Application-Specific Networks-on-Chips Design

Majid Janidarmian[1], Atena Roshan Fekr[2], Vahhab Samadi Bokharaei[3]

Abstract— **Mapping algorithm, which is an important phase of an NoC design tries to map most frequent and most critical communications in such a way that minimize the physical distance between the source and destination nodes. The objective of this paper is to achieve an application-specific NoC design that minimizes the communication cost and improves the fault tolerant properties. First, a heuristic mapping algorithm that produces a set of different mappings in a reasonable time is presented. Although this mapping does not explore the design space thoroughly, it considers a part of design space, which in general minimizes the communication costs of solutions while yielding optimum communication costs in some cases. Comparison of the communication cost results makes it obvious that final solutions found by our proposed approach outperform the results of other methods, which proposed in literature. Then, the used routing algorithm and the concept, vulnerability index, which is considered as a criterion for estimating the fault-tolerance of mapped application, are presented in details. Lower communication cost leads to an NoC with better metrics such as energy consumption and latency; and reducing the vulnerability index optimizes fault tolerant properties of NoC. In order to yield a mapping which considers trade-offs between these two parameters, a linear function is defined and introduced. It is also observed that more flexibility to prioritize solutions within the design space is possible by adjusting a set of if-then rules in fuzzy logic.**

Keywords: Application-Specific Network on Chip, Mapping, Routing, Fault-Tolerance, Vulnerability Index, Robustness Index

I. INTRODUCTION

A network-on-chip (NoC) is an on-chip communication infrastructure that implements multi-hop and predominantly packet-switched communication. Through pipelined packet transmission. NoCs permit a more efficient utilization of communication resources than traditional on-chip buses. Regular NoC structures reduce VLSI layout complexity compared to custom routed wires [1]. The mapping of real applications on distributed NoC-based architecture is still an open issue [2] which decides which core should be linked to which router. Mapping an application to on-chip network is the first and the most important step in the design flow as it will dominate the overall performance and cost [3]. Efficient routing of messages within the network is essential in order to fully exploit the power of the computing resources and achieve good performance for applications running on them. Routing schemes with the ability to tolerate the faults are important in the massively parallel multiprocessors networks [4]. The Routing algorithms are classified as deterministic or adaptive.

In deterministic routing the path from the source to the destination is completely determine by the source and the destination addresses. In adaptive routing multiple paths from the source to the destination are possible. Thus, generally, it provides packets with a better chance to avoid hot spot or faulty regions in the network as compared to deterministic routing [5]. Although the deterministic routing algorithms are simple and have less complexity in hardware design, they are not able to efficiently consume bandwidth of links. Since reliability and fault-tolerance of network are two important issues in scaling of NoCs, the adaptive algorithms are the recommended solution. These algorithms can tolerate the network failures better than deterministic algorithms by using multiple paths.

Different network links, connecting routers to other routers or modules, may have different bandwidths set during the design process to meet the QoS requirements [6]. The Links exceeding threshold bandwidth in network should be avoided in order to maximize the system performance [8]. Routing and mapping algorithms play important roles to achieve this goal. The main purpose of this article is to present a new mapping model that optimizes the results in accordance to communication costs and fault-tolerance considerations. In this work, we consider NoCs with 2D mesh topologies which offer many desirable properties including better parallelism and scalability, low cross-section bandwidth and fixed degree of nodes [7]. It is worth noting that the approach proposed in this paper is not topology-dependent and could be implemented on any other topologies, either.

This paper is an extension of [9] and the rest of this paper is organized as follows. In section 2 we discuss related research work. Section 3 is composed of five subsections. The proposed mapping algorithm, Citrine, and related formulation and results are presented in 3.1. The used routing algorithm is described in 3.2. Vulnerability index is defined in 3.3 as a criterion to evaluate fault-tolerance. 3.4 introduces a total cost function and experimental results and design space exploration. Finally in 3.5 the fuzzy logic solution and also the results of it are done in this subsection. The conclusion is drawn in the last section. The graphs which used as the case studies in section 3.1 are shown in appendix I.

II. RELATED WORK

Several approaches have been proposed in literature in the context of topological mapping in NoCs. The current researches mostly focus on mapping techniques for NoC platform with two dimension mesh topology. The NMAP method that runs mapping with regard to bandwidth constraints and minimizing communication delay is selected as the criterion in most projects, and is highly efficient as far as communication cost is considered [10]. In [3] a binomial mapping method is introduced. The latter comes along with an optimal algorithm aiming at minimizing total traffic on network, the number of hops, and hardware costs. The Branch-and-Bound algorithm, presented in [11] has been able to map IP cores on tiled-based NoC architecture, and it has tried to minimize total energy consumption, and to overcome bandwidth constraints. The proposed mapping algorithm in [12] is basically a genetic algorithm, which takes the advantages of the chaotic systems by using them instead of

[1] CE Department, Science and Research Branch of Islamic Azad University, Tehran, Iran, jani@srbiau.ac.ir.
[2] ECE Department, McGill University, Montreal, Quebec, Canada, atena.roshanfekr@mail.mcgill.ca
[3] ECE Department, Shahid Beheshti University, Tehran, Iran, v.samadi@sbu.ac.ir

random process in the GA. Chain-Mapping [13] is an algorithm for mapping cores onto a mesh-based Network-on-Chip architecture that its main aim is to produce chains of connected cores in order to introduce a new method to prioritize IP cores. Onyx is a heuristic method for mapping the cores onto a tile-based NoC architecture. Onyx defines four movements to assign priorities to the tiles on a lozenge-shaped path and obtains better communication cost compared with previous mapping algorithms [14]. Also, the mapping of clusters onto the physical topology of processors has been studied in the field of parallel processing. In [15], PMAP, a two phase mapping algorithm for placing clusters onto processors is presented. It is clear to understand that all mapping algorithms try to minimize hop count between related cores as much as possible. This way results in better communication cost, energy consumption, latency and other performance metrics [14].

III. The Proposed Methodology

The objective of this study is to achieve an application-specific NoC design that minimizes the communication cost and improves the fault tolerant properties. This section is consisted of four subsections; the first subsection presents a heuristic mapping algorithm that produces a set of different mappings in a reasonable time. Although this mapping does not explore the design space thoroughly, it considers a part of design space, which in general minimizes the communication costs of solutions while yielding optimum communication costs in some cases. The second subsection deals with routing algorithm used in this paper. Furthermore, the new concept, vulnerability index, which is considered as a criterion for estimating the fault-tolerance of mapped application, is presented in details in the third subsection. In order to yield a mapping which considers trade-offs between communication cost and fault-tolerance, a total cost function and fuzzy logic are defined and introduced at the last two subsections.

A. Citrine: Mapping Algorithm

To formulate mapping problem in a more formal way, we need to first introduce the following two concepts borrowed from [14]:

Definition 1: The core graph is a directional graph $G(V,E)$, whose each vertex $v_i \in V$ shows a core, and a directional edge $e_{i,j} \in E$ illustrates connection between v_i and v_j. The weight of $e_{i,j}$ that is shown as $comm_{i,j}$, represents the bandwidth requirement of the communication from v_i to v_j. We display an IP core along with a router connected to it by Resource Network Interface (RNI) as a tile.

Definition 2: The NoC architecture graph is a directional graph $A(T,L)$, whose each vertex $t_i \in T$ represents a tile in the NoC architecture, and its directional edge that is shown by $l_{i,j} \in L$ shows a physical link from t_i to t_j. The routing path from t_i to t_j is denoted by $r_{i,j}$ and $L(r_{i,j})$ is the set of links that make up the path $r_{i,j}$.

