IPSJ Transactions on System LSI Design Methodology Vol.6 34–41 (Feb. 2013)

[DOI: 10.2197/ipsjtsldm.6.34]

Regular Paper

An Efficient Algorithm for 3D NoC Architecture Optimization

Xin Jiang[1,a)] Ran Zhang[1] Takahiro Watanabe[1]

Received: May 25, 2012, Revised: August 30, 2012,
Accepted: October 30, 2012, Released: February 15, 2013

Abstract: With the progress of 3D IC integration technologies, the application of 3D Networks-on-chip (NoCs) has been proposed as a scalable and efficient solution to the global communication in the interconnect designs. In this work, we propose a new procedure for designing application specific irregular 3D NoC architectures. This procedure does not only satisfy the variability of the highly customized SoC designs, but also achieve significant performance improvement. The objective is to improve both communication latency and power consumption under several 3D constraints. A Genetic Algorithm (GA) based efficient algorithm is applied to optimize both the topology and floorplan. Numerical experiments are implemented on standard benchmarks by comparing the method application in 3D architectures with the 2D designs and then comparing the architecture obtained by our proposed algorithm with both classical topologies and custom based topologies. The experimental results show that the architectures by our design algorithm can achieve more performance improvement than other algorithms and the proposed algorithm also proves to be a time efficient method for exploration in the large solution space.

Keywords: 3D NoC, Genetic Algorithm, topology, floorplan

1. Introduction

As the on-chip technology achieves significant development in recent years, rapid rise in interconnects delay and power consumption due to smaller wire cross-section, tighter wire pitch, and longer lines that traverse across larger chips is severely limiting IC performance enhancement in current and future nodes. 3D integration with multiple active Si layers stacked vertically is a promising method to overcome this scaling barrier as it replaces long inter-block global wires with much shorter vertical inter-layer interconnects [1]. As a main bottleneck for the high performance IC integration, the global interconnect technique usually plays a critical role in the on-chip systems. The application of Network-on-Chip (NoC) models for 3D systems instead of bus architectures proves to be a much more scalable and efficient solution to the global communication in the interconnect designs. The on-chip networks should have low communication latency and low power consumption, and could be designed for particular application traffic characteristics. The basic architecture design including topology and floorplan for the 3D NoC synthesis always performs as a fundamental factor in the whole network performance enhancement. As for the topology design, some classical topologies such as mesh, Butterfly Fat Tree, torus etc. have taken on a lot of drawbacks due to their limitations and inflexibilities in the highly customized 3D systems, and the floorplan optimization for further performance improvement can also fail to be achieved.

In this work, we develop a new procedure for the 3D NoC architecture optimization which not only satisfies the variability of the customized 3D integration system designs, but also achieves significant performance improvement. A series of efficient algorithms are used to explore the minimum cost topology and optimized floorplan in order to decrease the power consumption and communication latency under the hardware and software constraints. In the customized topology design process, the number and size of switches are determined and physical links (including both vertical and horizontal links) between switches and cores are established through a three-step progressive optimization. In the multi-layer floorplan optimization process, switches in each layer are allocated and the definite positions for each switch are obtained as a result. The both optimization processes of topology and floorplan are interrelated and interact on each other, therefore we design a method for simultaneously considering the linkage and allocation of the switches. The linkage results are automatically adjusted according to the locations of switches, and the determination of locations is based on the establishment of the linkage. Since the architecture optimization process is NP-hard, a GA based heuristic algorithm is applied for efficient and deep search in the extremely large solution space. A combination of different representation schemes is applied in GA solution process and a 3D performance model involving both optimization factors is adopted for evaluation. Experimental results show that our proposed method is efficient for the high performance and low power 3D NoC architecture optimization, which is adaptive for different 3D integration systems.

This paper is organized as follows: Section 2 covers previous research related to the study. Section 3 details the design algo-

[1] Graduate School of Information, Productions and Systems, Waseda University, Kitakyushu, Fukuoka 808–0135, Japan
[a)] jiangxin@ruri.waseda.jp

rithms used to achieve the objective. Section 4 illustrates the experimental analysis. Finally, Section 5 concludes this paper.

