w * i) \tag{3.4}$$

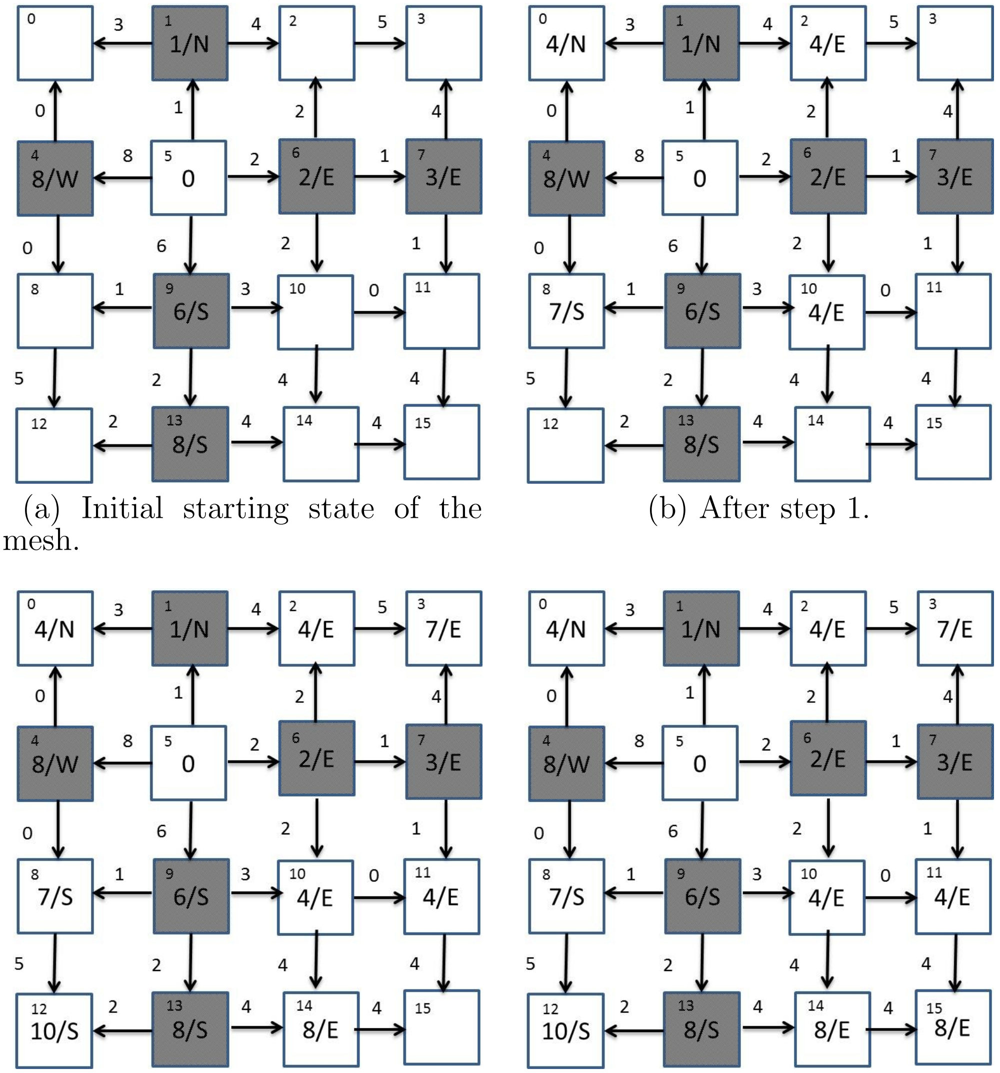
The scaled congestion value C'_i for a link with congestion value C_i is according to the Formula 3.5.

$$C'_i = ((C_i - G) * S_i) + G \tag{3.5}$$

In this manner, the node's own output links are not scaled ($S_i = 1$) and the scaling increases in steps of w as you move away from the node. For all the links with $i \geq 1/w$, the value of S_i is fixed at w as we do not allow negative congestion metrics.

3. Route Computation Example

In this subsection we illustrate the GCA algorithm. Consider the 4x4 mesh in Fig. 6. In the figure, the number in the top left corner of each node represents the node ID. We take node 5 to be the source node, searching for the optimal output port for packets destined to every other node in the network. The congestion values from the congestion map (section III.C), for every link, are shown in the numbers on the links. We assume they have already been scaled as shown. The local node (node 5) does not have any cost associated with it and therefore the cost of reaching itself is assigned zero. For all other links, we assign a value of infinity, we also define an optimal output port with respect to the local node. This output port corresponds



(a) Initial starting state of the mesh.

(b) After step 1.

(c) After step 2.

(d) End of computation.

Fig. 6.: Step by step illustration of the route computation technique for a 4 X 4 2-D mesh.

to the optimal path from the local node to that node. The number inside the node is the shortest path's congestion value in reaching that node while the letter in the node denotes the optimal output port from the local node (N:North, E:East, W:West, S:South). As mentioned earlier, the enforcement of minimal routing in the 2-D mesh allows us to dissect the network into four quadrants from the point of view of the source node. The nodes which are on the same X and Y dimension as the source node are termed *dimension nodes* (shaded in Fig. 6) and every other node lies solely in one quadrant. Traffic within each quadrant is restricted within that quadrant by the minimal routing constraint. In other words, if the source node needs to send the flit to a node in the fourth quadrant, its shortest path will be composed of only the boundary nodes or nodes in the fourth quadrant. This property allows us to perform parallel computation of the optimal paths in each quadrant as they are independent of each other.

We first describe the route computation algorithm at a high level and list the "rules" it enforces in order to keep the implementation simple.

- Every step of the routing algorithm, we update a set of nodes with the congestion value of their shortest path from the source node and the output port which corresponds to this path.

- The congestion value of the shortest path to a node is used in computing the shortest path of the neighboring nodes accessible from it.

- For every node X, we can define at most two *feeder* nodes which refer to the nodes in the network that can send traffic to node X.

- The orthogonal dimension nodes form a special case because they have only one legal output port from the source node and the congestion of their shortest path is simply the sum of the scaled congestion values of the links connecting such a node to the source.

- Due to the straightforward, static nature of computation for the dimension nodes, their output ports are hard coded and their shortest path distances are updated whenever their feeder links are updated.
- For any other node in the network, the route computation consists of a simple add and compare step. As every node can have at most two options, the algorithm compares between the congestion seen by those two paths and picks the lesser one.

Given these rules, the initial starting state of the computation mesh is as shown in Fig. 6a. We now illustrate the route computation technique step by step for the mesh in Fig. 6:

1. In the first step, all the nodes which can be reached from the dimension nodes only are “ready”. Consider the case of node 10 in the Fig. 6a which can be reached through either node 6 or node 9.

   (a) The cost of reaching node 10 through node 6 is the cost of reaching node 6 from the source node plus the congestion of the link from node 6 to node 10:

   $$Cost(6) = 2 + 2 = 4 \tag{3.6}$$

   (b) Similarly, for the path through node 9:

   $$Cost(9) = 6 + 3 = 9 \tag{3.7}$$

   (c) By comparing $Cost(6)$ and $Cost(9)$, we see that the cost to reach node 10 is least through node 6. Therefore, we assign a cost of 4 for reaching node

10 and the output port is carried over from node 6 which is the EAST port. A similar operation is carried out for all the other "ready" nodes and at the end of this step, we have Fig. 6b.