In core graph each edge is treated as a flow of single commodity, represented as c^k and its value which indicates the require bandwidth for each edge is displayed with $vl(c^k)$. The set of all commodities represented as C is achieved as follow:

$$C = \begin{Bmatrix} c^k : vl(c^k) = comm_{i,j}, k = 1,2,\dots|E|, \forall e_{i,j} \in E, \\ with\ source(c^k) = map(v_i),\ \ dest(c^k) = map(v_j) \end{Bmatrix}$$

The core graph mapping $G(V,E)$ on NoC architecture graph $A(T,L)$ is defined by a one to one mapping function.

$$map : V \to T, s.t. map(v_i) = t_j, \forall\, v_i \in V, \exists t_j \in T, |V| \le |T|$$

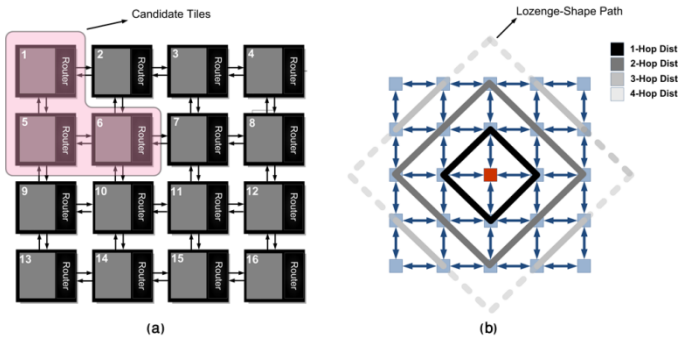


Figure 1. (a) Candidate tiles for mapping the first core (b) concept of lozenge-shape path

The proposed algorithm in [11], searches the optimal solution by alternating branch and bound steps, but it consumes a long run time due to large searching space. By increasing the number of cores, this method becomes unusable as the run time increases significantly. Onyx [14] is one of the best algorithms in mapping of cores onto mesh-based NoC architecture in terms of communication costs. Onyx defines four movements to assign priority to the tiles on a lozenge-shape path and obtains minimum hop count among the directly connected cores in the core graph. The disadvantage of the Onyx algorithm is the priority assignment process, which skips many valuable solutions and generates multiple identical mappings in some cases.

Hence, a new mapping algorithm "Citrine[1]" is proposed by combination of [11] and [14] to better explore design space and using lozenge-shape path to improve run time. In this study, although proposed mapping algorithm is not limited by a specific topology, the mesh topology is selected as a platform for mapping cores due to its flexibility, scalability, and easiness of implementation. Citrine uses Onyx to retrieve the order of cores which should be mapped and branch-tree tries to search different permutation among those defined by lozenge shaped rule of Onyx. Citrine is composed of two steps:

Step1: Mapping of 1st core

A core with the highest priority for mapping is determined by the $highest\_priority(G(V,E))$ function. First, the function finds $comm_{i,j}$ such that it has the highest value, followed by selecting the core with higher $Ranking(C_i)$ as the first core between the source and destination of that communication.

$$Ranking(C_i) = \sum_{\substack{\forall j=1,2,\cdots,|V| \\ i \neq j}} \left(comm_{i,j} + comm_{j,i}\right)$$

After selecting the first core, $\left(\sum_{i=1}^{\lceil \frac{n}{2} \rceil} i\right)$, different tiles can be selected as the candidates for the location of first core in a $n \times n$ mesh. Due to the symmetry of the 2D mesh networks, other tiles are just mirrors of these candidate tiles. The first core should be mapped onto one of the candidate locations and finally the most appropriate location will be determined. As presented by Fig. 1-a Citrine has selected candidate tiles for mapping the first core like Onyx. Each candidate position will be used as a root for a branch-tree.

[1] The Stone of Success

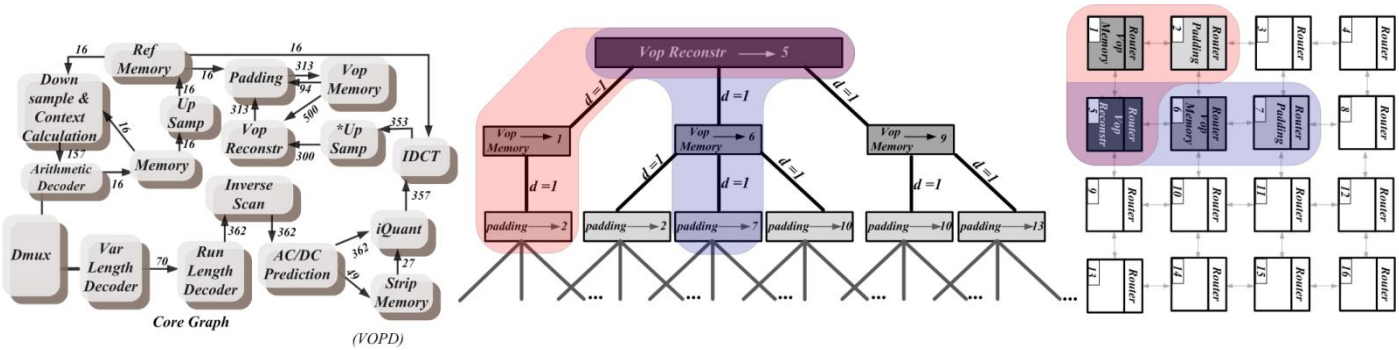


Figure 2. (a) VOPD Core Graph (b) Branch-Tree of VOPD core graph in Citrine (c) Mapping of the smaple branch-tree

Step2: Branch the tree

After mapping of the first core in one of the candidate tiles, unmapped cores are selected for mapping upon Onyx priority order. Onyx priority order defines that for mapping the next core, a core with the largest $(comm_{p,c} + comm_{c,p})$ should be selected, in which v_p is a mapped core and v_c is the next core to be mapped for all possible values of c and p. In other words, a core is selected for mapping which is not mapped yet and requires the largest bandwidth to communicate with one of the mapped cores among others. As the v_c is selected by its relation to the v_p, the v_p is called the parent and v_c as the child. After selecting a child to be mapped, different permutations should be searched for the child. The search will be carried out using the idea of lozenge-shape path of Onyx presented in Fig. 1-b. The searching algorithm is such that a child should be mapped with the nearest possible tiles to the parent. This rule defines that if d is the minimum possible distance between a child, v_c , and its parent, v_p , then all available tiles with distance d from the parent, v_p , should be considered as possible permutations for the child, v_c. It is worth noting that the available tiles are those, which are not occupied by a previously mapped tile. Citrine continues the procedure of selecting the next core to map and finds all possible permutations until completing the mappings. Fig. 2 shows in detail mapping of a real core graph, VOPD, for the first three cores.

1) Experimental results of Citrine

VOPD and MPEG-4 real core graphs are used to compare Citrine algorithm with other algorithms. The results of Citrine are compared with BMAP[3], results of NMAP, Partial Branch-and-Bound, GMAP, PMAP mentioned in [10], CGMAP[12], CHMAP[13] and Onyx[14].

Communication cost is the main criteria for comparing these algorithms and is calculated according to the following equation:

$$commcost = \sum_{k=1}^{|E|} vl(c^k) \times hop\_count\big(src(c^k), dst(c^k)\big)$$

where $src(c^k)$ is source and $dst(c^k)$ is destination of c^k.

