Efficient Routing in Irregular Topology NoCs

Evgeny Bolotin, Israel Cidon, Ran Ginosar and Avinoam Kolodny
Electrical Engineering Department,Technion—Israel Institute of Technology,Haifa 32000, Israel

Abstract

Networks on a Chip (NoC) commonly employ an irregular mesh topology because of variations in module sizes and shapes. Consequently, low cost routing techniques such as XY routing are inadequate, raising the need for low cost alternatives. In this paper we first define a hardware resource based cost model for comparing different routing mechanisms. Next, we propose three hardware efficient routing methods for irregular mesh topology NoCs. Our methods combine a fixed routing function (such as XY or "don't turn") and reduced size routing tables based on the known distributed and source routing techniques. For each method, we develop path selection algorithms that minimize the overall cost. Finally, we demonstrate by simulations a significant cost saving compared to standard solutions and examine the scaling of cost savings with the growing NoC size.

1. INTRODUCTION

Modern VLSI systems on Chip (SoCs) comprise many system modules. According to technology projections [1,2] the number of modules will grow to several hundreds in the near future. NoCs were shown to be effective for solving the global interconnect problem among modules [3-10]. NoC power and area saving along with QoS considerations have led to the common use of mesh topology along with static, destination based shortest path (SP) routing, using minimal amount of router logic [4-7]. In a regular mesh it is easy to accomplish shortest path routing, by employing a simple variation of a deadlock free dimension order routing [11] such as X-Y [4-7]. XY is also a "table-less" routing discipline whereby each packet is routed first in an "X" direction and then along the perpendicular dimension.

Practical NoC topologies become irregular meshes (Figure 1) because of modules shape and size variability in VLSI layouts and the need to physically separate between the modules internals and the NoC infrastructure. Nevertheless, to the best of our knowledge no previous studies addressed the problem of efficient static routing in irregular mesh NoCs.

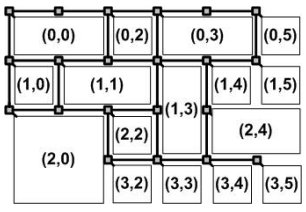

Figure 1. SOC modules interconnected by irregular mesh NoC

Our definition of an irregular mesh topology is that it is identical to the full mesh including the addresses used to identify the various modules, except that that some routers and links are missing (Figure 1). Packet routing in such NoCs resembles routing in a labyrinth, since some links are missing and may lead to a dead-end. Therefore, a simple X-Y scheme cannot be employed and different routing techniques need to be applied. In other networks, routing in irregular topologies is typically accomplished using routing tables (RT). The RTs can be located in routers (distributed routing) or in sources (source routing). RT size and the corresponding power and area costs 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.

We introduce a simple metric for the estimation of VLSI cost (area and power) of NoC routing based on the total size of the routing tables. Then, we develop novel, hardware-efficient routing techniques that reduce the VLSI cost of routing in irregular–mesh topology NoCs. The techniques are based on a combination of a fixed routing function (such as "route XY" or "don't turn") and reduced routing tables for both distributed and source routing approaches. The entries in the reduced routing tables are created only for destinations whose routing decisions differ from the output of the routing function. This way, we significantly reduce the area and power costs of full routing tables in most cases. Our routing algorithms perform routing path extraction for all source-destination pairs, together with minimization of the VLSI cost of the packet routing logic. We do not treat the deadlock problem, since there are standard ways to solve it after all static routes are selected [11]. Random simulations of different topologies and communication scenarios are used for comparing and estimating the VLSI cost savings obtained by different algorithms. We also check the scaling of the VLSI cost savings in NoCs with growing numbers of modules and compare the scalability of distributed and source routing techniques in NoC with growing number of destinations.

2. TRADITIONAL STATIC ROUTING TECHNIQUES

Traditional static routing techniques can be classified according to where routing information is held and where routing decisions are made.

In *distributed routing* (DR) each packet carries the destination address, e.g. the X-Y coordinates of the destination router or a module number. The routing decision can be implemented in each router either by looking up the destination address in a routing table (memory) or by executing a routing function in hardware. Using this method, each network router contains a predefined routing table or routing function logic whose input is the destination address of the packet and its output is the routing decision. When the packet arrives at the input port of the router, its output port is looked up in the table or calculated by the routing logic according to the destination address carried by the packet. The routing information regarding each destination is captured in the

tables (or logic) of each router along the path.