2. For the next step, we have a new set of nodes which are ready to be evaluated using the results of the previous step. Continuing in the same quadrant for our example, we illustrate the same operation for node 14 whose feeder nodes are node 10 and node 13.

   (a) For the path through node 10, the cost to reach node 14 is a sum of the cost to reach node 10 and the congestion on the link between the two nodes:

   $$Cost(10) = 4 + 4 = 8 \tag{3.8}$$

   (b) Similarly, for the path through node 13:

   $$Cost(13) = 8 + 4 = 12 \tag{3.9}$$

   (c) By comparing $Cost(10)$ and $Cost(13)$, we see that the cost to reach node 14 is least through node 10. Therefore, we assign a cost of 8 for reaching node 14 and the output port is carried over from node 10 which is the EAST port. At the end of this step, we have Fig. 6c.

3. At this point, we only have one node left (node 15) whose path still needs to be computed. Repeating the operation described above, we calculate that the minimum cost to reach it is 8 through node 11 and it is assigned an output port of EAST. Fig. 6d shows the final state after the algorithm has computed the optimal output ports for all the nodes in the network.

4. In this manner, the algorithm proceeds outward in the four quadrants till it hits

the extremities of the mesh.

The exact number of steps needed to perform route computation depends on the location of the local node in the mesh. The smallest number of steps are required for the nodes located close to the center of the mesh because they subdivide the rest of the mesh into quadrants and compute the quadrants independently. Nodes which are located on the extreme corners of the mesh require the most steps as the rest of the network lies in a single quadrant from their point of view. For a mesh of N nodes, the number of steps required lies in the range:

- The upper bound is always $(2 * \sqrt{N}) - 3$
- If $\sqrt{N}$ is even, the lower bound is $\lfloor\sqrt{N}/2\rfloor$
- If $\sqrt{N}$ is odd, the lower bound is $\lfloor\sqrt{N}/2\rfloor - 1$

Whenever the node updates one of the links in the map, our route computation module only needs to recompute a subgraph of the network. This is because the traffic flow from the source node is unidirectional and thus the change can potentially affect only the nodes downstream of that link. Also, the property of subcomputation makes the scheme amenable to a hardware implementation by reducing the amount of computation required at every update, as described in Chapter IV.

CHAPTER IV

IMPLEMENTATION

In this chapter we examine the hardware implementation of the GCA router. First we discuss the baseline, locally-aware adaptive router microarchitecture. We then present the GCA router microarchitecture, followed by a discussion of the storage requirements and design variables used in our implementation. This is followed by a discussion of LGCA, a lower-overhead variant of GCA. The standard on-chip router was first described by Peh and Dally [22] which consisted of five output ports, four of which are connect to the mesh while the fifth is connected to the injection source. The pipeline consists of four stages: route computation (RT), VC allocation (VA), switch allocation (XA), and crossbar traversal (XB).

A. Baseline Adaptive Router

The baseline adaptive router architecture used was proposed by Kim et al. [11]. This design reduces the router pipeline to three stages by pre-computing the optimal output port for the incoming flit. The first stage performs the port preselect, look-ahead route computation, speculative VC allocation and XB allocation. The flit traverses the switch in the second stage and traverses the link in the third cycle. Every header flit goes through these three stages but the body flits need pass through the second and third stages only.

The baseline adaptive router is shown in Fig. 7. It uses stored congestion metrics to pick an output port. The congestion information from neighboring nodes at each output port is stored in a structure called the Congestion Value Registers (CVRs), which are used in the port preselect stage. When minimal adaptive routing is used, there are at most two output ports for every quadrant. The preselect module uses the

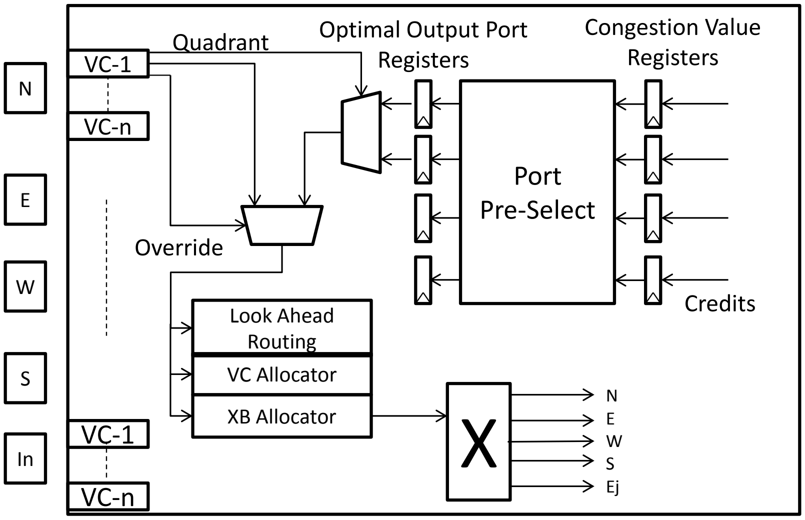


Fig. 7.: Baseline router microarchitecture.

values in the CVRs and does a pairwise comparison to compute the best output port for each quadrant. When a header flit enters the router, its destination quadrant is used to match the precomputed output port for that quadrant while the VC allocation and route computation can proceed in parallel. If the destination node is in the same dimension, then the flit has only one valid output port and the preselect logic is overridden.

B. GCA Router Microarchitecture

In this section, we present the GCA router microarchitecture. The GCA router is shown in Fig. 8, with the additional hardware components not found in the baseline router highlighted in gray. At the skeletal level, the router's microarchitecture is the same as that of the baseline adaptive model. Instead of using CVRs to store local congestion status, however, the router needs a larger structure to store information about the complete network. Despite the larger structure, look-ahead routing ensures

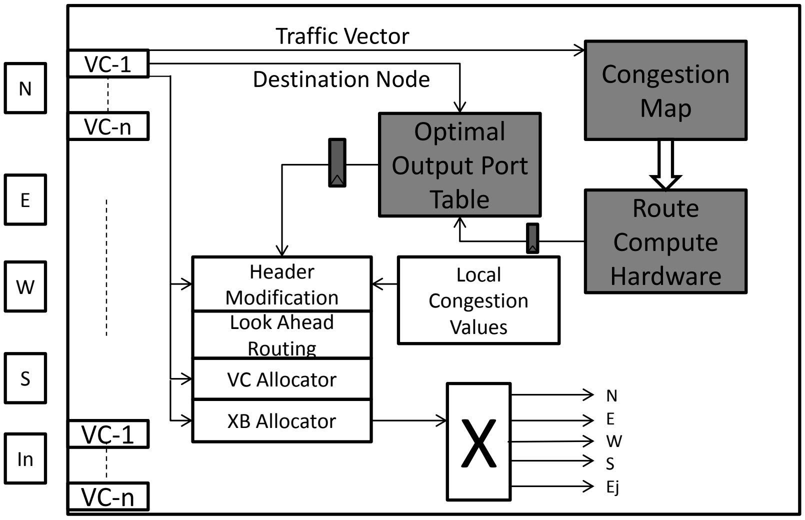


Fig. 8.: GCA router microarchitecture.

that the router pipeline is not adversely effected as we will discuss in the next section. In addition to introducing new storage elements, the algorithm also introduces a route computation module in order to use the congestion map and build the optimal output port case for all the other nodes.

