Routing Table Minimization for Irregular Mesh NoCs

ABSTRACT

The majority of current Network on Chip (NoC) architectures employ mesh topology and use simple static routing, to reduce power and area. However, regular mesh topology is unrealistic due to variations in module sizes and shapes, and is not suitable for application-specific NoCs. Consequently, simplistic routing techniques such as XY routing are inadequate, raising the need for low cost alternatives which can work in irregular mesh networks. In this paper we present a novel technique for reducing the total hardware cost of routing tables for both source and distributed routing approaches. The proposed technique is based on applying a fixed routing function combined with minimal deviation tables that are used only when the routing decisions for a given destination deviate from the predefined routing function. We apply this methodology to compare three hardware efficient routing methods for irregular mesh topology NoCs. For each method, we develop path selection algorithms that minimize the overall cost of routing tables. Finally, we demonstrate by simulations on random and specific real application network instances a significant cost saving compared to standard solutions, and examine the scaling of cost savings with growing NoC size.[1]

1. INTRODUCTION

Technology projections [1,2] predict that the number of modules in a near future System on Chip (SoC) will grow to several hundreds. NoCs were shown effective in solving the global module interconnect problem [2-16, 21-24] of such SoCs. One of the primary targets in NoC is a low hardware cost in terms of network area and power [2-7, 21-24]. Therefore, most current NoC architectures employ a 2D mesh topology due to the planar nature of VLSI chips, achieving power and area savings [3-8,10-13,23]. In addition, network interface and router logic complexity and power considerations have led to the common use of static, shortest path (SP) routing [3-7,12]. In particular, static routing is cost effective in SoCs where traffic patterns are known a priori and appropriate network topology and capacities can be designed and deployed [3,7]. In other cases, where traffic patterns are not known in advance, NoC routing that performs load balancing based on the dynamic state of the system were proposed [10,13,23,24], requiring more complex routing logic, and possibly network interface that can cope with out-of-order delivery.

In this work we focus on the former design case, i.e. a NoC for a fixed SoC with predefined communication patterns, where appropriate link bandwidth allocation in the network is conducted at the design phase [3,4]. The full design cycle of such a network consists of the following stages. First, identifying the traffic and QoS requirements of the target SoC. Next, customizing the network by an appropriate module placement and applying a *least cost*, static, SP (or lowest energy) routing function. Finally, performing network load balancing by link bandwidth allocation so that multi-class QoS requirements of each communication flow are satisfied while reducing the cost to minimum. First, note that the link capacity assignment is the final stage and is performed *after* the static routing of all source destination pairs are already in place. Therefore, this methodology is in contrast with off-chip networks, where the traffic requirements change in time and the routing mechanisms need to balance the load of this changing traffic over a given topology and link capacities which were designed for legacy or unknown loads.

In a regular mesh it is easy to accomplish SP routing, by employing a simple variation of dimension order routing [17] such as XY [4,5-6,12]. XY is also a "table-less" routing discipline whereby each packet is routed first in an "X" direction and then along the perpendicular dimension. However, practical application–specific NoCs are customized [3-5,14,15,21-24] for better performance and lower cost. As the result the NoC topologies become irregular meshes (Figure 1) because of module shape and size variability and the need to physically separate module internals from the NoC infrastructure.

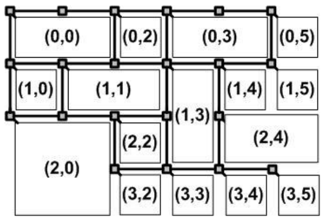

Figure 1. SoC interconnected by irregular mesh NoC. The address of each module corresponds to its top left corner coordinates.

Our definition of an irregular mesh topology is identical to the full mesh including module addresses, except that some routers and links are missing (Figure 1). Packet routing in such NoCs resembles routing in a labyrinth, since some links are missing or may lead to a dead-end. Therefore, a simple XY scheme cannot be employed and different low cost routing techniques need to be applied. In off-chip networks, routing in irregular topologies is typically accomplished using routing tables (RT). The RTs can be located in either the routers (*distributed routing*) or sources (*source routing*). RT size and the corresponding power and area grow with the network size. Moreover, the time required to access each table, which affects NoC performance, depends on its size and thus on the network size. Several recent studies addressed the problem of routing in irregular mesh NoCs[21-24]. Srinivasan et al.[22] have proposed a linear programming based algorithm for routing communication traces such that the total number of routers utilized in the topology is minimized. Schafer et al. [23] proposed to combine adaptive routing based on "The Odd-Even Turn Model" together with placement algorithm. In [24], Palesi et al. proposed a tradeoff between the degree of routing adaptivity and routing table size.