In *source routing* (SR) the pre-computed routing tables are stored in the network interface of the system modules. When a source node transmits a packet, it looks up the source routing information according to the destination address at the SR table and includes it in the header of the packet. Each packet carries in its header the 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. The routing field is then shifted in order to expose the relevant routing command for the next router on its path.

2.1. VLSI Implementation and Cost

As shown above, both distributed and source routing techniques make extensive use of routing tables. DR tables are located at each router, indexed by packet destination address and containing output port values. SR tables are located in each source, indexed by packet destination address and containing sequences of routing commands, one for each hop along the routing path.

Simple RTs are implemented as tables having as many entries as there are nodes in the network. However, this is inefficient, since an all-to-all communication pattern is very unlikely and the actual set of destinations used at each source is a small fraction of the number of modules.

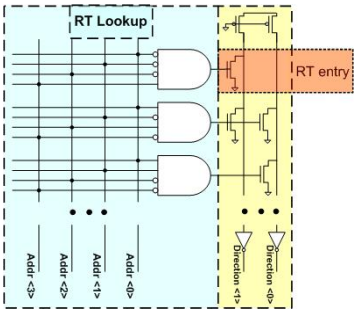

Figure 2. Reduced ROM Implementation of a Static Routing Table

More efficient implementations are the reduced sized ROM (Figure 2), or simple Boolean logic implementing an equivalent routing function. Both schemes only implement the necessary table entries for each node. The reduced ROM implementation is equivalent to a two-level implementation of a routing function by a Programmable Logic Array (PLA).

The total size (in bits) of such a RT for both DR and SR schemes can be estimated by the total size of the entries and the look-up logic. The total size of the entries of table i can be estimated by the sum of the sizes of each entry ($l_{i,j}$). The size of the look-up logic can be estimated by the address size, which is $\log_2(N)$, where N is the total number of modules in the network, multiplied by the number of entries in the table (n_i). Thus, the total area cost can be estimated by summing the costs of all RTs in the network:

$$Cost_{area} = \sum_{i\in\{\text{NoC tables}\}} \left(n_i \log_2(N) + \sum_{j\in\{\text{entries of table i}\}} l_{ij} \right) \quad (1)$$

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. Total power of RTs in the network can thus be estimated by the following formula:

$$Cost_{power} = KCost_{area} \quad (2)$$

where K is a constant.

Several previous works addressed memory complexity of routing mechanisms. Interval routing [12] was proposed as a way to reduce RT size in large networks by grouping the set of destination addresses that use the same output port into intervals of consecutive addresses. Gomez *et al.*[13] extended interval routing for regular meshes and tori network topologies. Interval routing may be used in combination with our scheme. A source routing scheme named "street-sign routing" minimizes source-routing information [14]. It resembles driving directions: Only the router name of the next turn and the direction of the turn are included in the packet header.

3. HARDWARE-EFFICIENT ROUTING METHODS

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. These features, which are not required in common SoC architectures, 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 routing tables. Entries are created in the routing table only for destinations whose routing decisions differ from the output of the routing function. That way, table cost is significantly reduced in most cases. To that end, we propose two routing techniques, *Turns Table* (TT) and *XY-Deviation Table* (XYDT). The third method uses an approach similar to SR. In general SR, the message header carries a routing tag for every node along the traversed path. This requires large storage at the sources. Our *Source Routing for Deviation Points* (SRDP) combines a fixed function (like "don't turn", or "XY") with a reduced list of tags that are used only at specific deviation points (DP).

3.1. Turns-Table (TT) Routing

In TT routing, an entry in the routing table (turn-table) exists if there is a turn in at least one path passing through this router towards the destination (Figure 3).

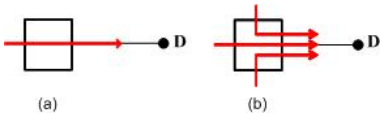

Figure 3. Routing paths toward destination D: (a) no path to D makes a turn (b) an entry in Turns-Table is required because some paths to D must make a turn in this router

When a packet arrives at the router, its destination is looked up in the table. If an entry exists, the routing is performed accordingly; otherwise, the packet proceeds without a turn. This eliminates many entries and reduces the area and power compared to a full routing table.

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.