In Fig.3 and Fig. 4 the minimum communication cost for all algorithms are presented. Communication costs of PMAP and GMAP are exceeding the upper bounds in charts so they

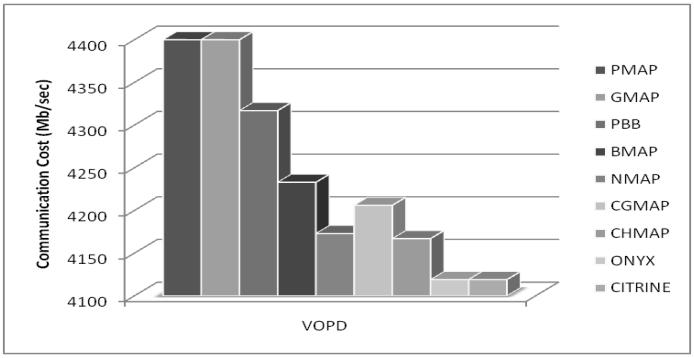


Figure 3. Communication cost of different mapping algorithms on VOPD core graph

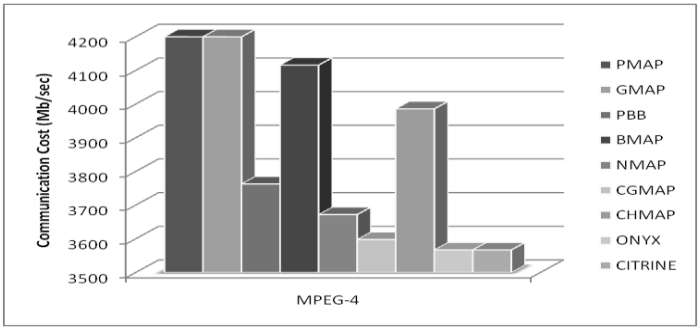


Figure 4. Communication cost of different mapping algorithms on MPEG-4 core graph

Communication costs of PMAP and GMAP are exceeding the upper bounds in charts so they are not shown completely in these figures. As illustrated in these figures, Citrine mapping algorithm and Onyx show the minimum communication cost. Lower communication cost is the result of smaller hop count between related cores. Reducing the hop counts is one of the most significant approaches for decreasing energy consumption and other performance metrics like latency [14]. As illustrated in Fig.3 and Fig. 4, Onyx and Citrine show similar best results in VOPD and MPEG-4 core graphs.

In order to better estimate Citrine abilities, much more complicated graphs are applied to the Citrine and the Onyx. The results of applying the Citrine and the Onyx as well as the perfect results are listed in Table 1. For NUG12, NUG15, NUG16b, NUG21,NUG24 and NUG25 [16] the Citrine shows 4.53%, 3.90%, 1.37%, 2.47%, 6.18% and 5.68% improvements

over Onyx, respectively. Figure 5 illustrates the results which demonstrated in Table 1.

TABLE I. COMPARISON OF THE COMMUNICATION COST OF CITRINE AND ONYX VS. PERFECT RESULTS

Case Study	Perfect Results	CITRINE	ONYX
NUG12	578	590	618
NUG15	1150	1182	1230
NUG16b	1244	1290	1308
NUG21	2438	2526	2590
NUG24	3488	3608	3846
NUG25	3744	3912	4148

B. *Routing*

This section focuses on the routing algorithm used in the proposed application-specific NoCs resulted after mapping procedure, as explained in the previous section. Following the mapping of cores onto a given topology, the routing algorithm significantly affects the overall performance of a Network-on-Chip. The deterministic routing algorithms are widely used in application-specific NoCs due to their simplicity and low area overhead in the router design. On the other hand, the adaptive routing algorithms usually result in a better performance and link utilization than deterministic routing algorithm. Another advantage of using the adaptive routing algorithm is the fault-tolerant NoCs by providing multiple alternative paths through network.

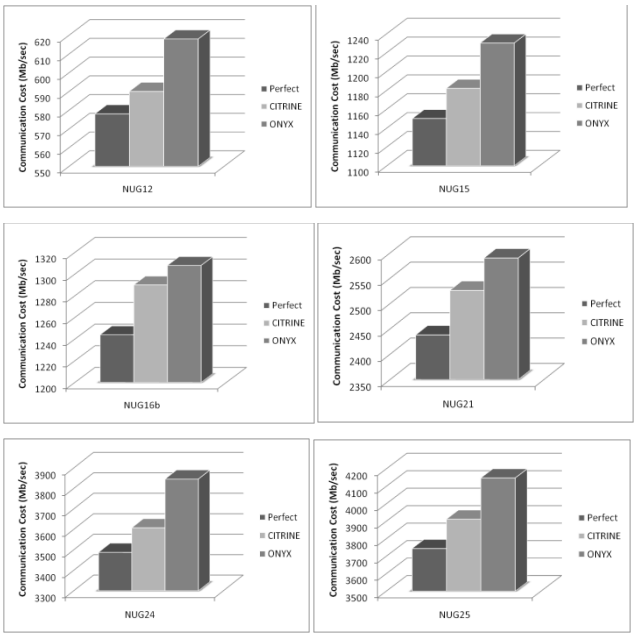


Figure 5. Communcation cost of Citrine, Onyx and perfect answers for NUG12, NUG15, NUG16b, NUG21, NUG24, NUG25

One of the best algorithms customized for routing in application-specific NoCs was presented by [5] which uses a highly adaptive deadlock-free routing algorithm. This routing algorithm takes into account the communication bandwidth requirements among core pairs which are mapped on different network nodes. The algorithm splits the communication, c^k, over multiple paths provided by routing function, between the source, S, and the destination, D, as shown in Fig.6. The routing function presented in [5] uses fully adaptive minimal routing. In general, every routing algorithm should include deadlock freedom feature. Nevertheless, more considerations are required when implementing the fully adaptive routing algorithms. The adaptive routing algorithm is prone to deadlocks, so channel dependency graphs (CDG) concept is used in [5] to avoid any possible deadlocks. The CDG is a directed graph with the network channels as the vertices and the direct dependencies between the two channels as the edges.

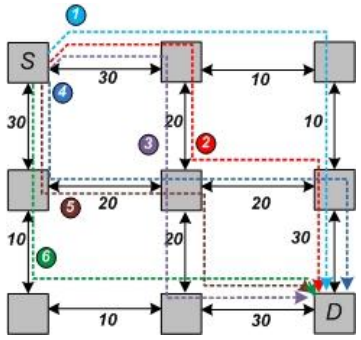


Figure 6. Splitting of bandwidth based on fully adpative minimal routing for a communication from node S to node D at 60 Mb/s