In this work we introduce a simple metric for the estimation of VLSI area and power cost of NoC routing based on the total size (transistor gate-count) of the routing tables [21,24]. Then, we introduce for the first time, hardware-efficient routing techniques that reduce the VLSI cost of static routing in irregular–mesh

[1] This work was partially supported by the Semiconductor Research Corporation (SRC), Intel Corp., and the iSRC consortium.

topology NoCs. The techniques are based on a combination of a fixed routing function (such as "route XY" or "don't turn") and reduced *deviation tables* for both distributed and source routing approaches. Deviation table entries are created only for destinations whose routing decisions differ from the fixed routing function. In most cases, this significantly reduces the area and power costs of full routing tables. Our routing algorithms perform SP path extraction for all source-destination pairs, and minimize the VLSI cost of packet routing. The performance of the network is not degraded by the logic saving actions, since we allow only shortest path routing. In addition, the capacity of each network link may be further tuned to provide the required QoS guarantees at NoC design time. We do not treat in this paper the deadlock avoidance problem related to wormhole based networks, since there are standard ways to solve it by removing circular dependencies using an appropriate virtual-channel ordering [17]. Simulations of random SoC topologies and communication scenarios are used for comparing and estimating cost savings obtained by the different algorithms.

2. TRADITIONAL STATIC ROUTING

Traditional static routing techniques are classified by where routing information is held and where routing decisions are made.

In *distributed routing* (DR) each packet carries the destination address, e.g. the XY coordinates. Each router contains a hard-coded RT or routing function logic whose input is the destination address of the packet and its output is the routing decision, i.e. the output port to which the packet will be forwarded. The routing decision is implemented in each router either by looking up the destination address in the RT or by executing the routing function.

In *source routing* (SR) the pre-computed RTs are stored in the network interface of the system modules. When a source node transmits a packet, it looks up the SR information according to the destination address and includes it in the header of the packet. The SR information includes a routing command for each hop along its path. When the packet arrives at a network router, its routing output port is extracted from its header routing field. Typically, the routing field is then shifted in order to expose the routing command for the next router on its path.

2.1 VLSI Implementation and Cost

As shown above, both DR and SR make extensive use of RTs. RTs can be implemented as tables having an entry for each node in the network. However, this is inefficient, since the set of destinations actually used by each source is likely to be much smaller. This is true because the communication patterns are known a priori and the routing itself is known and restricted.

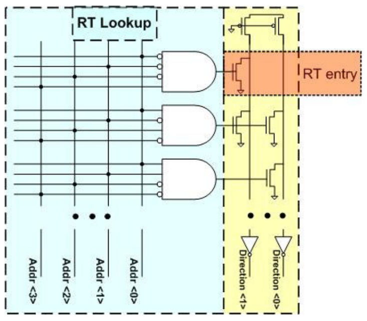

Figure 2. Efficient implementation of a static routing table

More efficient RT implementations employ logic that includes only the necessary table entries for each node (Figure 2). The table comprises routing entries and lookup gates. Our hardware cost model is based on transistor gate-count as an estimate for area and power of the routing logic. The total size of the routing entries of table *i* can be estimated by the sum of the entry sizes ($l_{i,j}$). The look-up logic size can be estimated by the address width, $\log_2(\mathrm{N})$, where N is the total number of modules in the network, multiplied by the number of table entries (n_i). Thus, the total area cost is the sum over all RTs in the network:

$$RT\ \ Area = \sum_{i \in \{\text{NoC tables}\}} \left(n_i \log_2(\mathrm{N}) + \sum_{j \in \{\text{entries of table i}\}} l_{ij} \right) \qquad (1)$$

The dynamic power dissipated in these tables can be also estimated by the size of the tables, since the total capacitance is proportional to the number of entries and the size of the entry. The same is true regarding static leakage power, since it is proportional to the number of leaking devices.

