Efficient Congestion-oriented Custom Network-on-Chip Topology Synthesis

Cristinel Ababei
Department of Electrical and Computer Engineering
North Dakota State University, Fargo ND, USA
Email: cristinel.ababei@ndsu.edu

***Abstract*—We propose a new custom Network-on-Chip (NoC) topology synthesis methodology consisting of floorplanning, routers assignment, and routing paths calculation steps. The proposed heuristic methodology integrates fast algorithms based on the B*-tree representation for floorplanning, on bipartite matching for the routers assignment step, and on multicommodity flow for congestion minimization for the routing paths calculation step. Hence, it is able to explore a large portion of the solution space efficiently. Network performance is estimated using an integrated cycle-accurate simulator. Experimental results demonstrate that custom irregular NoC topologies can achieve latencies comparable to those achieved by 2-layer 3D regular mesh topologies. The multicommodity flow based routing paths calculation is proven to be effective in improving the average latency at high packet injection rates.**

***Keywords*-irregular Network-on-Chip; topology synthesis; multicommodity flow;**

I. Introduction

There are two types of Network-on-Chip (NoC) topologies: *regular* and *custom*. Regular NoC topologies connect cores located in tiles with equal area. Custom or irregular NoCs are designed for applications with cores of different sizes. Because for these applications to assume tiles with equal area is unrealistic, the design flow must include the NoC *topology synthesis* design step. The problem of optimal custom topology synthesis does not have a known theoretical solution [1]. There are several fundamental challenges in the problem of custom NoC topology synthesis. First, due to its irregularity, the topology of a custom NoC affects unpredictably the network performance, power consumption, and area usage. Second, additional design steps such as floorplanning must be included in the design flow. These steps add to the design flexibility and as a result the design solution space is much larger and hence more difficult to explore. Finally, custom topologies require specialized routing algorithms or advanced router architectures to eliminate the risk of deadlock. The goal of this paper is to address these problems by proposing a new efficient topology synthesis methodology. Efficiency is achieved thanks to the short computational runtime of each of the algorithms employed for floorplanning, routers assignment, routing paths calculation, and network simulation.

II. Previous Work and Contribution

Most of the previous works on NoC topology synthesis have used floorplanning to estimate wirelength, area, and power consumption or to generate a *single* starting floorplan. For example, floorplanning information was used during regular NoC topology selection to provide feasible mappings in [2]. Floorplanning was used to estimate power and area during application-specific multi-level NoC synthesis in [3]. A tabu-search based integrated mapping with mixed integer linear program (MILP) based physical planning was used in [4] to find the best topology for regular NoCs from a predefined topology library. A floorplanner was used to estimate link power consumption and to detect timing violations of 2D [5] and 3D [6] application-specific NoCs.

Because the floorplanning solution significantly impacts the custom NoC performance (details will be provided later), limiting the synthesis process to only a single floorplan may lead to sub-optimal NoC topologies. To address this problem, we propose to automatically process multiple floorplans to explore a larger portion of the solution space. To address the complexity of the custom NoC synthesis problem, we propose a *constructive* design methodology similar to [7]–[10]. The proposed methodology combines a sequence of two heuristic steps: routers assignment and routing paths calculation with congestion minimization. The novelty of our approach lies in the utilization of ultra-fast algorithms for both of these steps: bipartite matching for the routers assignment step and multicommodity flow (MCF) ($300\times$ faster than CPLEX [14]) for the routing paths calculation step, respectively. In addition, the floorplanner, based on the efficient B*-tree representation, supports non-slicing floorplans. The efficiency of these algorithms enable practical computational runtimes in exploring topology synthesis starting from multiple floorplans.

III. Custom NoC Topology Synthesis

The block diagram of the proposed constructive NoC topology synthesis methodology is presented in Fig.1.

A. Floorplanning

The objective of the first step is to generate a list of M different floorplans, which will be processed in the next steps. This approach is motivated by the impact that the initial floorplan has on the NoC latency. This is illustrated by the histogram of the latencies of 30 custom NoC solutions synthesized by the proposed methodology starting from 30 different floorplans of testcase *ami25* (see Fig.2.a). Note that solutions vary significantly despite the fact that the area and wirelength of all floorplans have small standard deviations.