C. Storage Requirement

The first storage element that we describe is the *congestion map* which holds congestion status information about all the other links in the network. We use the number of free virtual channels per output port as a congestion metric for that link and normalize the value on a scale of 0-7, using 3 bits. A heavily congested link is in the *black* state which corresponds to the higher value on our scale (7) while *white* denotes the lower value (0). *Gray* lies in the middle of this scale at 4. This is the major storage overhead that this technique introduces as it requires as many latches as there are downstream links in the network from the given node. For a network with N nodes, this is equal to $3 * 2 * \sqrt{N} * (\sqrt{N} - 1) = 6\sqrt{N}(\sqrt{N} - 1)$ bits.

The second storage structure is an *optimal output port table* which stores the result of the route computation algorithm. The algorithm explores all the legal minimal path options for every node in the network from the current node and picks the path which has the least congestion from the current node's viewpoint. However, as we employ per-hop routing, there is no need to store the entire path for every destination and we can make do by just keeping track of the output port which would send the packet in the direction of the shortest path computed. Due to minimal routing's restriction of at most two valid output ports at any hop, we only need to use a single bit to select between them. The computation of the complete route helps in implementing the look-ahead route computation which enables *pre-routing* in the next hop. To do this, instead of storing the output port decision for the current hop, we store the decision that corresponds to the next hop and embed this information in the packet. We do not save anything for the dimension nodes as they have only one valid output port. Therefore, the overhead due to this structure is $N + 1 - 2\sqrt{N}$ bits for a network with N nodes.

The last storage structure that we wish to mention is the *flag array* which is used to implement the fading mechanism described in section III. C. As we need to fade any entry in the map which has not been modified in the last n cycles, we need to keep track of the last modification time-stamp of every entry. Keeping very accurate timestamps would greatly increase the storage requirements, however, instead we implement an approximation method. We use a flag bit for every entry in the map which denotes whether or not this value was modified in the last n cycle window. At the end of every n cycles the following algorithm is used to implement fading:

```
**for** every entry $i$ in the congestion map ($congestionmap_i$) **do**
    **if** $flag_i = 1$ **then**
        $flag_i \leftarrow 0$
    **else**
        **if** $congestionmap_i > Gray$ **then**
            $congestionmap_i \leftarrow congestionmap_i - x$
        **else if** $congestionmap_i < Gray$ **then**
            $congestionmap_i \leftarrow congestionmap_i + x$
        **end if**
    **end if**
**end for**
```

If the network size is large, then the number of entries in the congestion map would also be large. In that case, performing the fading operation for all entries in one go could be a slow process. We propose to stagger the fading operation by applying the above algorithm for one entry in the map every n/l cycles where l is the number of links in the network. The values of n and x were empirically determined to be 100 cycles and 1 respectively. This flag array adds $2\sqrt{N} * (\sqrt{N} - 1)$ bits.

Table I summarizes the storage overheads required for the GCA technique for a 64-node 2D-Mesh network.

D. Route Computation

The route computation hardware consists of a very simple basic circuit which uses the stored values in the congestion map to compute the best case output port for every node. As described in section III.D, the route computation algorithm performs two additions to determine the possible path costs for reaching a particular node and

Table I.: Storage overhead summary (in bits) for 64 node mesh.

Storage Element	**GCA**	**LGCA**
Congestion Map	336	72
Optimal Output Port Table	49	49
Flag Array	112	24

then compares the results to pick the optimal one. For nodes which are feeders to other nodes in the network, the resultant optimal path cost and optimal output port choice feed thorough to the computation basic block of its downstream nodes. For any node of the mesh which is not a dimension node, the circuit uses adders to add the cost of reaching its feeder nodes to the cost of the link connecting the node to its feeder. As all the evaluated nodes have two feeders, we end up with two cost values which are then compared to pick the lower one and also to pick the corresponding winner's output port to carry forward as the output port for this node. One copy of this basic block is instantiated for every node for which route computation has to be performed, such that two of these blocks feed into the downstream block as its inputs. As the whole circuit is combinatorial, there is no requirement of storage except for the port decisions that are made. If we consider a NxN mesh, we need to do the route computation for $N - (2\sqrt{N} - 1)$ nodes and would require as many basic blocks. For the 4x4 mesh as shown in Fig. 6, there are 9 nodes for which route calculation is to be performed. Therefore, for this network, the route computation hardware implements 9 of the basic blocks as described.

The weight w for the scaling factor as defined in Equation 3.4 was empirically

Table II.: Values of design parameters.

Design Parameter	**Value**
Size of window for fading (n)	100 cycles
Increment/Decrement step (x)	1 unit
Scaling constant (w)	0.25

determined to optimally be 0.20, however for ease of implementation the value of 0.25 was used. For a given source node, the scaling factor for every link in the network is predetermined as the network topology doesn't change. Therefore, whenever the link's congestion value is updated in the congestion map by the node, it is scaled and stored so that the route computation hardware can use this value directly.

Table II summarizes the design variables found to be optimal for a 64-node network employing GCA.

E. Limited GCA (LGCA)

The hardware overhead incurred by GCA is a function of the number of nodes in the mesh. As the mesh increases in size, the storage requirements would also increase and so would the size of the route computation hardware. To address this problem, we propose to impose a constrained GCA for large networks of size NxN by maintaining the congestion information for a limited window of j x j nodes around the current node.

The optimal value of j would be dependent on N and the traffic pattern. For a larger network, the packets would potentially travel a greater number of hops and therefore a larger window would allow better route computation. At the same time, the average number of hops depends on the traffic pattern. If the traffic is highly local, then most packets do not travel much and a small window might suffice. On the other hand, for a traffic pattern with high average hop counts, a larger window would allow us to optimally calculate more of the route of the packet.

The windowing of the network congestion state proves to be an intuitive solution. We know that the link state of nodes closer to the current node are more likely to be accurate as there is a much greater amount of traffic exchanged between them. Also, because of the algorithm's scaling technique we already assign a lower importance to congestion information which is far away from the node. Due to these factors, the window limit becomes a very simple and effective solution. The scaling factor from Equation 3.4 simply uses a nominal value (*gray*) for all links which lie outside the window. In order to route packets to nodes which lie outside the window, the router instead routes them to an *alias* node which lies inside the window. As the packet travels to the destination node, it will encounter a node for which that destination node will be present in its window and from then on the alias node would be the same as the destination node. If at a particular hop, the packets destination node lies outside the window, the alias node is the node inside the window which is closest to the destination node. This distance is measured in terms of the number of hops between the two nodes. There may be multiple nodes inside the window which have the same minimum distance, in which case the node with the lower cost is picked. This selection technique ensures a fairly even distribution of traffic along the periphery of the window.

Table III.: Summary of synthesis results.

Parameter	**Value**
Critical path delay	0.49ns
Number of cells	16
Area overhead (% of total router area)	0.96%

F. Synthesis Experiment

To evaluate the feasibility of this technique, we performed synthesis for a 64-node network with $j = 4$ for an LGCA implementation using Synopsys Design Compiler. We chose to evaluate for a process technology of 45nm.