2.2 Prior work on routing tables reduction

Several papers addressed memory complexity of routing mechanisms in off-chip networks. Interval routing [18] reduces RT sizes in large networks by grouping the set of destination addresses that share the same output port into intervals of consecutive integers. Gomez et al.[19] extended interval routing for regular meshes and tori network topologies. Another scheme named "street-sign routing" minimizes SR information [20]. It resembles driving directions: Only the router name of the next turn and the direction of the turn are included in the packet header. All the above schemes can be further incorporated into our schemes but will incur additional gate count costs.

3. HARDWARE-EFFICIENT ROUTING

In this section we present several hardware-efficient routing techniques for irregular topology NoCs. Our DR methods are based on the following observations. Traditional DR techniques are designed to support all possible source-destination pairs, general topologies and path diversity. As was explained above, these features are not required in common custom SoC architectures, but incur excessive VLSI costs. On the other hand, function-based routing (i.e. XY) constrains network topology and path diversity, but results in considerable savings in VLSI costs.

We propose a combination of a low cost fixed routing function and reduced size DR deviation tables. Entries are created in each deviation table only for destinations whose routing decisions differ from the output of the routing function. To that end, we propose two routing techniques, Turns Table (TT) and XY-Deviation Table (XYDT). A third method uses an approach similar to SR. Unlike general SR where the message header carries a routing tag for every node along the traversed path, our scheme termed Source Routing for Deviation Points (SRDP), combines a fixed function (like "don't turn", or "XY") with a short list of tags used only at specific deviation points (DP).

3.1 Turns-Table (TT) Routing

In TT routing, where we use a "don't turn" function, an entry in the deviation table (turn-table) exists if there is a turn in at least one path passing through this router towards the destination (Figure 3, b). When a packet arrives at the router, its destination is looked up in the table. If an entry exists, routing is performed accordingly; otherwise, the packet proceeds without a turn.

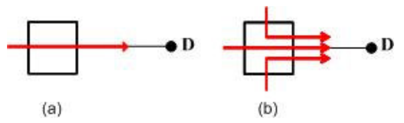

Figure 3. Routing paths toward D: (a) no TT entry (b) one TT entry for D

We develop an algorithm that finds SPs (preferred from power considerations [4]) while taking into consideration the "don't turn" routing function in the routers in order to minimize the overall number of TT entries in the network. Since an entry is created only if there is a turn at a router along some path to the destination, the intuitive solution would be to find SPs that make the least number of turns. However, further minimization can be achieved by exploiting the already existing routing entries in other SPs to the same destination [21].

TT problem definition:

Among all SPs between all sources and destination D, choose a covering set of paths that minimize the total number of entries in the network turns-tables.

TT Routing Algorithm

The algorithm uses the idea of aggregating SPs from different sources whenever possible. Using this heuristic, the algorithm attempts to utilize existing paths (and entries). First, we define an auxiliary Turns-graph (TG) for use by the TT algorithm.

Definition of Turns-Graph(TG): The vertices of the TG are the ports of the original network nodes and its edges are the original network links in four possible directions (+x, -x, +y, -y) and all possible interconnections (turns) among the ports of each network node (Figure 4). The weight of the edge that corresponds to an original network link is a large number K (larger than the maximum number of turns in any SP in the original network). The weights of the edges connecting the ports inside each router are set as follows: if the edge in TG consists a turn via the router, it is set to '1' (dashed lines), otherwise, it is set to '0' (dotted line).