We address this problem by exploring a large number of floorplans. The floorplanner is run N times with different

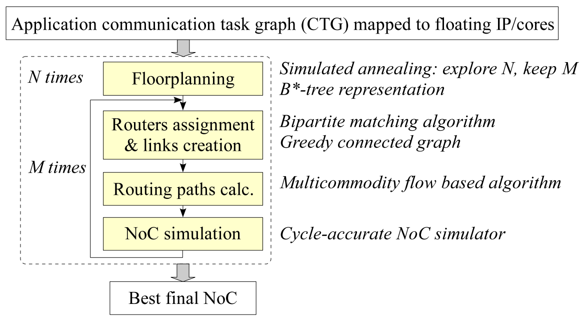

Application communication task graph (CTG) mapped to floating IP/cores
N times
Floorplanning
Simulated annealing: explore N, keep M
B*-tree representation
M times
Routers assignment & links creation
Bipartite matching algorithm
Greedy connected graph
Routing paths calc.
Multicommodity flow based algorithm
NoC simulation
Cycle-accurate NoC simulator
Best final NoC

Figure 1. Overview of the proposed constructive NoC topology synthesis methodology. The floorplanning step is run N times while the next two steps are run $M \leq N$ times.

seeds for the random number generator and the best $M \leq N$ floorplans are recorded. The default values are $N = 100$ and $M = 30$ to keep the computational runtimes practical. The floorplanner is based on the B*-tree representation from [11]. It employs a simulated annealing algorithm, with a cost function that combines area and wirelength (user can specify $\alpha \in [0, 1]$):

$$Cost\ function = \alpha \cdot Area + (1 - \alpha) \cdot WireLength \quad (1)$$

To minimize the wirelength of the source-destination connections with high communication volume (this will translate to reduced number of hops through the network), the edges of the graph utilized by the floorplanning algorithm are weighted by the communication volume (available from the communication task graph (CTG) of each application).

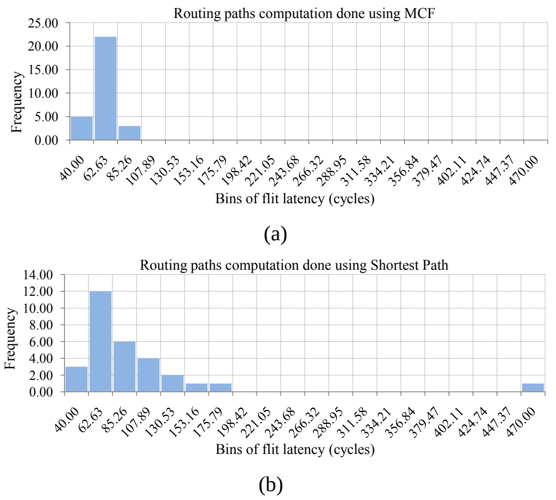


Figure 2. (a) Histogram of the average flit latencies of 30 NoC topologies of testcase *ami25*; median flit latency is 46.77 cycles. (b) Histogram when the routing paths calculation uses the shortest path instead of the proposed MCF based technique; median flit latency is 84.33 cycles.

B. Routers Assignment and Links Construction

In the second step, IP/cores to routers assignment is done for each floorplan from the list of best M floorplans. First, four routers are placed at the corners of each core. If two or more routers overlap or are very close to each other, they are collapsed to only one router. Then, routers assignment is done by employing a bipartite matching algorithm, which utilizes a *bipartite graph* with two sets of nodes: core-nodes representing the application cores and router-nodes representing the routers (see Fig.3). Weighted edges connect each of the core-nodes to the corresponding router-nodes. The weight[1] of an edge between a core C_i and a router R_j is inversely proportional to the number of neighbor routers of R_j within a distance shorter than a threshold $Dist_{th}$ and the accumulated communication volume of all the communications that have C_i as source or destination:

$$w(C_i, R_j) \sim \frac{1}{|Adj_R(R_j, Dist_{th})|}, \frac{1}{Comm_{vol}(C_i)} \quad (2)$$

For example, in Fig.3, the router R_6 has its adjacency list: $Adj_R(R_6, Dist_{th}) = \{R_3, R_4, R_5, R_7, R_8, R_9, R_{10}\}$ while the core c has its accumulated communication volume $Comm_{vol}(c) = 60$. The intuition behind this edge weight expression is that first, we want to assign routers with more neighbors (this will result into well connected network topologies later on during link construction) and second, from among these routers those with higher connectivity should be preferred by the cores with high communication volume (this will increase the chance of high-traffic cores to be assigned to routers that are *well connected*).