A dependency exists between the links $l_{i,j}$ and $l_{j,k}$ whenever there is a path to route packets from v_i to v_k through v_j which uses those links. An extension to CDG as a sub graph is the concept of application specific channel dependency graph (ASCDG) introduced in [17]. The ASCDG is a sub graph of the CDG and an edge in CDG between channels, $l_{i,j}$ and $l_{j,k}$ is removed if there was no application-specific dependency between $l_{i,j}$ and $l_{j,k}$. Deadlock is not inevitable when there are any cycles through ASCDG graph. A cycle in the ASCDG is a succession of application specific direct dependencies, $D = \{d_1, d_2, \cdots, d_n\}$, where $d \in D$ is a pair $(l_{i,j}, l_{j,k})$ with $l_{i,j}, l_{j,k} \in L$. The challenge is how to select the most appropriate dependencies to be removed from ASCDG graph to break the cycle, D. By removing a dependency, all of the corresponding paths to that dependency will be removed. Using this method guarantees that routing algorithm is deadlock free still with high levels of adaptivity. Palesi *et al.* also presented a technique to break cycles in the ASCDG which is bandwidth aware. As soon as a path is removed, the fraction portion of the bandwidth that passes through it must be redistributed over the remaining paths. The idea is to remove the dependency, d, such that the overhead of bandwidth that should be allocated to the remaining paths is minimized. The condition for removing the edge from the ASCDG is to minimize the cost (d), as:

$$\text{cost}(d) = \sum_{c^k \in C} \frac{vl(c^k) \times |PT^2(c^k, d)|}{|\rho(c^k)| \times (|\rho(c^k)| - |PT^2(c^k, d)|)}$$

where, $vl(c^k)$ represents bandwidth requirements for communication, c^k, $\rho(c^k)$ denotes the set of minimal paths admitted by the routing function for c^k, and $PT^2(c^k, d)$ is the path through dependency set that is the set of paths of c^k which use the dependency d.

Due to its inherent application-specific nature and high adaptivity provided by algorithm proposed by [5], this algorithm is used for routing of application-specific NoCs, which previously mapped by Citrine in section 3.1. Fault-tolerant properties are also considered in the proposed

methodology to include the advantage of adaptivity in routing algorithms. Next section deals with fault-tolerance by introducing the concept of vulnerability index.

C. Vulnerability Index

As the number of transistors on chip increases, the problem associate with deep sub-micron will become more pronounced. Moreover, the router and link failures will be more probable. Therefore, the NoCs need to be designed with high level of built-in fault tolerance [18]. Vulnerability index is considered as a criterion for estimating fault-tolerant properties of NoCs. By reducing the vulnerability index, the NoC design will become more fault-tolerant. The vulnerability index has the inverse relation to the robustness index introduced in [19] and definitions and formulations of robustness index are also extracted from the same reference. So first we define the robustness index and then definition of vulnerability index is proposed.

The robustness index, RI, is based on the extension of the concept of path diversity [20]. For a given communication, $c^k \in C$, an NoC architecture graph, $A(T, L)$, a mapping function, M, and a routing function, R, [19] defined the robustness index for communication c^k, $RI(c^k)$, as the average number of routing paths available for communication, c^k, if a link belonging to the set of links used by communication c^k is faulty. Formally,

$$RI(c^k) = \frac{1}{|L(c^k)|} \sum_{l_{i,j} \in L} \left| \rho(c^k) \backslash \rho(c^k, l_{i,j}) \right|$$

where, $\rho(c^k)$ is the set of paths provided by R for communication, c^k, $\rho(c^k, l_{i,j})$ is the set of paths provided by R for communication, c^k,that uses link $l_{i,j}$, and $L(c^k)$ is the set of links belonging to paths in $\rho(c^k)$.

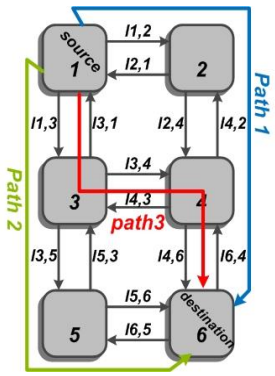


Figure 7. Routing paths provided by two different routing functions for the communicating pair (source, destination)

Suppose that there are two routing functions, A and B, which routing function A selects *path 1* and $path2$ and routing function B selects $path2$ and $path3$ to route packets between source and destination as shown in Fig. 7 . The routing function A selects two disjoint paths such that the presence of a faulty link in one path dose not compromise communication from source to destination since another path is fault-free. However, when the routing function B is used as shown in Fig. 7, the communication will not occur. As the alternative paths share the link, $l_{1,3}$, any fault in the link, $l_{1,3}$, makes the communication from "source" to "destination" impossible. Consequently, the NoC which uses routing function A, NOC_1, is more robust than the NoC which uses routing function B, let call it NOC_2. Such situation is reflected by the robustness index. The robustness index for the above two cases are:

$$RI^{(NOC_1)}(\text{source} \rightarrow \text{destination}) = \frac{1+1+1+1+1+1}{6} = 1\,,$$

$$RI^{(NOC_2)}(\text{source} \rightarrow \text{destination}) = \frac{0+1+1+1+1}{5} = 0.8\,.$$

The NOC_1using $path1$ and $path2$ is more robust than the NOC_2 using $path2$ and $path3$ for communication from "$source$" to "$destination$" as $RI^{(NOC_1)} > RI^{(NOC_2)}$.

The global robustness index, which characterizes the network, is calculated using the weighted sum of the robustness index of each communication. For a communication, c^k, the weight of $RI(c^k)$ is the degree of adaptivity [21] of c^k. The degree of adaptivity of a communication, c^k, is the ratio of the number of allowed minimal paths to the total number of possible minimal paths between the source node and the destination node associated to c^k. Thus, given a core graph $G(V, E)$, NoC architecture graph $A(T, L)$, a mapping function M, and a routing function R, the robustness index is defined as:

$$RI^{(NOC)} = \sum_{c^k \in C} \alpha(c^k) RI^{(NOC)}(c^k)$$

where $\alpha(c^k)$ indicates the degree of adaptivity of communication c^k.

So after defining $RI^{(NOC)}$, the Vulnerability index, $VI^{(NOC)}$, defined as:

$$VI^{(NOC)} = \frac{1}{\varepsilon + RI^{(NOC)}}$$

where ε is a constant to be defined. Note that $RI^{(NOC)}$ is always equal or greater than zero, therefore the maximum possible value of $VI^{(NOC)}$ is equal to $\frac{1}{\varepsilon}$ which happens in case $RI^{(NOC)}$ is equal to zero. In this paper ε was set to 0.01. If we want to set an upper bound for $VI^{(NOC)}$, let call it VI_Upper_Bound, ε will be calculated as follow:

$$\varepsilon = \frac{1}{VI\_Upper\_Bound}$$

Following section deals with a cost function that its objective is to find a mapping such that the sum of weighted communication cost and vulnerability index are to be minimized under previously mentioned routing algorithm.

D. Total cost function

As previously mentioned, lower communication cost leads to an NoC with better metrics such as energy consumption and latency. Another introduced metric was vulnerability index which is used as a measurable criterion for fault tolerant properties. A cost function is to be introduced in order to minimize the sum of weighted communication cost and the vulnerability index.

Given a core graph $G(V,E)$, NoC architecture graph $A(T,L)$, and a routing function R, a mapping function, M, is introduced from $G(V,E) \rightarrow A(T,L)$ to minimize the following:

$$Total\ Cost\ Function = \frac{(1-\alpha)}{\beta} \times commcost + \frac{\alpha}{\gamma} \times VI^{(NOC)}$$

where, $commcost$ is the communication cost and $VI^{(NOC)}$ is the vulnerability index of NoC after applying mapping function.

The constants β and γ are used to normalize the $commcost$ and $VI^{(NOC)}$. In this study the maximum value of communication cost and vulnerability index are used for β and γ, respectively. α is a weighting coefficient meant to balance the communication cost and vulnerability index. It was set to 0.8 and 0.7 for core graphs VOPD and MPEG-4 respectively.