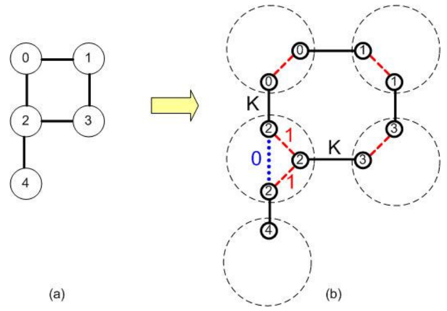

Figure 4. TG example: (a) Original network; (b) Resulting TG

The TT algorithm is formally described in Figure 5. The algorithm is performed for each destination. It uses a greedy approach, iteratively selecting a source node (for paving a path from it to a destination) that adds the minimal number of turns-table entries (heuristic) to the network along its SP to the destination or to an already paved path. The algorithm starts by constructing a TG and initializing node attributes. For each node v, the following attributes are maintained: a pointer to the predecessor node, the distance from the destination in TG, and a Boolean variable which retains information about whether the node has already paved a path to destination D. All nodes except D are initialized as not-reached (lines 2-3). Then the algorithm repeatedly paves SPs from all sources to D (lines 4-12). The process of paving the path starts from relaxing the distances of all non-paved nodes in the graph. The process of relaxing (line 5) improves the distance of each non-paved node to D and updates the predecessor information in each node, until no distance in the network can be improved. At that point, the distance of each non-paved node consists of the distance in hops to D multiplied by K, plus the number of turn-entries that should be inserted into the network tables for this path. Then the non-paved source with the shortest distance among all non-paved sources is selected, and the path is paved from that source to D. The process of paving the path includes marking the nodes on the path as paved (line 9) and resetting its distance from D to only the distance in hops multiplied by K (line 10). The distances of the paved nodes do not include the number of turns to the destination, since any future path that will pass through these nodes will not create any additional routing entries to destination D. The algorithm terminates when all sources have a paved path to D.

```
1)  construct a Turns-Graph TG
2)  $\forall v \in V: \quad Dist(v) = \infty,\ Paved(v) = False, P(v) = nil$
3)  $Paved(D) = True;\ \mathrm{Dist}(D) = 0$
4)  while (!($\forall s \in Sources: Paved(s) = true$))
5)     Relax_not_paved(D,TG)
6)     Pick $S_{min}$ ($\forall s', s_{\min} \in Sources$ ):/*Heuristic*/
          $Dist(s_{\min}) < Dist(s') \cap Paved(s_{\min}) = Paved(s') = false$
7)     Pave_Path($S_{min}$,D)
8)     foreach node v' on Path:
9)        Paved(v') = True;
10)       Distance(v') = hop_num*K;
11)    end foreach
12) end while
```

Figure 5. TT Routing Algorithm - for one destination D

Theorem 3.1:

In each iteration, the TT algorithm selects a non-paved source S and paves a SP from it to D (or to an already paved path to D) which makes the minimal number of turns among all SPs from all non-paved sources to D (or to an already paved path to D)[2] .

Then, for each destination D the routing paths from all source nodes towards D in the original network are extracted by backtracking using the predecessor information in each node. The turns along the paths are found and the TT entries for each turn are inserted in the network nodes along the routing paths. In addition, there is a need to store the direction of the first routing hop for each destination in the source nodes. We use a source default direction technique for minimizing the amount of routing entries in the sources, whereby a default routing direction is stored in the source router for all packets originating from it. A routing entry is inserted into the source router table only for destinations that the first routing step towards them deviates from the default routing direction in the source.

3.2 XY-Deviation Table (XYDT) Routing

In the XYDT method, an entry in the deviation table towards destination D exists only if the next hop from this router deviates from the next hop calculated by the XY routing function. We

[2] Proofs are omitted due to space limitations

assume that packets carry the XY coordinates of the destination. When a packet enters a router its next hop is looked up in the table. If it is found it is routed according to the table. Else, the hardware function calculates the exit port for that packet.

Clearly, the path that makes the minimum number of routing steps that deviate from XY would result in a minimal total number of table entries in the network. In addition, as already mentioned, we consider only SPs. Therefore the XYDT path extraction algorithm solves the following problem.

XYDT Problem definition:

Among all SPs between each S-D pair, select a path that minimizes number of routing steps which deviate from XY routing policy.