Because assigning multiple cores to single routers would require the usage of bigger routers[2] (due to the increased number of ports), we assume that each core is assigned an individual router. The routers assignment problem is solved by the efficient Hungarian algorithm [12]. Note that previous studies formulate this problem as an LP [6] or MILP [8] and solve it using *lp_solve* and *Xpress-MP*, respectively. However, these approaches may become computationally expensive.

Only the routers that are assigned to cores are used to synthesize the NoC topology. This is done by creating

[1]Edge weights will represent the *costs* inside the bipartite matching algorithm.

[2]Even though overall, fewer routers may reduce the area usage, bigger routers are more difficult to insert within the white space of the floorplan – this could be addressed by also considering routers during the floorplanning step. In addition, bigger routers may consume more power due to larger crossbars and arbiters [5], [7].

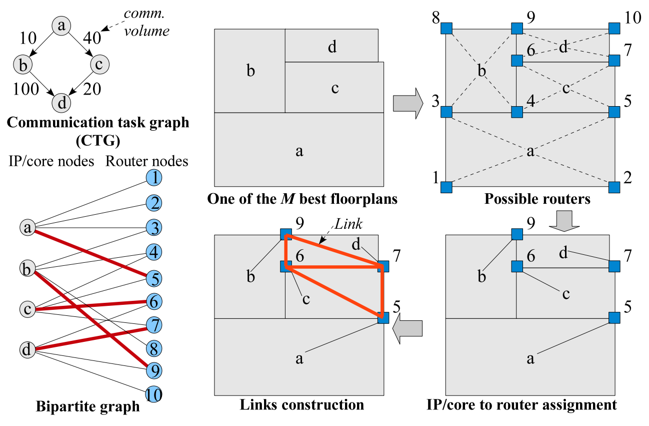


Figure 3. IP/cores to routers assignment and links construction steps.

physical links between any two assigned routers that are within less than the threshold distance $Dist_{th}$ from each other. At the end, we greedily remove some of the created links to restrict the maximum number of ports of each router and hence save area usage (the longest links are removed first). The result of this process, for example for testcase *ami33*, is shown in Fig.6.a.

C. Multicommodity Flow based Routing Paths Calculation

1) MCF problem formulation for latency minimization: Once the custom NoC topology has the communication links constructed, all routing paths must be computed. The information about the routing paths will be stored in routing look-up tables (LUTs) inside each router. To compute the routing paths for all communication demands we formulate this problem as a multicommodity flow (MCF) problem. The communications between application tasks represent the commodities (each commodity has associated a demand given by the communication volume between the tasks), and the main goal is to minimize the communication latency under link capacity constraints. The communication latency is computed based on the wire delays, while capacities are associated with the link bandwidths. Because the MCF algorithm treats simultaneously all the source-destination communication pairs, the communications will be evenly routed through the network. This will lead to an uniformly loaded network and will result in lower congestion and better network performance [13].

To formulate this problem, we build a network graph $G(V, E)$ with the nodes in V associated with the routers that were assigned to cores in the previous step. For each physical link between two routers we create two directed edges in E. Let us assume that, there are k commodities and each commodity has a demand d_i between source node s_i and destination node t_i. Let p_i be the set of all paths p_i between s_i and t_i and $\mathbf{p} = \bigcup_i p_i$ the union of all possible paths for all commodities. Let also $f(p)$ denote the amount of flow sent along path p, for every $p \in \mathbf{p}$. Each edge e has associated a delay D_e and capacity c_e. The MCF problem formulation with the goal of minimizing the total latency as the sum of the delays of flows through all edges is as follows.

$$Min: \quad \sum_{j=1}^{k} \sum_{p \in p_j} \sum_{e \in p} f(p) \cdot D_e \tag{3}$$

$$S.t. \quad \forall 1 \leq j \leq k : \sum_{p \in p_j} f(p) \geq d_j \tag{4}$$

$$\forall e : \sum_{p: e \in p} f(p) \leq c_e \tag{5}$$

$$\forall p : f(p) \geq 0 \tag{6}$$

The constraint in equation (4) ensures that the communication demand for each source-destination pair is satisfied, while the constraint in equation (5) limits the cumulated flow through each edge to less than the edge capacity.