All the storage and computation elements were written in verilog for implementing GCA in the baseline router. This primarily includes the congestion map, the route computation hardware and the flag array. The route computation hardware implemented as detailed in the previous section for 16 nodes. Everything outside the window is assumed to be of a nominal congestion value and therefore we do not require dedicated route computation hardware for that region. Instead for nodes lying outside the window, we would need a small comparison hardware to select from amongst its choices of an alias node within the window.

We found that the incurred area overhead is negligible (less than 1%). The synthesized route computation unit is capable of running at 2GHz. For designs where the size of the route computation unit exceeds our experimental 16-node design, we can add latches at clock boundaries to stagger the computation over multiple cycles. These latches would be storing the boundary node's optimal cost as well as the optimal output port with respect to the local node. In the following cycle, these values are fed through to the downstream nodes to enable their route computation. We would require latches for all nodes which feed traffic into nodes lying outside the 16-node unit. This also ensures that the routes are first calculated for nearby nodes which turns out to be beneficial. The route for far away destination nodes is always an approximation which might be modified at any of the downstream nodes. Table III summarizes the important synthesis results for the design described above.

CHAPTER V

EVALUATION

A. Methodology

This section presents the performance results of the GCA routing algorithm on different synthetic traffic patterns and a suite of realistic workloads. The results are compared against oblivious routing as well as local and regional adaptive techniques. We also present our results from the network sensitivity studies that we performed.

Table IV.: Simulation parameters

Parameter	**Synthetic traffic**	**SPLASH-2**
Network Size	8x8 2D Mesh	7x7 2D Mesh
Router uarch	Two stage speculative	Same
Per hop latency	3 cycles:2 in router; 1 to cross channel	Same
Virtual channels/port	8	8
Flit buffers/VC	5	5
Traffic workload	Uniform random, Transpose and Bit-complement	SPLASH-2 traces
Duration of simulation	10000 warmup cycles followed by 100000 packets	10 million cycles or end of trace

We employ a C++ based cycle-accurate on-chip network simulator which models the two-stage baseline router microarchitecture as described in section IV.A. All the new structures as described in section IV.B were built into the simulator and the congestion of the links is denoted by the amount of free virtual channels in the downstream node of the link. There are three algorithms that GCA competes against: (1) DOR, an oblivious routing algorithm for 2-D meshes; (2) *Local*, a local adaptive routing technique; (3) *RCA-1D*, a regional congestion aware adaptive routing technique [1]. GCA is also compared against another globally aware routing scheme, DAR [19] under realistic workloads. Table IV details all the parameters that are used to configure the network for our simulations.

B. Realistic Traffic

To demonstrate the effectiveness of GCA on realistic workloads, we run a trace driven simulation of the SPLASH-2 suite of benchmarks [9]. These traffic patterns represent typical scientific workloads. The traces correspond to a 49-node shared memory CMP organized in a 7x7 2D Mesh topology [23].

Fig. 9 shows the average packet latency for the seven benchmarks. The numbers are normalized to the average packet latency of the DOR algorithm. Using the methodology of Gratz et al. [16], we group the SPLASH benchmark traffic into contended and uncontended benchmarks. Uncontended benchmarks *(barnes,ocean, radix, and raytrace)* are the ones where contention is less than 15% of the packet latency. Contended benchmarks *(fft, lu, water-nsquared, and water-spatial)* have significant amounts of contentious traffic and we expected adaptive routing algorithms to be highly effective on these benchmarks. The final bar shows the geometric mean

[1] In all further instances, RCA refers to RCA-1D.

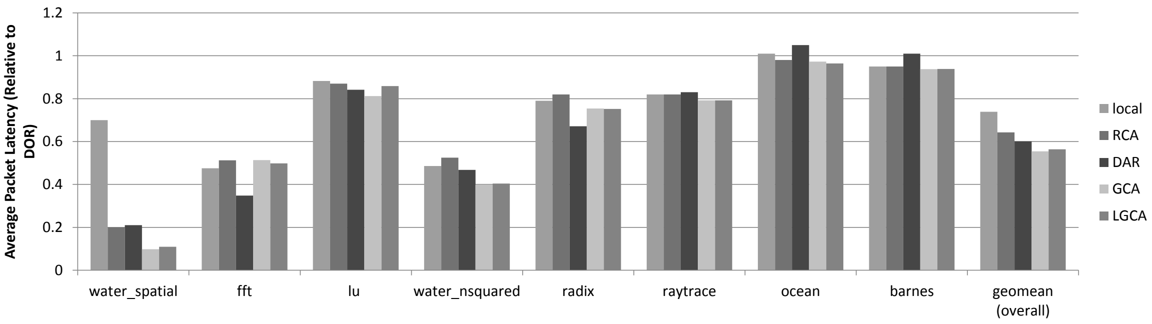


Fig. 9.: SPLASH-2 benchmark results.

of all the evaluated techniques over all benchmarks.

The results show a 44% overall benefit on average packet latency for GCA over DOR, 26% over *local*, 15% over RCA and 8% over DAR. The best case is seen for the benchmark *water-spatial* where GCA performs 51% better than RCA and 53% better than DAR on average packet latency. More importantly, GCA improves latency ∼27% on *water-nsquared* and by 4% on *lu* over RCA. These two contended benchmarks do not see much improvement for RCA over *local*; GCA shows the real benefit that is derived by increasing the awareness that every node has about the congestion status of the network. Comparing the performance of DAR and GCA over the four benchmarks - *fft*, *radix*, *ocean* and *barnes*, we find some interesting details. DAR outperforms GCA on *fft* and *radix* while GCA comfortably beats DAR on the other two. In fact, DAR performs worse than local adaptive on *ocean* and *barnes*. These results highlight the difference between the two globally-aware techniques. *fft* and *radix* both have very stable traffic patterns and hence DAR excels at reducing latency for these workloads, while *radix* and *ocean* have more rapidly changing traffic patterns where DAR's slow route calculation and propagation are not able to keep up. GCA's faster congestion propagation and route computation ensures that it is competitive on all workloads and has a better mean performance.

C. Limited GCA (LGCA)

We also ran simulations for the GCA variant where every node only keeps congestion state information about a j x j mesh around itself instead of the complete network as described in section IV. E. This greatly reduces the amount of storage and computation required per node and is a necessary trade-off for huge mesh based networks where implementing the complete version of GCA might not be feasible. We simulate the SPLASH-2 traces on a 7x7 mesh and synthetic traffic patterns on an 8x8 mesh by keeping information only about a 4x4 mesh around every node. The links outside the window are all scaled to the nominal value which in the case of our experiments is $G = 4$. As expected, the decrease in visibility does cause a degradation in the performance of the algorithm but it is not considerable.

As shown in Fig. 9, the limited visibility variant of GCA outperforms DOR by 43%, *local* by 24% and RCA by 12% on average over all the benchmarks. It improves over DAR by 6% on average. In the best case, it performs 45% better than RCA on the benchmark *water-spatial.* The same benchmark also shows the best performance improvement over *local* and DOR also; an improvement of 84% and 88% respectively.

On the synthetic traffic patterns, we see performance which very closely matches that of the full blown variant though there is a slight degradation as we move to higher injection rates.

D. Multiple Region Performance

As described in chapter II, regionally aware adaptive techniques suffer from aggregating excess information are unable to offer isolation for consolidated workloads. Here we evaluate the performance of GCA in a scenario where there are multiple traffic patterns in different partitions of a network, for example in the case of virtual machines.