1) Experimental Results

In order to better investigate the capabilities of Citrine mapping algorithm and for better understanding of total cost function, we have done some experiments. As mentioned before, one of the advantages of Citrine over other algorithms is its diversity of solutions which have near to optimum communication costs. Although Citrine explores a wide range of solutions, its runtime for examined real core graphs, VOPD and MPEG-4, was a fraction of a second.

We have run Citrine for core graphs VOPD and MPEG-4 to evaluate the generated mappings by using total cost function.

Citrine generates 6420 and 2808 different mappings for MPEG-4 and VOPD, respectively. Some mappings have the same communication cost and vulnerability index values. So by dismissing the duplicate items, the unique values for MPEG-4 and VOPD were extracted among whole results. Results of running Citrine for MPEG-4 and VOPD core graphs and evaluating the values in our 2D design space, i.e. communication cost and vulnerability index shown in Fig. 8 and Fig. 9, respectively for extracted set of results.

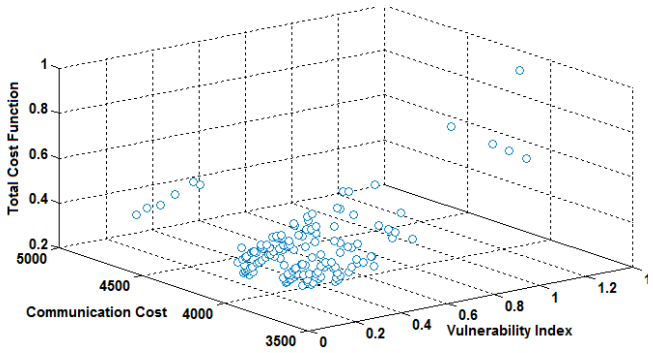


Figure 8. Citrine mappings of MPEG-4 core graph

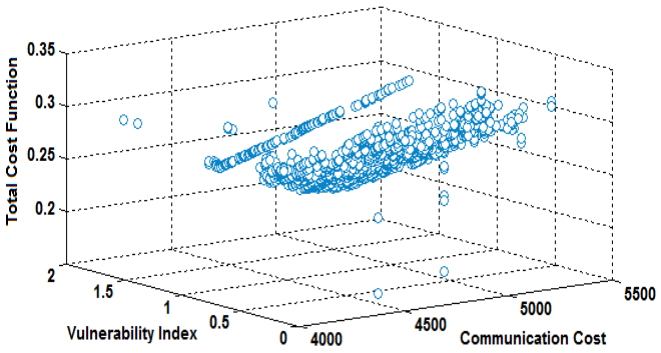


Figure 9. Citrine mappings of VOPD core graph

Although the communication costs for most of mappings are spread in a range near to optimal, there are wide ranges of total cost for mappings due to different vulnerability index values, as it can be seen in some extreme points in design space which have small communication cost but having large total cost and vulnerability indices. Notice that Citrine does not take vulnerability index into account while generating mapping but by providing a large design space, we can achieve different total costs which is one of the good points about Citrine.

For better analyzing the correlation of different parameters in our design space, Fig. 10 to Fig. 13 are shown for results of VOPD core graph.

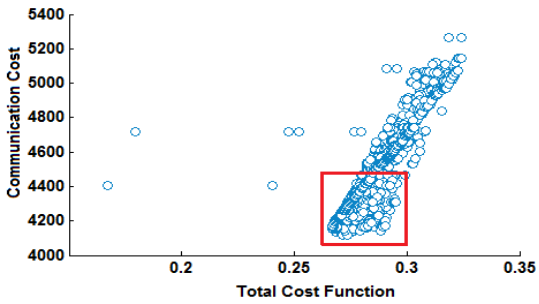


Figure 10. Commnication Cost vs. Total Cost Function

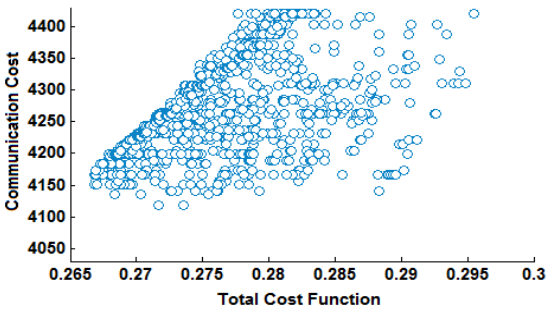


Figure 11. Commnication Cost vs. Total Cost Function – detailed

Figure 12. Total Cost Function vs. Vulnerability Index

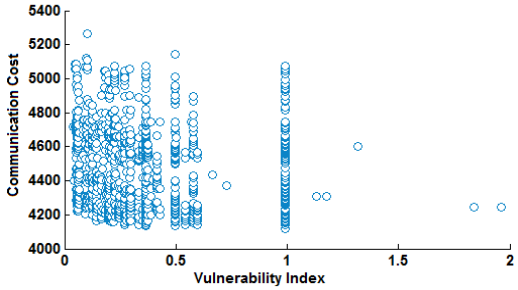


Figure 13. Communication Cost vs. Vulnerability

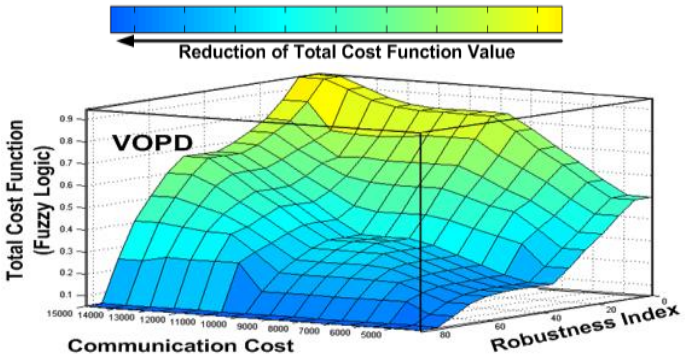
Fig. 10 considers communication cost vs. total cost. It is observed that increasing communication cost leads to greater values of total cost in most cases but this is not true for all. For better understanding, a set of results which highlighted in red box are shown in Fig. 11. Although the best communication cost for VOPD is 4119, this mapping has not the best total cost. It is worth noting that weighting coefficient $\alpha = 0.8$ which means that we only left a minor effect for communication cost in total cost function.

Fig. 12 illustrates total cost vs. vulnerability index. Description of this figure is like two former described figures, Fig. 10 and Fig. 11.

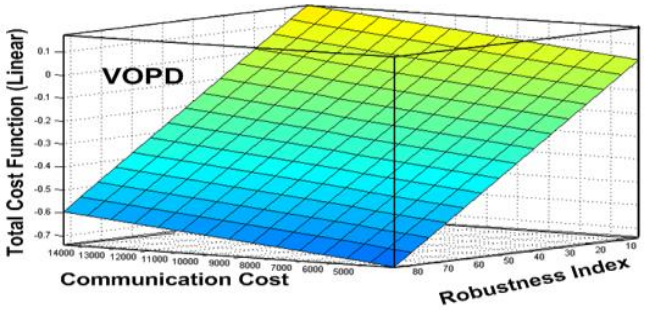
At last, Fig. 13 depicts communication cost vs. vulnerability index without noticing total cost function. As you can see, the power of Citrine is to generated different mappings with diverse set of features, e.g. mappings with same communication costs but different vulnerability indices and vice versa.