2) Solving the MCF problem using an approximation algorithm: To efficiently solve the MCF problem formulated in the previous section we use an adapted version of the polynomial time approximation algorithm studied in [14]. It is based on the recent advancements in polynomial time approximation schemes (PTAS) [15] and can achieve $(1+\epsilon)$ optimal solutions. The approximation algorithm finds the largest λ and a multicommodity flow solution that routes at least λd_j units of commodity j so that the global latency LT and capacity constraints are satisfied. The *primal* problem formulation is as follows.

$$Max: \quad \lambda \tag{7}$$

$$\forall j: \quad \sum_{p \in p_j} f(p) \geq \lambda \cdot d_j \tag{8}$$

$$\sum_{j=1}^{k} \sum_{p \in p_j} \sum_{e \in p} f(p) \cdot D_e \leq LT \tag{9}$$

$$\forall e: \quad \sum_{p: e \in p} f(p) \leq c_e \tag{10}$$

In order to be able to write the *dual* problem formulation, we introduce the following dual variables: Z_j for each commodity demand constraint from equation (8), Φ_d for the latency constraint from equation (9), and Y_e for each capacity constraint from equation (10). Then, the dual problem formulation is as follows.

$$Min: \quad \sum_{e \in E} c_e \cdot Y_e + LT \cdot \Phi_d \tag{11}$$

$$\forall j, \forall p: \quad \sum_{e \in p} Y_e + \sum_{e \in p} D_e \cdot \Phi_d \geq Z_j \tag{12}$$

$$\sum_{j=1}^{k} d_j \cdot Z_j \geq 1 \tag{13}$$

$$\forall e: \quad Y_e \geq 0 \tag{14}$$

$$\forall j: \quad Z_j \geq 0 \tag{15}$$

The pseudocode of the multicommodity flow solver for total latency minimization is shown in Fig.4. Its idea is to use a binary search technique to find the minimum global latency that still satisfies $\lambda \geq 1$. It uses as a subroutine the algorithm $mcf(G, d, LT)$ whose pseudocode is shown in Fig.5. This subroutine solves the problems defined by equations (7)-(10) and (11)-(15) by iteratively updating the primal λ and dual D values until the gap between them is sufficiently small. In Fig.4, λ_{max} represents the value of λ obtained using the subroutine $mcf(G, d, LT)$ with LT relaxed to infinity. In Fig.5, $l(e)$ defines the *length* associated with each edge e and $dist(j)$ is the shortest path from source to destination of commodity j under the length function $l(e)$.

```
Algorithm 1: Latency minimization MCF algorithm
 1: mcf_solver()
 2: In: graph $G$ with edge capacities $c_e$, demand $d$, threshold $\epsilon$
 3: Out: $(1+\epsilon)$ optimal total latency
 4: Set $\lambda_{max} \leftarrow mcf(G, d, \infty)$
 5: Set lower-bound latency: $LL \leftarrow 0$
 6: Set upper-bound latency: $UL \leftarrow$ total latency under $\lambda_{max}$
 7: while $(UL - LL)/UL > \epsilon$ do
 8:     $L \leftarrow (LL + UL)/2$
 9:     $(LL', UL') \leftarrow mcf(G, d, L)$
10:     $LL \leftarrow LL'; UL \leftarrow UL'$
11: end while
12: return $UL$
```

Figure 4. Pseudocode of the MCF solver.

The solution found using the above approximation MCF algorithm is a *fractional* flow solution, meaning that the communication between a given source-destination pair may be split between multiple paths. Using directly such a fractional flow solution may lead to energy savings [16], but