Table V.: Trace combinations

Combination	**Trace 1**	**Trace 2**	**Trace 3**	**Trace 4**
Combination A	*lu*	*water-nsquared*	*ocean*	*raytrace*
Combination B	*lu*	*water-nsquared*	*ocean*	*fft*
Combination C	*barnes*	*ocean*	*raytrace*	*radix*

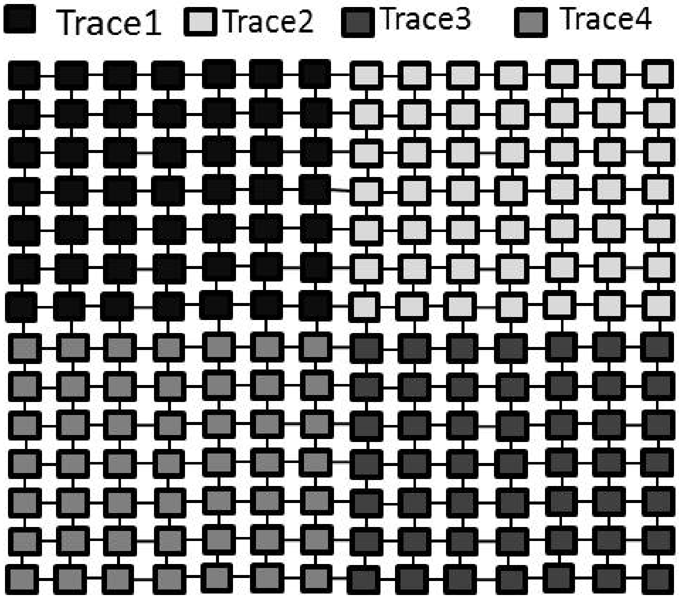
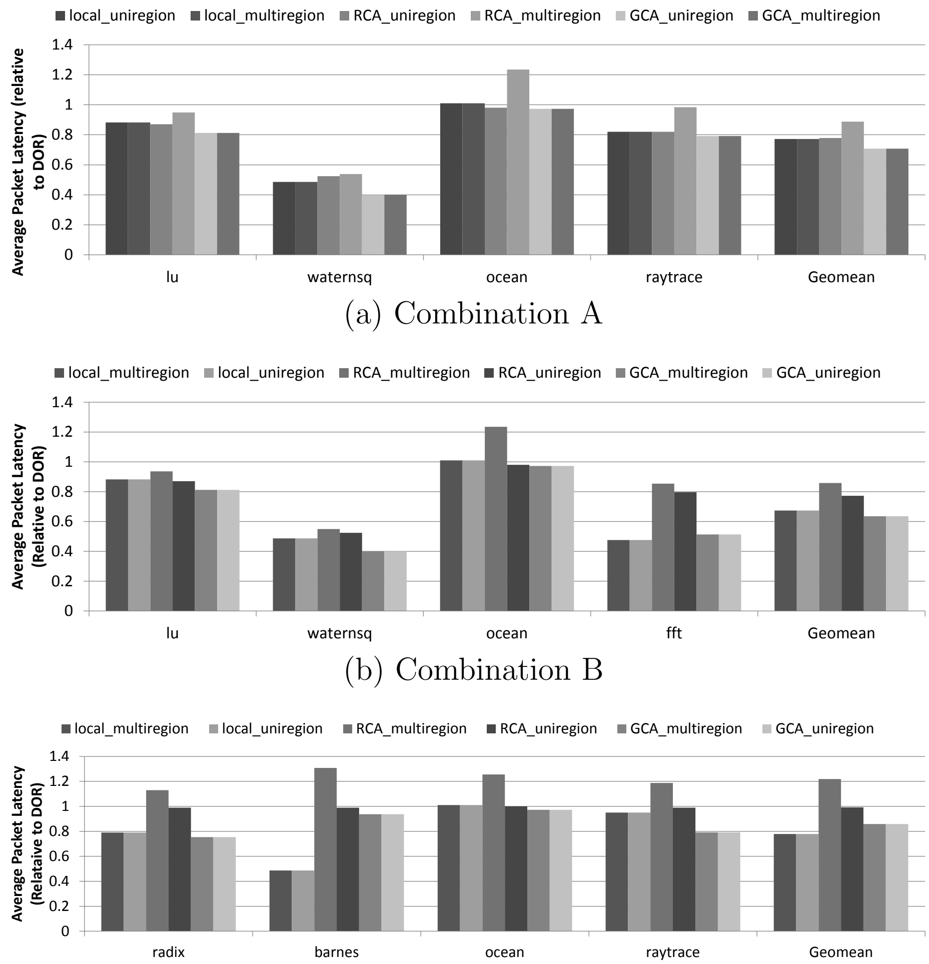


Fig. 10.: Network set-up for multiple region evaluation.

For the purpose of our experiments, we ran simulations of SPLASH traffic on a 14x14 mesh, subdivided into four 7x7 meshes as shown in Fig. 10. Each sub-mesh runs a different trace and the traffic from one sub-mesh does not cross over to another. GCA naturally constrains its route calculation to only the links which are relevant to the packet destination. Hence, it should avoid interference from congestion values in



(a) Combination A

(b) Combination B

(c) Combination C

Fig. 11.: Multiple region evaluation.

the different partitions.

Fig. 11 shows the results from the different combinations of traces, contrasted with the performance of the same benchmarks in isolation. Table V lists the compositions of the different combinations evaluated. The performance of local adaptive and GCA remains unaffected relative the isolated versions of the same benchmark as expected. RCA-1D, however, is adversely affected in this scenario as it would always consider congestion information of other sub-meshes thereby degrading its perfor-

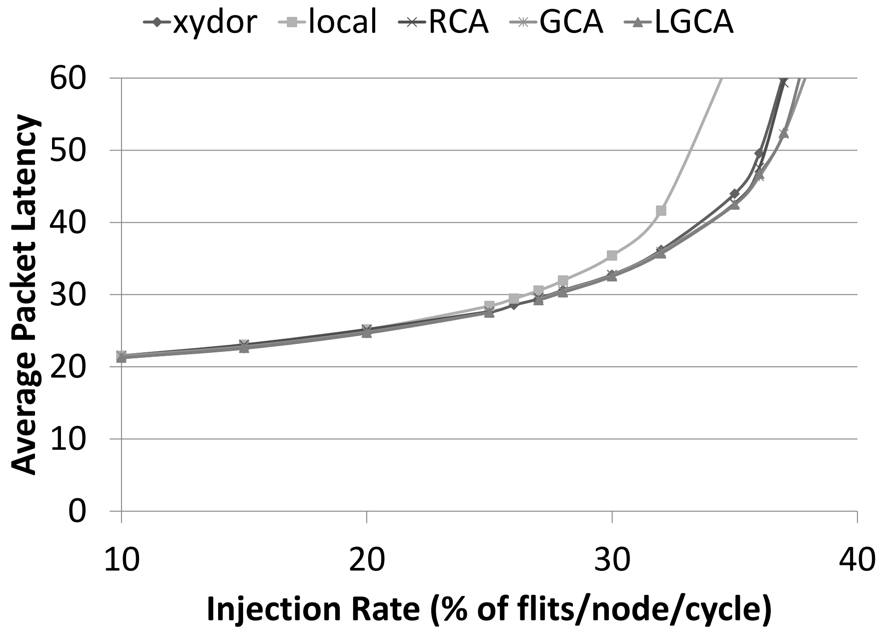


Fig. 12.: GCA's performance for uniform random traffic.

mance. GCA performs better than RCA by 20% on this combination of benchmarks due to the fact that it takes into consideration the locality of traffic in the sub-meshes. It still outperforms local by 9% due to the fact that it is globally adaptive in that sub-mesh.