E. Fuzzy Logic Solution

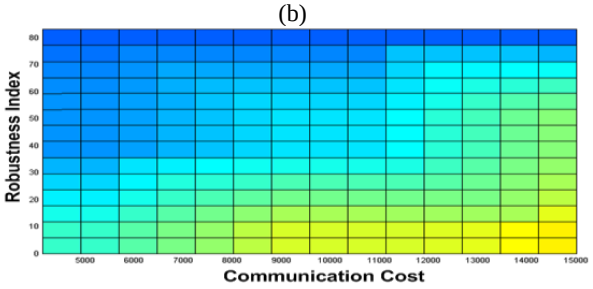
Over the past few decades, fuzzy logic has been used in a wide range of problem domains and provides an alternative way of thinking, which allows to model complex systems using a higher level of abstraction originating from the user knowledge and experience [22]. In this section, fuzzy logic is used to find an appropriate application-specific network on chip configuration according to designer's demands. In the proposed fuzzy system, the Mamdani's fuzzy interface is used and if-then rules are presented in Fig. 15(e). As it can be seen, these rules are flexible and easy to use, which are the key benefits of the fuzzy logic. Fig.15 (a) to Fig.15 (d) show the total design cost surfaces of fuzzy logic and linear function, which highlight advantages of fuzzy logic. As it was shown in the results, in this step we have evaluated our fuzzy logic design by considering two parameters Robustness Index and Communication cost. It is observed that the designer has more flexibility to prioritize solutions within the design space by adjusting a set of if-then rules.



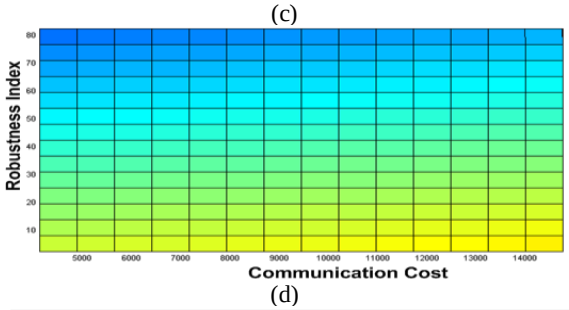
(a)



(b)



(c)



(d)

Cost	*Robustness Index*	*Output*
---	Excellent	VeryExcellent
Excellent	Good	Excellent
Good	Good	Very Good
Excellent	Bad	Good
Bad	Good	Average
Good	Bad	Bad
Bad	Bad	VeryBad

(e)

Figure15. , (a)3D demonstration of fuzzy logic of VOPD mappings (b) 3D demonstration of linear function of VOPD mappings (c) 2D demonstration of fuzzy logic of VOPD mappings (d)2D demonstration of linear function of VOPD mappings, (e) If-Then rules of fuzzy logic using "and" operator.

IV. Conclusion

Considering the importance of mapping algorithm as one of three aspects of NoC design, in this paper, a novel mapping algorithm was proposed, called Citrine. It was able to obtain the best communication cost among a number of efficient mapping algorithms. Vulnerability index was also introduced and defined as a criterion for evaluating routing fault tolerance. The most appropriate mapping is selected by total cost function using either a linear or fuzzy logic cost function. The function can be customized by a designer, considering the impact of two key parameters, i.e., communication cost and vulnerability index.

Refrences

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks", in *Proceedings of the 38th Design Automation Conference (DAC)*, 2001.

[2] F. Clermidy, R. Lemaire, Y. Thonnart, P. Vivet, "A Communication and configuration controller for NoC based reconfigurable data flow architecture," NoCS 2009. 3rd ACM/IEEE International Symposium on , pp153-162, 12 June 2009.

[3] W. Shen, C. Chao, Y. Lien, A. Wu," A NEW BINOMIAL MAPPING AND OPTIMIZATION ALGORITHM FOR REDUCED-COMPLEXITY MESH-BASED ON-CHIP NETWORK,", Networks-on-Chip, NOCS 2007, pp.317 – 322, 7-9 May 2007.

[4] J. Chen, D. Xu, L. Xie, "A Positive-first and Negative-first Fault-tolerant routing schemes for concave and convex faults," Future Computer and Communication (ICFCC), 2010 2nd International Conference on, May 2010, pp. V1-53 - V1-58.

[5] M. Palesi, G. Longo, S. Signorino, R. Holsmark, S. Kumar, V. Catania, "Design of Bandwidth Aware and Congestion Avoiding Efficient Routing Algorithms for Networks-on-Chip Platforms", Networks-on-Chip, NoCS 2008. Second ACM/IEEE International Symposium on, pp. 97 – 106, April 2008.

[6] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, "Efficient Link Capacity and QoS Design for Network-on-Chip," Design, Automation and Test in Europe, 2006. DATE '06. Proceedings, pp.1-6, 24 July 2006.

[7] JERGER N.E., PEH L.S., LIPASTI M.H.: 'Virtual circuit tree multicasting: a case for on-chip hardware multicast support'. Proc. Int. Conf. Computer Architecture, China, 2008, pp. 229–240

[8] Chen-Ling Chou, R. Marculescu, " Contention-aware Application Mapping for Network-on-Chip Communication Architectures", Computer Design, ICCD , IEEE International Conference on ,pp. 164 – 169, Oct. 2008.

[9] M. Janidarmian, A. Khademzadeh, A. Roshan Fekr, V. Samadi Bokharaei," Citrine: A Methodology for Application-Specific Network-on-Chips Design," Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2010, WCECS 2010, 20-22 October, 2010, San Francisco, USA, pp. 196-202.

[10] S.Murali, V.De Micheli, "bandwidth constrained mapping of cores onto NoC architectures", Design, Automation and Test in Europe Conference and Exhibition, Proceedings, Vol.2,pp. 896- 901, Feb. 2004.

[11] Hu Jingcao, R. Marculescu, "Energy aware mapping for tile-based NoC architectures under performance constraints", Design Automation Conference, Proceedings of the ASP-DAC 2003. Asia and South Pacific,pp. 233- 239, Jan. 2003.

[12] Fahime Moein-darbari, Ahmad Khademzade and Golnar Gharooni-fard, "CGMAP: a new approach to Network-on-Chip mapping problem", IEICE Electron. Express, Vol. 6, No. 1, pp.27-34, (2009) .

[13] Misagh Tavanpour, Ahmad Khademzadeh and Majid Janidarmian, "Chain-Mapping for mesh based Network-on-Chip architecture", IEICE Electron. Express, Vol. 6, No. 22, pp.1535-1541, (2009) .

[14] M. Janidarmian, A. Khademzadeh, M. Tavanpour, "Onyx: A new heuristic bandwidth-constrained mapping of cores onto tile based Network on Chip", IEICE Electron. Express, Vol. 6, No. 1, pp.1-7, January 2009.

[15] N. Koziris et al.," An efficient Algorithm for the physical mapping of clustered Task Graphs onto Multiprocessor Architectures ," Proceedings of 8th EuroPDP, pp.406-413, Jan.2000.

[16] http://www.seas.upenn.edu/qaplib/inst.html

[17] M. Palesi, R.Holsmark, S.Kumar, "a methodology for design of application specific deadlock-free routing algorithms for NoC systems", Hardware/Software Codesign and System Synthesis, CODES+ISSS '06. Proceedings of the 4th International Conference,pp. 142-147, Oct. 2006.