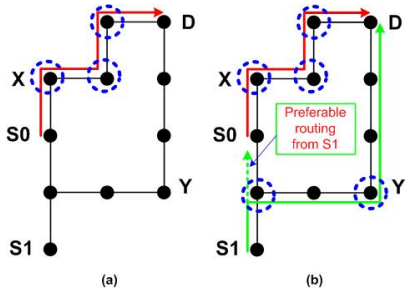

Figure 4. Routing paths towards D: For a path from S1 it is better to prefer a path with more turns via X than over a path via Y

We develop a routing algorithm that finds shortest routing paths (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 routing table 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 most intuitive solution would be to find shortest routing paths that make the least number of turns on their way from source to destination. Additional minimization of the number of TT entries can be achieved by exploiting the already existing routing entries in other routing paths to the same destination. Figure 4 shows example where a routing path which makes more turns results in a smaller number of turns-table entries in the network. There are two traffic sources S0 and S1 and destination D. There is only one possibility for a minimum turns SP from S0 to S1 resulting in a path that passes through node X (Figure 4 a) and creates turns-table entries (dashed circles) in three intermediate routers on its way to D. On the other hand, there are two possible SPs from S1 to D (Figure 4 b). One passes through Y, makes two turns and creates two additional routing entries on its way. Another possible routing path, which is preferable, passes via X. It is also SP and makes more turns than its alternative. However, it creates no additional routing entries in the network, since it utilizes the already existing entries that were created by the previously established path from S0 to D.

TT Routing Algorithm

The algorithm uses the idea of aggregating routing paths from different sources whenever possible. Using this heuristic, the algorithm utilizes the already created paths (and entries) and does not add additional entries over parallel extra routing. First, we define a *Turns-graph* (TG), an auxiliary graph to be used by the TT algorithm.

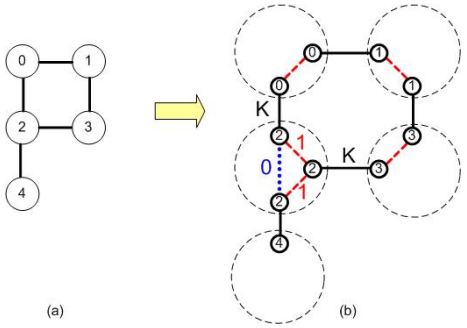

Figure 5. TG example: (a) Original network; (b) Resulting TG

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 5). The weight of the edge that is 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 interconnection edges among 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).

The TT routing algorithm is formally described in Figure 6. The algorithm is performed for each destination node. 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 shortest path to the destination or to an already created (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 of that node from the destination in TG, a Boolean variable which retains information about whether the node has already created (paved) a path to destination D. All network nodes except D are initialized as not-reached (lines 2-3). Then the algorithm repeatedly paves routing paths from all sources to the destination (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 the destination 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 the destination multiplied by N, 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 the destination. The process of paving the path includes marking the nodes on the path as paved (line 9) and resetting its distance from the destination to only the distance in hops multiplied by N (line 10). The distances of the paved nodes do not include the number of turns to the destination, since any future path (and related

distances) 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) ∀v ∈ V :  Dist(v) = ∞, Paved(v) = False, P(v) = nil
3)   Paved(D) = True; Dist(D) = 0
4) while (!(∀s ∈ Sources : Paved(s) = true))
5)   Relax_not_paved(D,TG)
6)   Pick S_min (∀s', s_min ∈ Sources ):/*Heuristic*/
     Dist(s_min) < Dist(s') ∩ 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*N;
11) end foreach
12)end while
```

Figure 6. 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 shortest path from it to D (or to an already paved path to D) which makes the minimal number of turns among all other shortest paths from all other non-paved sources to D (or to an already paved path to D)[1].

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 (output port number) 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 routing table towards destination D exists only if the next hop from this router deviates from the next hop calculated by the X-Y routing function. We assume that packets carry the X-Y 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 shortest routing paths. Therefore the XYDT path extraction algorithm solves the following problem.

[1] All proofs are omitted due to space limitations

XYDT Problem definition:

Among all SPs between each S-D pair, select a path that makes a minimal possible 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. For all nodes at same distance from the destination (h+1) the algorithm assigns an XY-correlated SP routing step towards a destination if possible, otherwise it assigns any other SP routing step. The algorithm is formally described in Figure 7.

All nodes except the destination are initialized as not-reached. The destination node is initialized as 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 destination) for later routing path extraction. Then iteratively, the non-reached nodes that can be reached in h+1 hops from destination are marked as reached in h+1 hops and their predecessors would be set in the next iteration. The function set_xy_Predecessor (line 6) 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)  ∀v ∈ V :  Dist(v) = ∞, P(v) = nil ; Dist(D) = 0
2)  R_h = {D}, R_{h+1} = {}, h = 0;
3)  while (!(∀v ∈ V : Dist(v) < ∞))
4)    foreach node  v_h ∈ R_h :
5)      set_xy_Predecessor(v_h)
6)      foreach v' in 1 hop from v_h :
7)        if Dist(v') = ∞ : R_{h+1} ← {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 7. XYDT routing algorithm – for one destination D

Theorem 3.2:

Among all SPs between each S-D pair, the XYDT algorithm selects a path which makes a minimal possible number of routing steps that deviate from XY routing policy.

The algorithm in Figure 7 is performed for each destination. Then, for each destination D the routing paths from all source nodes to D in the original network 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 regime. Consider the examples in Figure 8. Applying XYDT in network (a)

results in zero routing entries because proceeding upwards from node Z is the only choice that also matches the Y-X regime (doesn't require an entry). On the other hand, applying XYDT in (b) results in one entry in the RT of node Z, since the path contradicts XY.

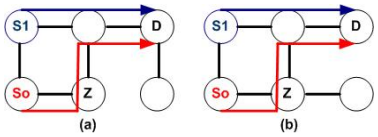

Figure 8. XYDT Examples: (a) No routing table entries (b)One routing table entry in node Z towards destination D

3.3. Source Routing for Turning-Points (SRDP)

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

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 port are missing. DP nodes 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 8 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 8b).

For example, let us apply the SRDP method on the example illustrated in Figure 8b. 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 would reduce 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 the XYDT, when the SRDP routing method is used, the path that makes minimum route deviations from XY results in the minimal total number of DPs and consequently minimizes the total amount of SRDP routing headers. Therefore, the problem of SRDP is equivalent to the problem of XYDT (see Section 3.2).

SRDP Routing Algorithm:

The algorithm is formally described in Figure 9. First SRDP applies the XYDT algorithm to all destinations in order to create XY-correlated routing paths between all S-D pairs (lines 1-6). 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 (line 7). When all DPs are found, SRDP headers are calculated for all routing paths (lines 8-10).