XYDT Routing Algorithm:

The algorithm performs a topological sort of the network nodes by their distance from the destination D. For all nodes at same distance from D (h+1) the algorithm assigns an XY-correlated SP routing step towards D if possible, otherwise it assigns any other SP routing step. The algorithm is formally described in Figure 6. All nodes except D are initialized as not-reached. The algorithm starts from D and runs iteratively over the increasing number of hops *h*. In each iteration, the algorithm sets the predecessors to the nodes that were reached in the previous iteration (in *h* hops from D) for later routing path extraction. Then iteratively, the non-reached nodes that can be reached in *h*+1 hops from D are marked as reached and their predecessors would be set in the next iteration. The function set_xy_Predecessor (line 5) is applied to a newly reached node, setting its XY-correlated predecessor on SP to destination if it exists; otherwise it sets any other existing SP predecessor. The algorithm terminates when all nodes are reached.

```
1)  $\forall v \in V: \; Dist(v) = \infty,\; P(v) = nil \;;\; Dist(D) = 0$
2)  $R_h = \{D\},\; R_{h+1} = \{\},\; h = 0;$
3)  while ( ! ( $\forall v \in V : Dist(v) < \infty$ ) )
4)    foreach node   $v_h \in R_h$ :
5)       set_xy_Predecessor( $v_h$ )
6)       foreach v' in 1 hop from $v_h$ :
7)          if $Dist(v') = \infty$ : $R_{h+1} \leftarrow \{v'\}$ , $Dist(v') = h + 1$
8)          end if
9)       end foreach
10)   end foreach
11)    $R_h = R_{h+1},\; R_{h+1} = \{\},\; h = h + 1$
12)end while
```

Figure 6. XYDT routing algorithm – for one destination D

The algorithm in Figure 6 is performed for each destination node D. Then, for each D the routing paths from all source nodes to D are extracted by backtracking using the predecessor information in each node. The XY deviations along the paths are found and the XYDT entries for each deviation are inserted in the network nodes along the routing paths. The algorithm does not insert entries in case of deviation when the following two conditions coexist: (i) the XY-correlated output port is missing and (ii) the routing path continues according to the YX routing function. Consider the examples in Figure 7. Applying XYDT in network (a) results in zero routing entries because proceeding upwards from node Z is the only choice that also matches the YX function (doesn't require an entry). On the other hand, applying XYDT in (b) results in one entry in the table of node Z, since it contradicts an available XY routing option.

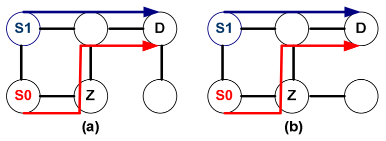

Figure 7. XYDT Examples: (a) No routing table entries (b)One routing table entry in node Z towards destination D

3.3 Source Routing for Deviation-Points (SRDP)

SRDP is an SR method that reduces the size of the full SR headers that are stored in the sources. It combines a fixed routing function (for example XY) with a partial list of SRDP tags which are only used at specific nodes, termed deviation points.

SRDP tag is a list of routing commands for each DP node on the traversed path. The size of the SRDP tag is two bits for a DP node that implements all ports and less in cases when some ports are missing. DPs are network nodes such that a direction of at least one routing path through them deviates from the decision of the fixed routing function (i.e. XY). SRDP algorithm marks these nodes as DPs and any packet (for each destination) that traverses them would have to carry an SRDP routing tag for these nodes. Usually, nodes that become DPs are routers that do not implement all ports (Z in Figure 7 a) or routers that lead to a dead-end when using a fixed routing function, because of a mesh irregularity on the reminder of the path (Z in Figure 7b).

For example, let us apply the SRDP method on the example illustrated in Figure 7b. The example shows a network with two sources S0 and S1 and a destination D. Applying a traditional SR scheme would result in six routing tags because S0 and S1 are both three hops from the destination. Applying the SRDP scheme reduces the amount of SR information to only one tag, since the path from S0 to D can utilize XY function at each hop and the path from S1 to D deviates from XY in only one hop (node Z). Therefore node Z is defined as a DP and requires one SR tag.

Similar to XYDT, when SRDP routing method is used, the path that makes the minimum deviations from XY results in the minimal total number of DPs and consequently minimizes the total amount of SRDP routing headers. Therefore, the SRDP problem is equivalent to the XYDT problem (see Section 3.2).