Algorithm 2: Maximum concurrent flow algorithm

```
1:  mcf(G,d,LT)
2:  In: graph $G$, demand $d$, total latency budget $LT$, threshold $\delta$
3:  Out: $(1-\delta)$ optimal maximum concurrent value $\lambda$
4:  $\forall e, f(e) \leftarrow 0, Y_e \leftarrow \delta/c_e, \Phi_d \leftarrow \delta/LT$
5:  $l(e) \leftarrow Y_e + D_e \cdot \Phi_d$
6:  while gap between primal and dual not small enough do
7:      for each commodity $j$ do
8:          $rd_j \leftarrow d_j$
9:          while $rd_j > 0$ do
10:             // Find shortest path and $f$ units of flow to route:
11:             $P \leftarrow shortest\_path(s_j \rightarrow t_j)$
12:             // Route $f$ units of flow along shortest path:
13:             $f(e) \leftarrow f(e) + f, \forall e \in P$
14:             $\Phi_d^i \leftarrow \Phi_d^{i-1} \cdot [1 + \frac{\delta}{3} \cdot \frac{\sum_{e \in P} f \cdot D_e}{LT}]$
15:             $Y_e^i \leftarrow Y_e^{i-1} \cdot [1 + \frac{\delta}{3} \cdot \frac{f}{c_e}], \forall e \in P$
16:             $l^i(e) \leftarrow l^{i-1}(e) + (Y_e^i - Y_e^{i-1}) + D_e \cdot (\Phi_d^i - \Phi_d^{i-1}), \forall e \in P$
17:             $rd_j \leftarrow rd_j - f$
18:         end while
19:     end for
20:     Compute primal: $\lambda$
21:     Compute dual: $D \leftarrow \frac{\sum_{e \in E} Y_e \cdot c_e + L \cdot \Phi_d}{\sum_{j=1}^{k} d_j \cdot dist(j)}$
22: end while
23: return $\lambda$
```

Figure 5. Pseudocode of $mcf(G, d, LT)$ that finds the maximum concurrent flow such that total latency budget LT is satisfied.

requires complications of the network interface to support packet ordering at destinations [10], [17]. To avoid such complications, we use a simple heuristic to round the fractional flows to *integral* flows such that only one routing path is selected for each source-destination communication. The technique is based on sorting communications in non-increasing order by their communication volume and then rounding their flows along the paths with highest fractions. In our experiments, this approach lead to better results compared to randomized rounding [13].

3) Ensuring deadlock-free routing: Resource ordering is a method to prevent deadlocks in custom NoCs [18]. However, it may over-constrain the MCF based routing paths calculation. In addition, removing deadlocks by breaking all cycles in the channel dependency graph [19] may become computationally expensive and also may over-constrain the routing paths calculation. Hence, to ensure the deadlock-free property, we design each router with a number of virtual channels (VCs) for each port equal to the number of communications (e.g., flows as determined by the MCF algorithm) that pass through. In this way, cyclic dependencies are eliminated. This approach may however use many VCs when the number of communications passing through a given router is high. This problem is in part alleviated by the MCF algorithm, which distributes all flows uniformly across the network.

D. NoC Simulation

In the last step, each of the M NoC topologies is verified using the integrated cycle-accurate simulator. The simulator supports routers with arbitrary number of ports and virtual channels. The following default values are used: packet size of 5 flits with each flit being 64 bits wide, and input buffer size of 5 flits. The router architecture is similar to that studied in [20]. The uniform traffic is generated with packet injection rates that are proportional to the communication volumes of the source-destinations pairs in the CTG of each application. The estimated latency is recorded for each of the floorplans from the list of M floorplans. The topology with the best final latency is selected as the final result.

IV. Experimental Results

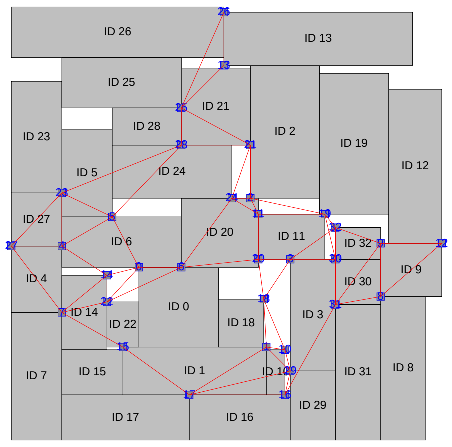
The proposed algorithms were implemented as a computer program in C++. The program can be downloaded from [21]. Simulations were done on a Linux machine running on a 2.8 GHz Intel Quad processor with 2 GB memory. The testcases are shown in Table I. The first two testcases are from [22] and the other testcases are constructed from the classic Microelectronics Center of North Carolina (MCNC) testcases [11]. We scaled the area of each testcase to achieve an average chip size of about $1cm \times 1cm$, which is typical for systems designed using the NoC paradigm reported in the open literature [23], [24]. The last column of Table I reports the computational CPU runtime of the proposed methodology. For example, the CPU runtime is 120 minutes for the largest testcase *ami49*. Note that this includes the total processing time of $N = 100$ floorplanning steps, of $M = 30$ iterations (see Fig.1) of routers assignment and routing paths calculation steps, and of multiple runs of the cycle-accurate simulator for various packet injection rate values. In contrast, the CPU runtime is 20 min for a testcase with 27 cores in [7] (on 2.0 GHz Pentium-M processor) and 8 h for a testcase with 17 cores in [8] (on 950 MHz SPARC processor). The CPU runtime of the routers assignment and routing paths calculation steps only during a single iteration is less than 1 sec and of the same order of magnitude as the heuristic proposed in [9].