```
1) foreach destination D
2) run XYDT(D)
3) end foreach
4) foreach S and D:
5)   Paths <- Extract_Routing_Path(S,D)
6) end foreach
7) Find_and_Set_DPs(Paths)
9) Calc_SRDP_header(Paths)
```

Figure 9. SRDP routing algorithm for all S-D pairs

4. PERFORMANCE COMPARISON

In this section we compare existing table-based routing techniques (DR and SR) with the proposed routing techniques (TT, XYDT and SRDP) in irregular meshes. We also explore the scalability of the techniques, by plotting the cost savings versus network size. In addition, we demonstrate that DR is preferred over SR as the number of destinations per source grows.

4.1. Evaluation method

A random irregular mesh topology is created by random insertion of holes into a regular mesh (removing routers and links). The following assumptions regarding the traffic pattern (amount of S-D connections) in typical NoCs are used: Several nodes are hotspots, with a very high probability to be a destination to other network nodes, and all others are non-hotspots, with low-probability of being a destination(not all possible communication pairs exist).We perform a set of simulations on several such random networks, while varying the degree of mesh irregularity (number of holes) and the probability of a node to communicate to a hot-spot node. The probability to communicate to a non-hotspot node is kept low at 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).

4.2. Algorithm Comparison in Typical NoCs

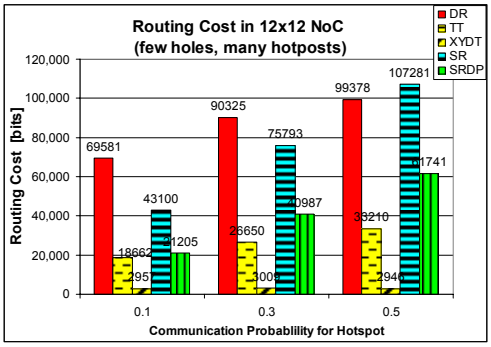

Figure 10. The routing costs as a function of hotspot traffic(few holes, many hotspots): 34X savings by XYDT; 2X by SRDP

Figure 10 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 nodes). Among the DR methods, XYDT cost 34 times less than the original table-based DR (from 99Kbits to 2.9Kbits, a 97% saving).

Among SR methods, SRDP halves the cost of the original SR (from 43Kbits to 21Kbits). The TT method also reduces the cost of DR (3.7 times), but it is less efficient than XYDT. The routing 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, thanks to the regularity of the network (few holes).

Figure 11 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 8 times (87%) of the original DR. SRDP achieves 2.5 times (60%) savings of the original SR.

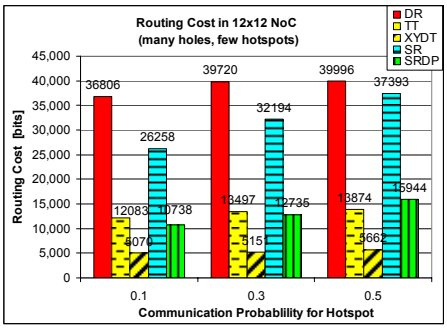

Figure 11. The routing costs as a function of hotspot traffic in typical NoC: 8X savings by XYDT; 2.5X by SRDP

4.3. Scaling of Savings in Routing Cost

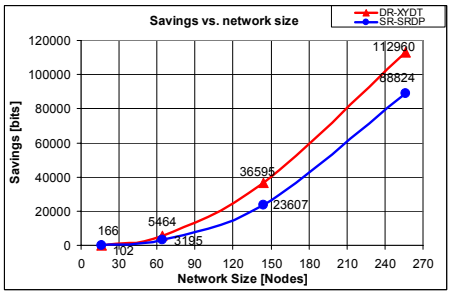

Figure 12. Savings vs. network size (90% of DR and 60% of SR) in typical NoC