SRDP Routing Algorithm:

First SRDP applies the XYDT algorithm to all destinations in order to create XY-correlated routing paths between all S-D pairs. Then, all routing paths are analyzed, and nodes that at least one routing step through them deviates from the predefined routing function are marked as DPs. When all DPs are found, SRDP headers are calculated for all routing paths.

4. PERFORMANCE COMPARISON

In this section we compare the traditional DR and SR techniques with the proposed deviation tables routing techniques (TT, XYDT and SRDP) in irregular meshes.

4.1 Evaluation method

In order to evaluate and compare the heuristic techniques presented above, we first apply them to numerous random problem instances and finally check several real application examples. We use random irregular mesh networks in which various modules are randomly designated as hotspots. A random irregular mesh is created by inserting random holes into a regular mesh (removing routers and links). The following assumptions are used regarding the traffic patterns, as an abstraction of typical SoC traffic behavior. Several nodes are defined as hotspots, with a high probability to be destinations for messages from other nodes. Non-hotspot nodes have low-probability of being a destination. We perform a set of simulations on several such random networks, while varying the degree of mesh irregularity (number of holes), number of hotspots and the probability of a node to communicate to a hotspot node. The probability to communicate to a non-hotspot node is kept relatively low (0.1). Locations of holes and hotspots are also randomly generated. The results are averaged over 40 random systems derived with the same parameters. The cost of each routing method is derived by Equation (1). For real application examples we used two video processing applications described by Bertozzi et al. at [15]: *Video Object Plane Decoder* (VOPD) and *MPEG-4 Decoder,* both are mapped on to 12 cores example.

4.2 Numerical Results

Figure 8 shows the significant savings obtained by the proposed hardware-efficient routing methods. It illustrates a 12x12 mesh with a low number (10) of holes and many hotspots (50 out of 134). Comparing the DR methods, XYDT costs less than the original table-based DR by a factor of *34X* (a 97% saving). Among SR methods, SRDP halves the cost of the original SR. The TT also reduces the cost of DR (*3.7X*), but it is less efficient than XYDT. The cost of traditional table-based methods grows considerably with the number of S-D pairs (connection probability growing), while the cost of XYDT remains almost constant as it utilizes XY routing function in most cases, due to the high network regularity.

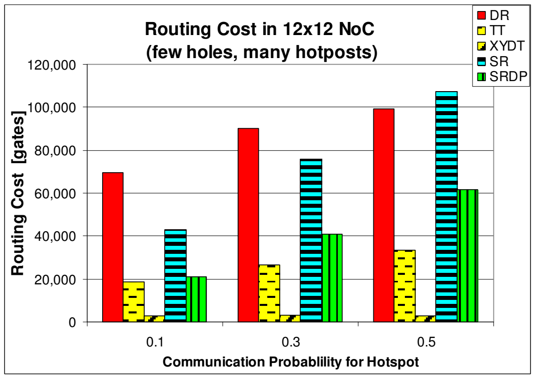

Figure 8. The routing costs as a function of hotspot traffic(few holes, many hotspots): 34X cost reduction by XYDT; 2X by SRDP

Figure 9 illustrates a typical NoC with many holes and few hotspots (50 holes, 10 hotspots). As a result there are fewer source nodes in the network. The costs of DR and SR are smaller, since there are less source-destination pairs. The cost of XYDT grows due to higher irregularity. The savings obtained by XYDT reach 8X (87%) of the original DR. SRDP achieves 2.5X (60%) savings of the original SR.

Figure 10 demonstrates the performance of the proposed algorithms in the two real video processing applications described in [15]. In spite of the fact that the available examples are very small, the obtained savings are very impressive. In the Mpeg4 example for DR case the cost reductions are: 35X by XYDT; 4.3X by TT, and 2.9X by SRDP for SR case. In the VOPD example for DR case the cost reductions are: 40X by XYDT; 4.4X by TT, and 1.9X by SRDP in SR case. Applying our algorithms on future SoC applications examples of hundreds of cores and large communication matrix is expected to yield greater savings.