Table I
Testcases and their characteristics.

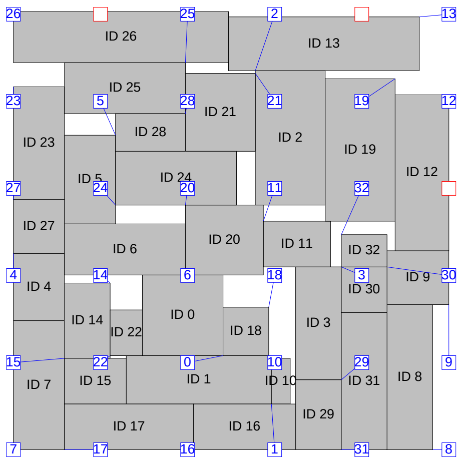
Testcase	Num. of IP/cores	CTG connectivity	Min/max comm. vol.	CPU runtime (minutes)
vopd	12	low	1/500	4.83
mpeg4	14	low	1/942	5.75
ami25	25	medium	1/4	13.32
ami33	33	medium	1/14	22.95
ami49	49	high	1/14	120.08

We cannot compare our results to the studies in [5], [7]–[9] because their implementations are not publicly available. In addition they focused on system-level power consumption and area optimization, while we focus on computational efficiency and network latency. However, to investigate the performance of the custom topologies synthesized by the proposed methodology, we compare them with regular topologies. In the second set of experiments, to investigate the impact of computing the routing paths using the MCF based technique, we compare the average flit latency with that when the routing paths are computed using a shortest path (SP) based technique (approach that is used in previous studies such as [7], [10], [16]).

A. Comparison with Regular Mesh NoC Topologies



(a)

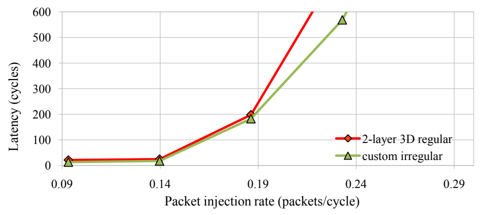


(b)

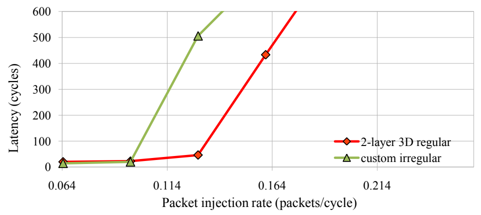
Figure 6. (a) Routers assignment and links construction for testcase *ami33*. (b) Top view for testcase *ami33* of the 2-layer 3D mesh NoC topology: cores reside on the first layer and the mesh network resides on the second layer.

In the first set of experiments, we compare the flit latency of the custom NoC topologies synthesized using the proposed methodology with that of regular mesh NoC topologies. We construct the regular topology assuming a 2-layer 3D NoC architecture [25]. This architecture is constructed by implementing the regular mesh network on one layer and the irregular floorplan on a second layer. Cores are connected to the network routers with extra-links and through silicon vias (TSVs) for vertical connections. The 2-layer 3D regular NoC uses XY routing. An example is shown in Fig.6.b. This architecture achieves better latencies than 2D architectures that use only one layer for both the regular mesh NoC and cores. This is because 2D architectures would either use a large tile area to be able to accommodate the largest core (in such cases a lot of silicon area would be waisted) or expand the cores to make room for routers (which is not realistic because in most practical situations the cores have layouts that cannot be changed).