We study scaling of cost savings by simulating typical NoCs with a growing number of nodes (Figure 12). NoC size grows from 9 to 256 nodes. About 40% of the routers are missing in each NoC, and about 10% of the nodes are hotspots. The probability of each node to communicate with each hotspot is 0.5 and the probability to communicate with a non-hotspot node is 0.1. The curve with triangles shows the saving of XYDT against traditional DR and the circled curve shows the saving of SRDP against 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.

4.4. Scaling of DR vs. SR

Table-based routing suffers from lack of scalability when the size of the network grows (Figure 12). When using source routing, scaling is even worse. In SR, in addition to the linear growth of the table with the size of the network, the amount of the routing information that is stored in each entry grows linearly with the length of the routing path. Therefore SR is feasible only for communication patterns with a small number of S-D pairs. This is shown clearly in Figure 13. When the number of destinations is low, the cost of SR is on-par with the cost of DR. As the number of destinations grows, the cost of SR grows faster than the cost of DR. The same is true for the more efficient SRDP.

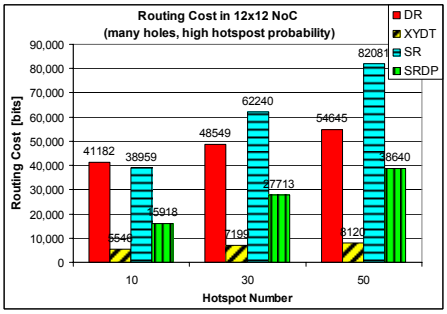

Figure 13. The routing cost as a function of hotspot number. SR scales poorly with growing number of destinations

REFERENCES

1. ITRS, 2003 edition. Design Chapter.
2. J. Liu *et al.*, "Interconnect intellectual property for Network-on-Chip (NoC)," *JSA*, Feb. 2004.
3. A. Andriahantenaina *et al.*, "SPIN: a Scalable, Packet Switched, On-Chip Micro-network," DATE 2003, 70-73.
4. E. Bolotin, et al., "QNoC: QoS Architecture and Design Process for Networks on Chip", JSA,Feb2004
5. W. Dally and B. Towles, "Route packets, not wires," DAC'01, Jun. 2001, pp. 684-689
6. F.Moraes *et al*,"HERMES: an Infrastructure for Low Area Overhead Packet-switching NoC," *VLSI Journal*, 2004.
7. K. Goossens et al. "A design flow for application-specic networks on chip with guaranteed performance to accelerate SOC design and verication." DATE, 2005.
8. M. Dall'Osso *et al.*, "XPIPES: a Latency Insensitive Parameterized Network-on-Chip Architecture For Multi-Processor SoCs," ICCD, 2003.
9. D.S. Tortosa and J. Nurmi, "Proteo: A New Approach to Network-on-Chip," IASTED CSN'02, Spain, 2002.
10. M. Millberg *et al.*, "The Nostrum Backbone-A Communication Protocol Stack for Networks on Chip," VLSI Design Conf., Jan 04.
11. W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comp.*, C-36(5):547-553, 1987.
12. J. Van Leeuwen and R.B. Tan, "Interval routing," *The Computer Journal*, 30(4):298-307, Aug. 1987.
13. M.E. Gómez *et al.*, "A Memory-Effective Routing Strategy for Regular Interconnection Networks," IPDPS 2005.
14. S. Borkar et al., "iWarp: An Integrated Solution to High-Speed Parallel Computing," Proc. Supercomputing, 1988