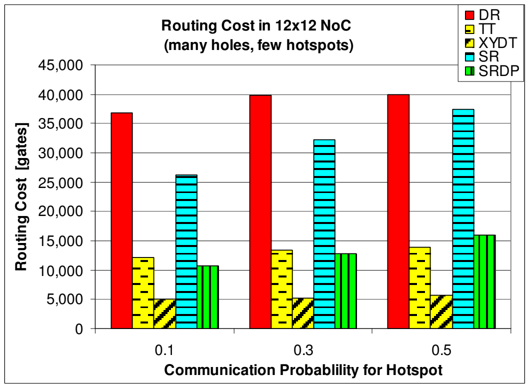

Figure 9. The routing costs as a function of hotspot traffic in typical NoC: 8X cost reduction by XYDT; 2.5X by SRDP

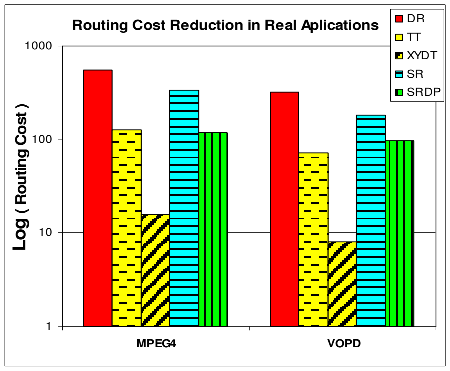

Figure 10. The routing costs on real video processing application examples: up to 40X cost reduction by XYDT; and 2.9X by SRDP

Summarizing all examples, the possible routing table cost reduction obtained by XYDT reach 40X of the original DR cost. SRDP achieves 2.9X cost reduction of the original SR. From both random and real application cases we can see that we can gain significant savings using XYDT in distributed routing and SRDP in source routing. The savings by TT routing scheme are more moderate but still impressive.

4.3 Scalability of Savings in Routing Cost

We study the scaling of the potential cost savings of our methods by simulating typical NoCs with a growing number of nodes (Figure 11). We focus on the two best performing methods (XYDT and SRDP). NoC size grows from 9 to 256 nodes. On average, in each NoC, 40% of the routers are missing and 10% of the nodes are hotspots. The probability of each node to communicate with each hotspot is relatively high (0.5) and the probability to communicate with a non-hotspot node is relatively low (0.1). The triangle marked curve shows the saving of XYDT versus traditional DR and the circle marked curve shows the saving of SRDP versus traditional SR. The graph clearly shows that savings in routing costs grow rapidly (super-linear) with the size of the network. In all points, the relative savings obtained by XYDT and SRDP were around 90% and 60% respectively.

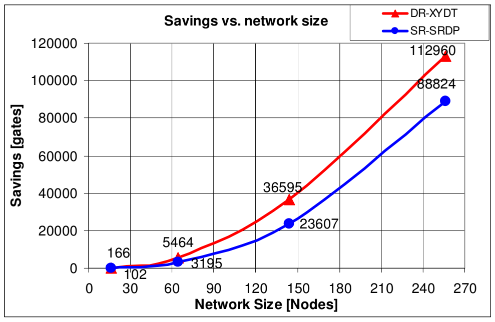

Figure 11. Savings vs. network size in typical NoC

4.4 Scaling of DR vs. SR

Table-based routing suffers from lack of scalability when the network grows (Figure 11). When using SR, scaling is even worse. In SR, in addition to the linear growth of the table with the network size, the amount of the routing information that is stored in each entry grows linearly with the length of the routing path. Therefore SR is efficient only for patterns with a small number of S-D pairs (Figure 12). For a small number of destinations, SR is on-par with DR. As the number of destination grows, the cost of SR grows much faster than the cost of DR. The same is true for the more efficient SRDP and XYDT.