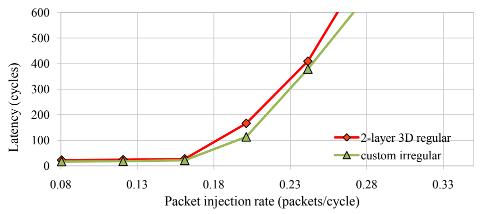
The results of this experiment are shown in Fig.7. Based on our simulations, we make the following observations:



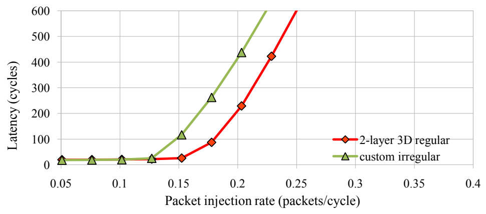
(a)



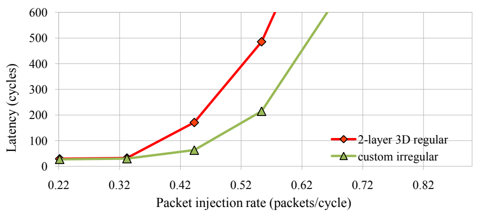
(b)



(c)



(d)



(e)

Figure 7. Latencies achieved by 2-layer 3D regular NoC topologies and proposed irregular NoC topologies: (a) *vopd*, (b) *mpeg4*, (c) *ami25*, (d) *ami33*, and (e) *ami49*.

(i) The extra-links and the vertical TSVs do not affect significantly the average flit latency of the 2-layer 3D mesh NoCs. That is because such delays will only time-shift the injection traces of all source cores while the steady-state average latency will be affected minimally. As a consequence, the 2-layer 3D mesh topologies already offer very good performance.

(ii) The saturation latency achieved by the custom topologies synthesized by the proposed methodology is better than that of 2-layer 3D mesh topologies for testcases *vopd*, *ami25*, and *ami49*.

(iii) We found that a smaller network diameter and a higher radix of the custom topologies are keys to achieving

better network performance. Such topologies can be constructed using small and medium length links rather than only short links, whose usage tends to lead to an increase of the network diameter. A direct consequence of this is that the links construction step has to judiciously use links to synthesize the optimal custom topology.

(iv) Our results are obtained under the assumption that the NoC topologies studied in this experiment are operated as synchronous systems at equal clock frequencies. However, if each topology will be operated at the maximum possible clock frequency – assumed to be given by the longest link between any two connected routers – then the latency of the 2-layer 3D mesh NoC topologies will improve compared to that of the custom topologies. This is because custom topologies must use L-shaped links (implemented on two different metal layers) as opposed to regular meshes that may use straight links in both x,y directions.

B. Comparison between Multicommodity Flow and Shortest Path based Routing Paths Calculation

In the second experiment, we compare the latency results obtained when the routing paths calculation is done using either the proposed MCF based technique or a shortest path based technique. The result for the testcase *ami25* is shown in Fig.8. The other testcases have similar plots, which are not included here due to space limitations.

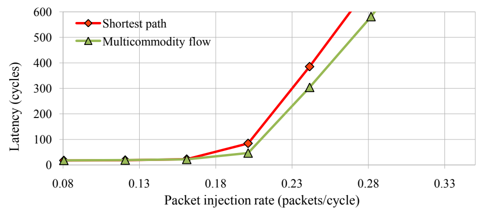


Figure 8. Average flit latency for testcase *ami25*.

Based on our simulations, we make the following observations:

(i) The MCF based technique improves the NoC performance at high packet injection rates. This is because heavy traffic is distributed more uniformly and consequently congestion is minimized.

(ii) At low packet injection rates, the shortest path technique tends to achieve better latencies irrespective of the network topology. This is because the MCF based technique still detours some traffic via longer paths in an attempt to achieve a more uniform distribution of traffic. However, that is not necessary when the network is exercised at low packet injection rates.

(iii) If the custom NoC topology has a spanning-tree like structure, then the two routing paths calculation techniques achieve similar results. Because such a topology does not offer any *path diversity*, the shortest path based technique achieves already the best solution and congestion cannot be alleviated.