E. Synthetic Traffic

We evaluate the adaptive routing techniques under three different synthetic traffic patterns: random, bit-complement and transpose. These workloads are characteristic of being nominal, adversarial and friendly to adaptive routing and are injected using a uniform random process. The load latency graphs for these traffic patterns are

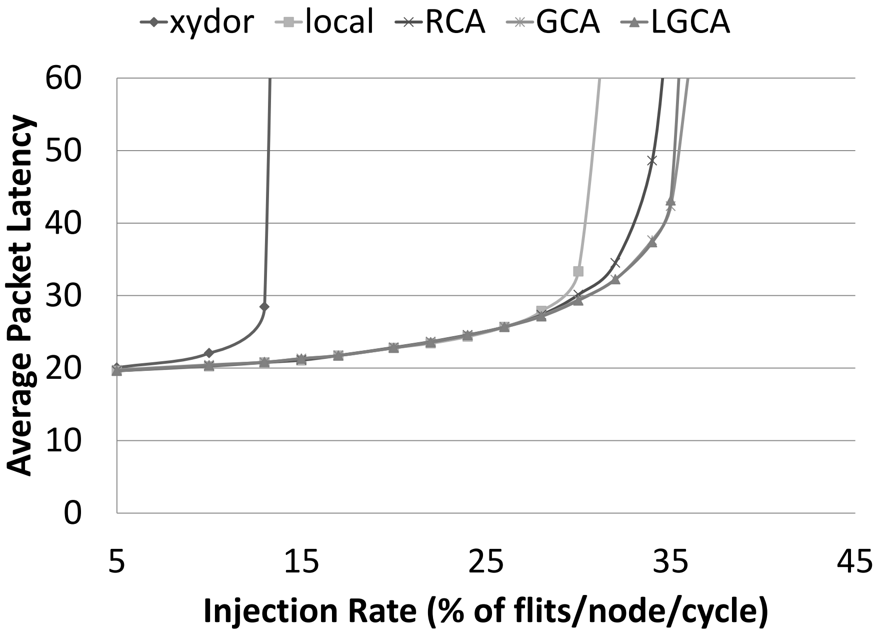


Fig. 13.: GCA's performance for transpose traffic.

presented in Figs. 12, 13, 14 and we consider saturation bandwidth to be three times of zero load latency.

For the transpose traffic pattern, DOR causes load imbalances which are greatly corrected by introducing adaptive routing. We see *local* shows a huge improvement in the saturation bandwidth and it further improves as the adaptive routing algorithm's awareness increases. GCA is able to achieve a 5% improvement in throughput over RCA without sacrificing latency.

In uniform random traffic, adaptivity does improve performance over oblivious routing but the amount of improvement that can be exploited by increasing awareness is limited and that's why there is not much of a difference between *local*, RCA or GCA. However, GCA still offers the best throughput without sacrificing latency. Bit-

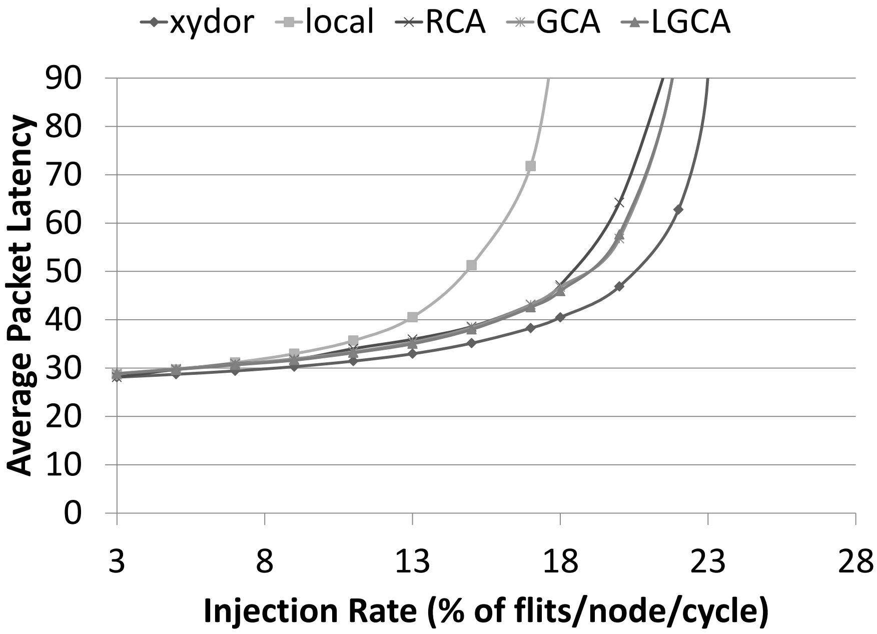


Fig. 14.: GCA's performance for bit-complement traffic

complement traffic is adversarial for adaptive routing because it is ideally balanced and thus DOR performs best on this traffic pattern. RCA is able to significantly improve on *local* and GCA does marginally better.

F. Sensitivity to Network Design Point

In this section, we evaluated the GCA algorithm's sensitivity to various parameters of both the network as well as those of the algorithm itself. We choose the transpose traffic pattern as it shows a clear distinction in performance for GCA and compare it against *local*.