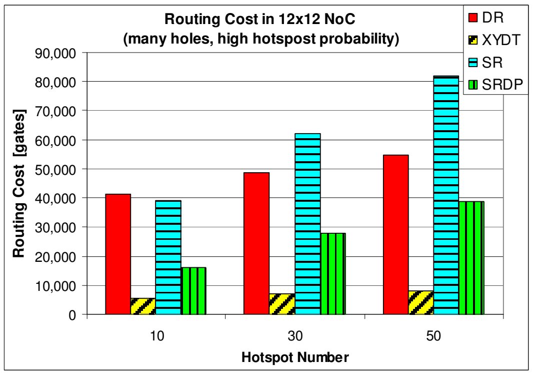

Figure 12. SR scales poorly with growing number of destinations

5. SUMMARY

Novel, hardware-efficient methods for routing in irregular mesh NoCs and routing table size minimization have been presented. The methods are based on static shortest path routing, as typically employed in SoC based NoCs. They overcome the practical issue of mesh irregularities, at minimal cost in terms of size of routing tables. For distributed routing, the preferred method is a fixed routing function along with reduced deviation tables that are used only when the routing decisions deviate from the predefined routing function. For SR, a fixed routing function is combined with a partial list of SRDP tags which are only used at specific nodes, termed deviation points. Path selection algorithms minimize the overall routing cost for each technique. Simulations of random and real application examples have demonstrated a significant cost saving compared to standard DR and SR (*40X* and *2.9X*). We show a super-linear saving growth with the size of the network. In addition, we show scalability advantages of DR over SR as the number of destinations grows.

6. REFERENCES

[1] ITRS, 2003 edition, Design Chapter.

[2] J. Liu et al., “Interconnect intellectual property for Network-on-Chip (NoC),” JSA, Feb. 2004

[3] Z. Guz et al., "Efficient Link Capacity and QoS Design for Wormhole Network-on-Chip", DATE 2006

[4] E. Bolotin, et al., “QNoC: QoS Architecture and Design Process for Networks on Chip”, JSA, Feb 2004

[5] K. Goossens et al. “A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. DATE, 2005.

[6] F.Moraes et al,“HERMES: an Infrastructure for Low Area Overhead Packet-switching NoC,“ VLSI Journal, 2004.

[7] M. Dall'Osso et al., “XPIPES: a Latency Insensitive Parameterized Network-on-Chip Architecture” ICCD, 2003.

[8] M. Millberg et al., “The Nostrum Backbone-A Communication Protocol Stack for Networks on Chip,” VLSI Design Conf., Jan 2004.

[9] D.S. Tortosa and J. Nurmi, “Proteo: A New Approach to Network-on-Chip,” IASTED CSN’02, Spain, 2002.

[10] M. Majer et al., "Packet Routing in Dynamically Changing Networks on Chip", IPDPS 2005.

[11] S. Kumar et al., “A Network on Chip Architecture and Design Methodology,” ISVLSI 2002.

[12] Banerjee N. et al. “A power and performance model for network-on-chip architectures”, DATE 2004.

[13] J. Hu, R. Marculescu, “DyAD - Smart Routing for Networks-on-Chip,” DAC 2004.

[14] J.Henkel, W.Wolf, and S.Chakradhar, "On Chip Networks: A scalable communication-centric embedded system design paradigm", in Procedings, VLSI Design 2004

[15] D. Bertozzi et al., “ NoC synthesis flow for customized domain specific multiprocessor systems-on-chip”. IEEE Trans. on Parallel and Dist.Systems, 16(2):113–129, 2005.

[16] A. Hemani et al., Network on a chip: An architecture for billion transistor era. In IEEE NorChip, 2000.

[17] W. Dally et al, “Deadlock-free message routing in multiprocessor interconnection networks,” IEEE Trans. Comp., C-36(5):547-553, 1987.

[18] J. Van Leeuwen and R.B. Tan, “Interval routing,” The Computer Journal, 30(4):298-307, Aug. 1987.

[19] M.E. Gómez et al., “A Memory-Effective Routing Strategy for Regular Interconnection Networks,” IPDPS 2005.

[20] S. Borkar et al., “iWarp: An Integrated Solution to High-Speed Parallel Computing,” Proc. Supercomputing, 1988

[21] E. Bolotin, at al., "Efficient Routing in Irregular Topology NoCs", CCIT Report #554, EE Dept, Technion, Sep. 2005

[22] K. Srinivasan, at al. "An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks", ICCAD 2005

[23] M. Schafer, et al. "Deadlock-free routing and component placement for irregular mesh-based networks-on-chip",ICCAD 2005.

[24] M.Palesi, S.Kumar, R.Holsmark. A Method for Router Table Compression for Application Specific Routing in Mesh Topology NoC Architectures”, SAMOS VI Workshop, 2006.