V. Conclusions

We proposed an efficient design methodology for custom NoC topology synthesis. The methodology integrates several efficient steps: floorplanning, routers assignment, routing paths calculation, and verification via cycle-accurate simulation. Experimental results demonstrated that custom irregular NoC topologies can achieve latencies comparable to those achieved by 2-layer 3D regular mesh topologies.

References

[1] R. Marculescu, U.Y. Ogras, L.-S. Peh, N.E. Jerger, and Y. Hoskote, “Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives,” *IEEE TCAD*, 2009.

[2] S. Murali and G. De Micheli, “SUNMAP: a tool for automatic topology selection and generation for NoCs,” *ACM DAC*, 2004.

[3] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, “A design methodology for application-specific networks-on-chip,” *ACM TECS*, 2006.

[4] S. Murali, L. Benini, and G. De Micheli, “Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees,” *ACM ASP-DAC*, 2005.

[5] S. Murali et al., “Designing application-specific Networks on Chips with floorplan information,” *ACM ICCAD*, 2006.

[6] S. Murali, C. Seiculescu, L. Benini, and G. De Micheli, “Synthesis of networks on chips for 3D systems on chips,” *ACM ASP-DAC*, 2009.

[7] J. Chan and S. Parameswaran, “NoCOUTs: NoC topology generation with mixed packet-switched and point-to-point networks,” *ACM ASP-DAC*, 2008.

[8] K. Srinivasan, K.S. Chatha, and G. Konjevod, “Linear-programming-based techniques for synthesis of Network-on-Chip architectures,” *IEEE TVLSI*, 2006.

[9] K.S. Chatha, K. Srinivasan, and G. Konjevod, “Automated techniques for synthesis of application-specific Network-on-Chip architectures,” *IEEE TCAD*, 2008.

[10] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, “Synthesis of on-chip interconnection structures: from point-to-point links to networks-on-chip,” *Technical Report UCB/EECS-2006-147*, UC Berkeley, 2006.

[11] T.-C. Chen and Y.-W. Chang, “Modern floorplanning based on B*-trees and fast simulated annealing,” *IEEE TCAD*, 2006.

[12] J. Munkres, “Algorithms for the assignment and transportation problems,” *J. of the Society of Industrial and Applied Mathematics*, 1957.

[13] T. Leighton, S. Rao, and A. Srinivasan, “Multicommodity flow and circuit switching,” *Annual Hawaii Int. Conf. on System Sciences*, 1998.

[14] Y. Hu, Y. Zhu, H. Chen, R. Graham, and C.-K. Cheng, “Communication latency aware low power NoC synthesis,” *ACM DAC*, 2006.

[15] G. Karakostas, “Faster approximation schemes for fractional multicommodity flow problems,” *ACM/SIAM SODA*, 2002.

[16] S. Murali and G. De Micheli, “Bandwidth-constrained mapping of cores onto NoC architectures,” *ACM DATE*, 2004.

[17] S. Murali, D. Atienza, L. Benini, and G. De Micheli, “A method for routing packets across multiple paths in NoCs with in-order delivery and fault-tolerance guarantees,” *VLSI Design Journal*, 2007.

[18] G. De Micheli, L. Benini, Networks on Chip, Morgan Kaufmann, 2006.

[19] J. Duato, S. Yalamanchili, and L. Ni, Interconnection Networks, Morgan Kaufmann, (2002).

[20] L.-S. Peh, and W.J. Dally, “A delay model and speculative architecture for pipelined routers,” *HPCA*, 2001.

[21] HNOC, 2010, [Online]. Available: http://venus.ece.ndsu.nodak.edu/~cris/software.html

[22] E.B. van der Tol and E.G.T. Jaspers, “Mapping of MPEG-4 decoding on a flexible architecture platform,” *SPIE Media Processors*, 2002.

[23] S.R. Vangal et al., “An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS,” *IEEE J. Solid-State Circuits*, 2007.

[24] S. Bell et al., “TILE64 - processor: a 64-Core SoC with mesh interconnect,” *IEEE SSCC*, 2008.

[25] V. de Paulo and C. Ababei, “A framework for 2.5D NoC exploration using homogeneous networks over heterogeneous floorplans,” *IEEE ReConFig*, 2009.