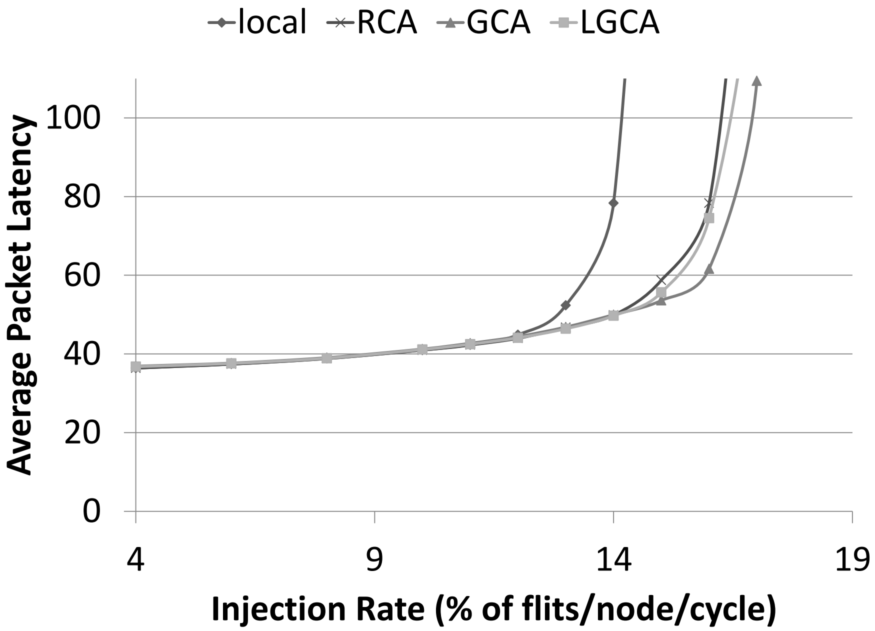
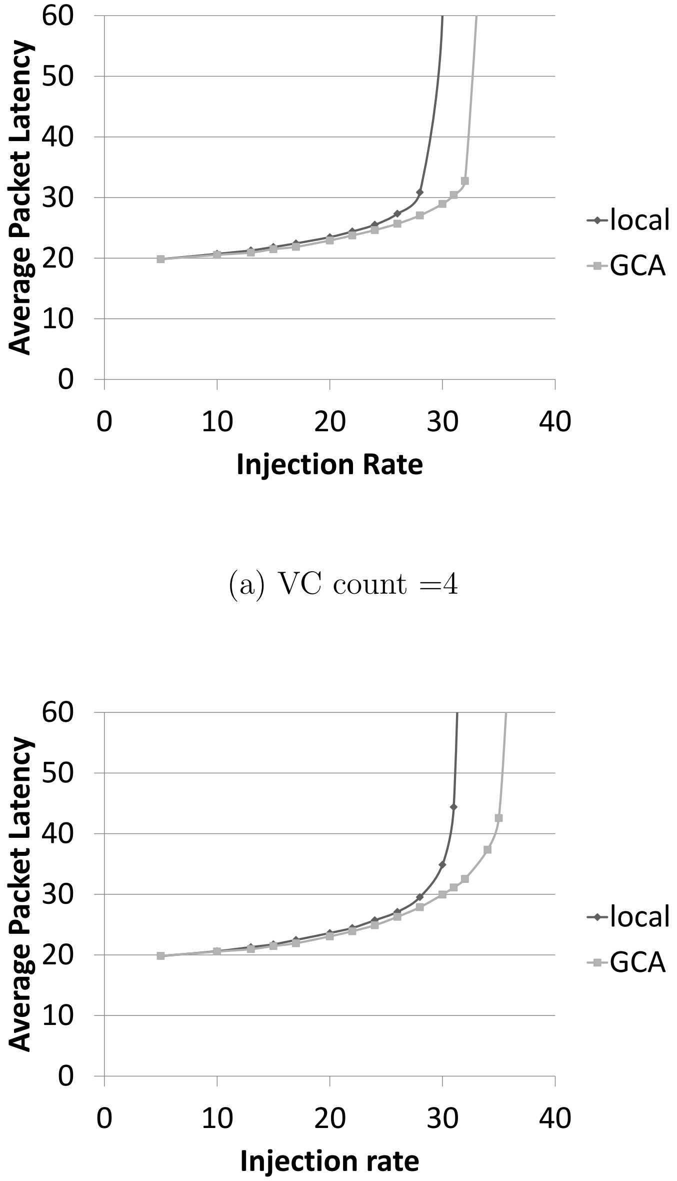


Fig. 15.: Load latency graph for transpose traffic on a 16x16 2-D mesh.

1. Network Dimension

As the sizes of the mesh may vary greatly, we also perform simulations to investigate the variation in performance of GCA based on the network's size. We evaluate transpose traffic on a 16x16 mesh and compare it with *local*. As the size of the network increases, locally adaptive algorithms base their decisions on a much smaller subset of links thereby making even more sub-optimal routing decisions as they would in a medium sized network. In such a scenario, global awareness comes to the rescue by providing a complete view of the network and as we see in Fig. 15, GCA outperforms local by 21% in a 16x16 mesh as compared to 5% in a 8x8 mesh.



(a) VC count =4

(b) VC count =6

Fig. 16.: Load latency graphs for transpose traffic with varying virtual channel count.

2. Number of VCs

Fig. 16 shows the load-latency graphs comparing the performance of GCA over *local* for different numbers of virtual channels per port of the router. The simulations were run on the transpose traffic patterns as it shows the maximum benefit for GCA over *local* and gives a clear view of the effect of varying the virtual channel count. The count of virtual channels affects the resolution at which the congestion information is presented to the route computation algorithm. In such a case, the algorithm is presented with a much coarser view of the network thereby diminishing its performance. However, we still see that GCA outperforms *local* over the counts of 4,6 and 8 virtual channels.

G. Design Space Exploration

This section presents results obtained from changing the parameters of the GCA and LGCA algorithm.

1. Scaling Formula

We present here the effect of variation of the parameter G in Formula 3.5. Apart from the default value of 4 that we use for G, we consider the effect of applying $G = 0$. This value scales the congestion values towards 0 as we go further away from the source node. On the other hand, the default value scales distant links to the nominal value. Fig. 17 and Fig. 18 show the results for GCA and LGCA respectively. In both cases, we see only a slight difference in the overall performance of both the techniques.

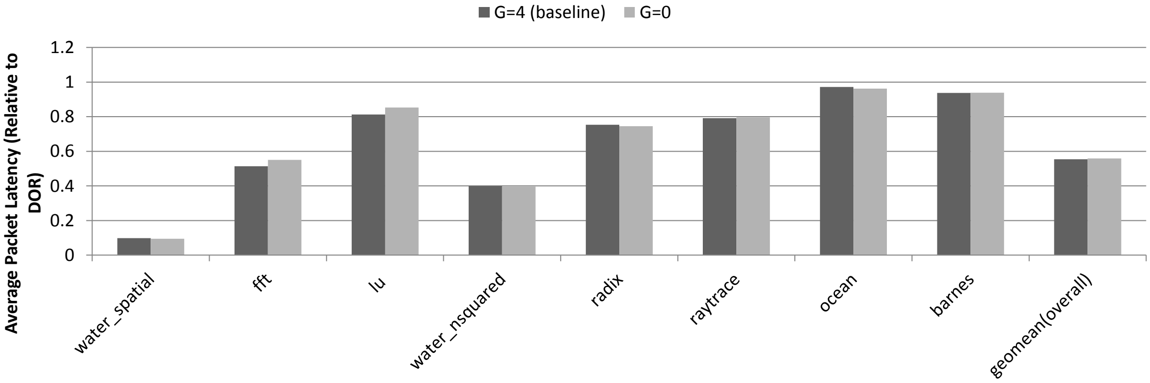


Fig. 17.: Variation of parameter G for GCA.

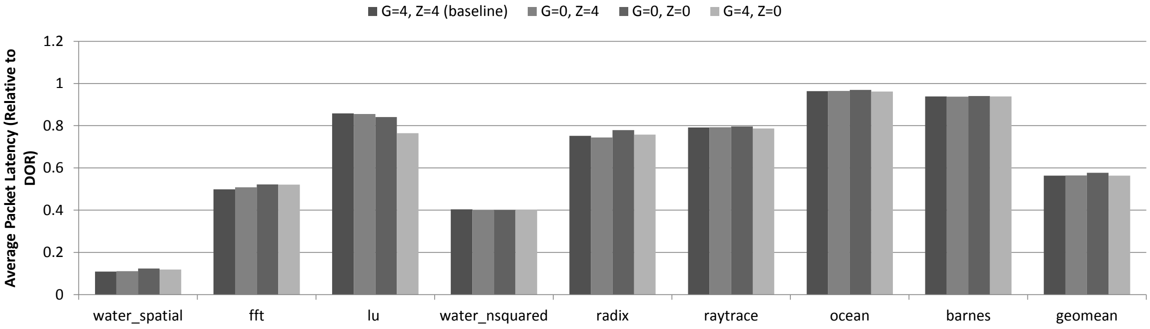


Fig. 18.: Variation of parameters G and Z for LGCA.