[18] M. Ali, M. Welzl, S. Hessler, S. Hellebrand, "A Fault tolerant mechanism for handling Permanent and Transient Failures in a Network on Chip," Information Technology, 2007. ITNG '07. Fourth International Conference on, p.p. 1027-1032, 2007

[19] Rafael Tornero, Valentino Sterrantino, Maurizio Palesi ,Juan M. Orduna," A Multi-objective Strategy for Concurrent Mapping and Routing in Networks on Chip" Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing",pp. 1-8 ,2009.

[20] W. J. Dally and B. Towles, Principle and Practice of Interconnection Network. San Francisco, CA : Morgan Kaufmann, 2004

[21] C. J. Glass L. M. Ni, "The turn model for adaptive routing," Journal of the Association for Computing Machinery, vol. 41, no. 5, pp. 874-902, Sep. 1994.

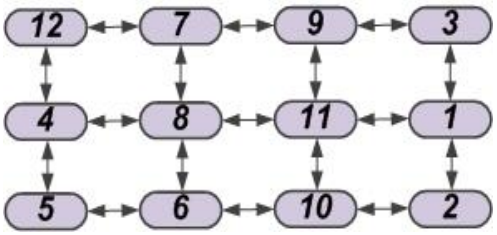
[22] I.Nedeljkovic, "IMAGE CLASSIFICATION BASED ON FUZZY LOGIC", The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. 34, Part XXX, 2004.

I. APENDIX

1) Graphs used in experiments of Ruby in 3.1

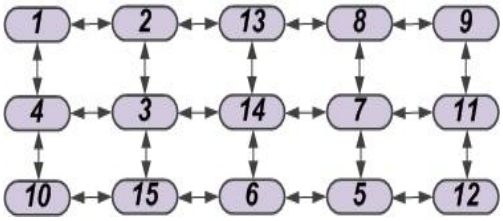
Nug12:

```
 0  5  2  4  1  0  0  6  2  1  1  1
 5  0  3  0  2  2  2  0  4  5  0  0
 2  3  0  0  0  0  0  5  5  2  2  2
 4  0  0  0  5  2  2 10  0  0  5  5
 1  2  0  5  0 10  0  0  0  5  1  1
 0  2  0  2 10  0  5  1  1  5  4  0
 0  2  0  2  0  5  0 10  5  2  3  3
 6  0  5 10  0  1 10  0  0  0  5  0
 2  4  5  0  0  1  5  0  0  0 10 10
 1  5  2  0  5  5  2  0  0  0  5  0
 1  0  2  5  1  4  3  5 10  5  0  2
 1  0  2  5  1  0  3  0 10  0  2  0
```



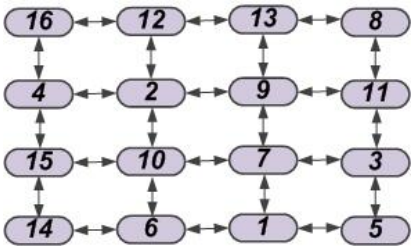
Nug15:

```
 0 10  0  5  1  0  1  2  2  2  2  0  4  0  0
10  0  1  3  2  2  2  3  2  0  2  0 10  5  0
 0  1  0 10  2  0  2  5  4  5  2  2  5  5  5
 5  3 10  0  1  1  5  0  0  2  1  0  2  5  0
 1  2  2  1  0  3  5  5  5  1  0  3  0  5  5
 0  2  0  1  3  0  2  2  1  5  0  0  2  5 10
 1  2  2  5  5  2  0  6  0  1  5  5  5  1  0
 2  3  5  0  5  2  6  0  5  2 10  0  5  0  0
 2  2  4  0  5  1  0  5  0  0 10  5 10  0  2
 2  0  5  2  1  5  1  2  0  0  0  4  0  0  5
 2  2  2  1  0  0  5 10 10  0  0  5  0  5  0
 0  0  2  0  3  0  5  0  5  4  5  0  3  3  0
 4 10  5  2  0  2  5  5 10  0  0  3  0 10  2
 0  5  5  5  5  5  1  0  0  0  5  3 10  0  4
 0  0  5  0  5 10  0  0  2  5  0  0  2  4  0
```



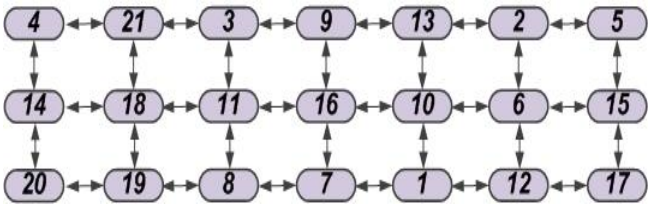
Nug16b:

```
 0  0  5  0  2 10  3  1  5  5  0  0  4  4  0  0
 0  0  3 10  1  5  1  2  2  5  0 10  3  0  5 10
 5  3  0  2  5  2  4  4  0  0  0  5  0  0  5  0
 0 10  2  0  0  5  2  1 10  2  2  0  1  5  2  5
 2  1  5  0  0  5  2  1  0  0 10  0  0  1  0  1
10  5  2  5  5  0  0  0  5 10  2  2  1  2  1  0
 3  1  4  2  2  0  0  1 10 10  2  0  2  5  2  2
 1  2  4  1  1  0  1  0  0  3  5  5  5  0  0  0
 5  2  0 10  0  5 10  0  0  5  2  5 10  0  2  2
 5  5  0  2  0 10 10  3  5  0  2 10  0  1  1  2
 0  0  0  2 10  2  2  5  2  2  0  2  1  0  0  0
 0 10  5  0  0  2  0  5  5 10  2  0  5  1  5  5
 4  3  0  1  0  1  2  5 10  0  1  5  0  0  0  2
 4  0  0  5  1  2  5  0  0  1  0  1  0  0  5  2
 0  5  5  2  0  1  2  0  2  1  0  5  0  5  0  1
 0 10  0  5  1  0  2  0  2  2  0  5  2  2  1  0
```



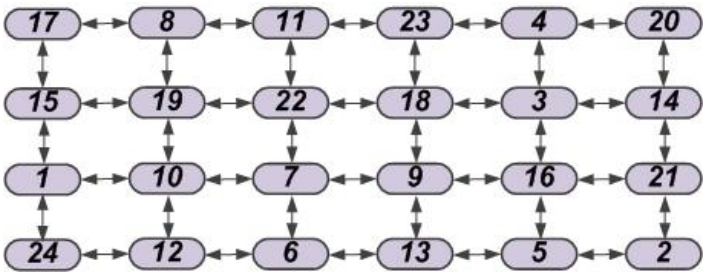
Nug21:

```
 0  3  2  0  0  2 10  5  0  5  2  5  0  0  2  0  5  6  3  0  1
 3  0  4  0 10  4  0  0  2  2  1  0  5  0  0  0  0  2  0  1  6
 2  4  0  3  4  0  5  5  5  1  4  1  0  4  0  4  0  6  3  2  5
 0  0  3  0  0  0  0  0  2  2  0  6  0  2  5  2  5  1  1  1  1  2
 0 10  4  0  0  5  2  0  0  0  0  0  2  0  0  0  0  2  1  0  0  2
 2  4  0  0  5  0  1  2  2  1  4 10 10  2  5  5  0  5  0  0  0
10  0  5  0  2  1  0 10 10  5 10 10  6  0  0 10  2  1 10  1  5
 5  0  5  2  0  2 10  0  1  3  5  0  0  0  2  4  5  2 10  6  0
 0  2  5  2  0  2 10  1  0 10  2  1  5  2  0  3  0  2  0  0  4
 5  2  1  0  0  1  5  3 10  0  5  5  6  0  1  5  5  0  5  2  3
 2  1  4  6  0  4 10  5  2  5  0  0  0  1  2  1  0  2  0  0  0
 5  0  1  0  2 10 10  0  1  5  0  0  5  5  2  0  0  0  0  2  0
 0  5  0  2  0 10  6  0  5  6  0  5  0  2  0  4  2  2  1  0  6
 0  0  4  5  0  2  0  0  2  0  1  5  2  0  2  1  0  5  3 10  0
 2  0  0  2  0  5  0  2  0  1  2  2  0  2  0  4  5  1  0  1  0
 0  0  4  5  0  5 10  4  3  5  1  0  4  1  4  0  0  3  0  2  2
 5  0  0  1  2  0  2  5  0  5  0  0  2  0  5  0  0  2  2  0  0
 6  2  6  1  1  5  1  2  2  0  2  0  2  5  1  3  2  0  5  1  2
 3  0  3  1  0  0 10 10  0  5  0  0  1  3  0  0  2  5  0  0  5
 0  1  2  1  0  0  1  6  0  2  0  2  0 10  1  2  0  1  0  0  5
 1  6  5  2  2  0  5  0  4  3  0  0  6  0  0  2  0  2  5  5  0
```