2. Links Outside LGCA Window

LGCA's implementation only stores link congestion information for a window around the source node and assumes a value Z for the links which lie outside this window. We evaluate the performance of a 4x4 window LGCA on a 8x8 mesh for two values of Z, ($Z = 0$ and $Z = 4$) and present the results in Fig. 18. The performance for $Z = 4$ is better as it is consistent with the other design principles of assuming a nominal value for the links about which we do not have any information. We also present results for combinations of Z and G.

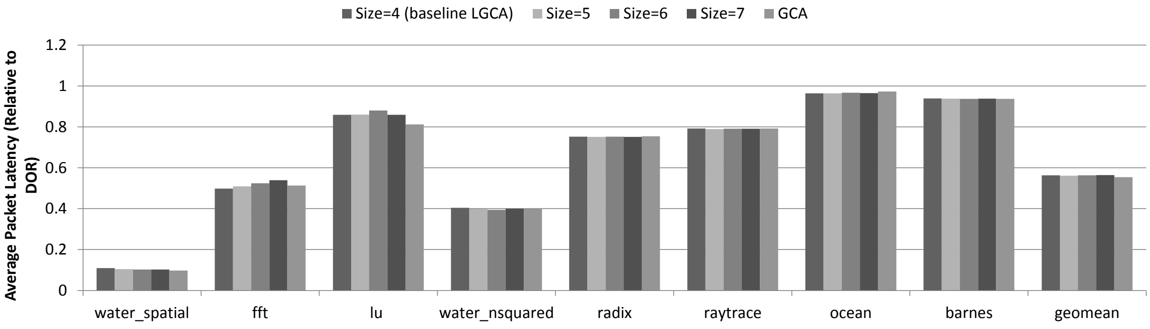


Fig. 19.: Variation of window size (Size) for LGCA.

3. LGCA Window Size

Fig. 19 presents results obtained from varying the window size (Size) of the LGCA implementation on a 8x8 mesh. The default implementation refers to a 4x4 mesh and we explore $Size = 5, 6, 7$. The obtained results tell us that 4x4 is the optimal option when we consider the implementation constraints.

CHAPTER VI

CONCLUSION

In this thesis we present a novel routing adaptive routing technique for on-chip networks which aims to make routing decisions based on global knowledge of network state, Global Congestion Awareness(GCA). GCA is a light-weight, low-complexity adaptive routing algorithm which makes per-hop routing decisions based upon awareness of the congestion of links throughout the network. The route computation technique evaluates all permissible minimal paths from the current node to the destination node and then picks the optimal path to route the traffic. It differs from other globally aware routing schemes in that it utilizes the existent packets within the network to convey ("piggyback") congestion information instead of requiring a sideband network dedicated to congestion status propagation. This makes it a more scalable solution than other existing techniques. Our experiments show that GCA consistently performs better than Regional Congestion Awareness (RCA-1D) on a variety of workloads. On SPLASH-2 traffic, GCA is 51% better in latency over RCA in the best case and 15% better on average. It also betters a competing globally aware routing algorithm, DAR, by 8% on average on realistic workloads.

REFERENCES

[1] D. Edenfeld, A.B. Kahng, M. Rodgers, and Y. Zorian, "2003 technology roadmap for semiconductors," *Computer*, vol. 37, no. 1, pp. 47 – 56, January 2004.

[2] K. Skaugen, "Petascale to Exascale: Extending Intel's HPC commitment," http://download.intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf.

[3] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson, "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2010, pp. 108 –109.

[4] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70 –78, 2002.

[5] W.J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of 38th Design Automation Conference*, 2001, pp. 684 – 689.

[6] C. Grecu, P.P. Pande, A. Ivanov, and R. Saleh, "Structured interconnect architecture: a solution for the non-scalability of bus-based SoCs," in *Proceedings of the 14th ACM Great Lakes Symposium on Very Large Scale Integration (VLSI)*, 2004, pp. 192–195.

[7] J. D. Meindl, J. A. Davis, P. Zarkesh-Ha, C. S. Patel, K. P. Martin, and P. A. Kohl, "Interconnect opportunities for gigascale integration," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 245 –263, 2002.

[8] S. Kumar, A. Jantsch, J.P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proceedings of the 1st IEEE Computer Society Annual Symposium on Very Large Scale Integration (VLSI)*, 2002, pp. 105 –112.

[9] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 24–36.

[10] W.J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466 –475, 1993.

[11] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C.R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in *Proceedings of the 42nd Annual Design Automation Conference*, 2005, pp. 559–564.

[12] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles, "GOAL: a load-balanced adaptive routing algorithm for torus networks," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003, pp. 194–205.

[13] A. Singh, W.J. Dally, B. Towles, and A.K. Gupta, "Globally adaptive load-balanced routing on tori," *Computer Architecture Letters*, vol. 3, no. 1, pp. 2, 2004.

[14] J. Hu and R. Marculescu, “DyAD: Smart routing for networks-on-chip,” in *Proceedings of the 41st Annual Design Automation Conference*, 2004, pp. 260–263.

[15] M. Li, Qing-An Zeng, and Wen-Ben Jone, “DyXY: A proximity congestion-aware deadlock-free dynamic routing method for network on chip,” in *Proceedings of the 43rd Annual Design Automation Conference*, 2006, pp. 849–852.

[16] P. Gratz, B. Grot, and S.W. Keckler, “Regional congestion awareness for load balance in networks-on-chip,” in *IEEE 14th International Symposium on High Performance Computer Architecture*, 2008, pp. 203 –214.

[17] S. Ma, N. Enright Jerger, and Z. Wang, “DBAR: An efficient routing algorithm to support multiple concurrent applications in networks-on-chip,” in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011, pp. 413–424.

[18] R. Manevich, I. Cidon, A. Kolodny, I. Walter, and S. Wimer, “A cost effective centralized adaptive routing for networks-on-chip,” in *14th Euromicro Conference on Digital System Design (DSD)*, 2011, pp. 39 –46.

[19] R.S. Ramanujam and B. Lin, “Destination-based adaptive routing on 2D mesh networks,” in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2010, pp. 19:1–19:12.

[20] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[21] H. Gredler and W. Goralski, *The Complete IS-IS Routing Protocol*, Springer, Dordrecht, 2004.

[22] L.-S. Peh and W.J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001, pp. 255 –266.

[23] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: towards the ideal interconnection fabric," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007, pp. 150–161.

VITA

Name: Mukund Ramakrishna

Address: C/O Dr. Paul Gratz
Computer Engineering Group
Department of Electrical and Computer Engineering
WERC 333C, MS 3259,
Texas A&M University,
College Station, TX 77843-3259.

Email Address: mukundjuly@gmail.com

Education: B.Tech., Electronics & Communication Engineering,
International Institute of Information Technology, Hyderabad,
India, 2010.
M.S., Computer Engineering,
Texas A&M University, 2012.

The typist for this thesis was Mukund Ramakrishna.