Nug24:

```
 0  3  2  0  0  2 10  5  0  5  2  5  0  0  2  0  5  6  3  0  1 10  0 10
 3  0  4  0 10  4  0  0  2  2  1  0  5  0  0  0  0  2  0  1  6  1  0  1
 2  4  0  3  4  0  5  5  5  1  4  1  0  4  0  4  0  6  3  2  5  5  2  1
 0  0  3  0  0  0  0  2  2  0  6  0  2  5  2  5  1  1  1  1  2  2  4  0
 0 10  4  0  0  5  2  0  0  0  0  2  0  0  0  0  2  1  0  0  2  0  5  1
 2  4  0  0  5  0  1  2  2  1  4 10 10  2  5  5  0  5  0  0  0 10  0  0
10  0  5  0  2  1  0 10 10  5 10 10  6  0  0 10  2  1 10  1  5  5  2  3
 5  0  5  2  0  2 10  0  1  3  5  0  0  0  2  4  5  2 10  6  0  5  5  2
 0  2  5  2  0  2 10  1  0 10  2  1  5  2  0  3  0  2  0  0  4  0  5  2
 5  2  1  0  0  1  5  3 10  0  5  5  6  0  1  5  5  0  5  2  3  5  0  5
 2  1  4  6  0  4 10  5  2  5  0  0  0  1  2  1  0  2  0  0  0  6  6  0
 5  0  1  0  2 10 10  0  1  5  0  0  5  5  2  0  0  0  0  2  0  4  5 10
 0  5  0  2  0 10  6  0  5  6  0  5  0  2  0  4  2  2  1  0  6  2  1  5
 0  0  4  5  0  2  0  0  2  0  1  5  2  0  2  1  0  5  3 10  0  0  4  2
 2  0  0  2  0  5  0  2  0  1  2  2  0  2  0  4  5  1  0  1  0  5  0  2
 0  0  4  5  0  5 10  4  3  5  1  0  4  1  4  0  0  3  0  2  2  0  2  0
 5  0  0  1  2  0  2  5  0  5  0  0  2  0  5  0  0  2  2  0  0  0  6  5
 6  2  6  1  1  5  1  2  2  0  2  0  2  5  1  3  2  0  5  1  2 10 10  4
 3  0  3  1  0  0 10 10  0  5  0  0  1  3  0  0  2  5  0  0  5  5  1  0
 0  1  2  1  0  0  1  6  0  2  0  2  0 10  1  2  0  1  0  0  5  2  1  3
 1  6  5  2  2  0  5  0  4  3  0  0  6  0  0  2  0  2  5  5  0  4  0  1
10  1  5  2  0 10  5  5  0  5  6  4  2  0  5  0  0 10  5  2  4  0  5  0
 0  0  2  4  5  0  2  5  5  0  6  5  1  4  0  2  6 10  1  1  0  5  0  0
10  1  1  0  1  0  3  2  2  5  0 10  5  2  2  0  5  4  0  3  1  0  0  0
```



Nug25:

```
 0  3  2  0  0 10  5  0  5  2  0  0  2  0  5  3  0  1 10  0  2  1  1  1  0
 3  0  4  0 10  0  0  2  2  1  5  0  0  0  0  0  1  6  1  0  2  2  5  1 10
 2  4  0  3  4  5  5  5  1  4  0  4  0  4  0  3  2  5  5  2  0  0  3  1  0
 0  0  3  0  0  0  2  2  0  6  2  5  2  5  1  1  1  2  2  4  2  0  2  2  5
 0 10  4  0  0  2  0  0  0  0  0  0  0  0  0  2  0  0  2  0  5  0  2  1  0  2
10  0  5  0  2  0 10 10  5 10  6  0  0 10  2 10  1  5  5  2  5  0  2  0  1
 5  0  5  2  0 10  0  1  3  5  0  0  2  4  5 10  6  0  5  5  5  0  5  5  0
 0  2  5  2  0 10  1  0 10  2  5  2  0  3  0  0  0  4  0  5  0  5  2  2  5
 5  2  1  0  0  5  3 10  0  5  6  0  1  5  5  5  2  3  5  0  2 10 10  1  5
 2  1  4  6  0 10  5  2  5  0  0  1  2  1  0  0  0  0  6  6  4  5  3  2  2
 0  5  0  2  0  6  0  5  6  0  0  2  0  4  2  1  0  6  2  1  5  0  0  1  5
 0  0  4  5  0  0  0  0  2  0  1  2  0  2  1  0  3 10  0  0  4  0  0  4  2  5
 2  0  0  2  0  0  2  0  1  2  0  2  0  4  5  0  1  0  5  0  0  0  5  1  1
 0  0  4  5  0 10  4  3  5  1  4  1  4  0  0  0  2  2  0  2  5  0  5  2  5
 5  0  0  1  2  2  5  0  5  0  2  0  5  0  0  2  0  0  0  6  3  5  0  0  5
 3  0  3  1  0 10 10  0  5  0  1  3  0  0  2  0  0  5  5  1  5  2  1  2 10
 0  1  2  1  0  1  6  0  2  0  0 10  1  2  0  0  0  5  2  1  1  5  6  5  5
 1  6  5  2  2  5  0  4  3  0  6  0  0  2  0  5  5  0  4  0  0  0  0  5  0
10  1  5  2  0  5  5  0  5  6  2  0  5  0  0  5  2  4  0  5  4  4  5  0  2
 0  0  2  4  5  2  5  5  0  6  1  4  0  2  6  1  1  0  5  0  4  4  1  0  2
 2  2  0  2  0  5  5  0  2  4  5  0  0  5  3  5  1  0  4  4  0  1  0 10  1
 1  2  0  0  2  0  0  5 10  5  0  0  0  0  5  2  5  0  4  4  1  0  0  0  0
 1  5  3  2  1  2  5  2 10  3  0  4  5  5  0  1  6  0  5  1  0  0  0  0  0
 1  1  1  2  0  0  5  2  1  2  1  2  1  2  0  2  5  5  0  0 10  0  0  0  2
 0 10  0  5  2  1  0  5  5  2  5  5  1  5  5 10  5  0  2  2  1  0  0  2  0
```