2. Related Works

The role of 3D IC integration in ensuring performance growth has become increasingly important, which has attracted more and more researches in the recent years. Potential and limits of 3D integration were quantified in Ref. [2] by analyzing the theoretical performance of various 2D and 3D topologies. Reference [3] explored a 3D IC model for constructing multi-layer blocks and evaluated the performance improvement for the vertical integration. Different representation methods were proposed in Refs. [4], [5], [6] to solve the 3D floorplan problem. In Ref. [7], thermal-driven design flows including 3D routing algorithms were implemented in 3D ICs.

Recent researchers have paid much attention to the 3D NoCs designs in the communication interconnect of 3D integration. In Ref. [8], opportunities and challenges associated with 3D NoCs, nanophotonic communication, and wireless interconnects were outline and compared. In Ref. [9], the performance of 3D NoC architectures were evaluated and the functionality of in terms of throughput, latency, energy dissipation and wiring area overhead compared to traditional 2D implementations were demonstrated. Problems relating to standard topologies with distinctive characters in NoCs were instructed in Ref. [10]. Mapping and placement of cores onto standard NoC topologies has been explored in Refs. [11], [12], [13], [14]. Exploration of irregular application specific topologies designs were researched in Refs. [15], [16] and [22], and different algorithms were used for the synthesis procedure. In Ref. [22], the first stage for establishing the core to switch connectivity is based on min-cut partition, which doesn't execute a complete search for the total solution space, and the switch position and switch connectivity are optimized separately. Since the optimizations of topology and floorplan are interrelated on each other, the two procedures should be considered simultaneously. In this paper, we also explore an irregular application specific 3D NoC design by developing an efficient method considering both optimization processes simultaneously.

3. Design Algorithms

The system architecture design is illustrated in **Fig. 1**. In this system, we input communication parameter files, core floorplan files, mathematical model, and the corresponding parameters. The communication parameter files describe the application bandwidth, latency, message types etc.. The core floorplan files describe the core locations and core communication graph. Given the input data, the system executes the encoding, perturbation, decoding, and selection as its optimization procedure. Finally the system generates the optimized architecture including topology and floorplan for each layer in the 3D system.

We adopt progressive three optimization steps and apply different algorithms in each step. The first step is aiming at optimizing the flow paths and deleting the redundant links in general. The second step is to further trim the network by using the minimum links. The third step intends to reconstruct the network by combining some nodes and reforming some new nodes, which needs

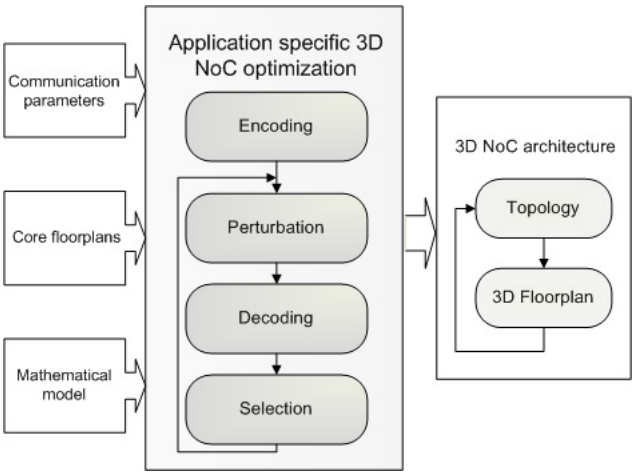


Fig. 1 Application specific 3D NoC system design flow.

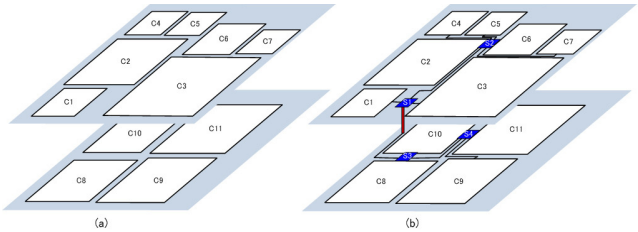


Fig. 2 Example of 3D NoC optimization results.

deep search of the whole network. Firstly, we assign a switch for each core to formulate a regular mesh and then apply the shortest path algorithm to search the minimum cost paths for the entire network, based on which, physical links between cores and switches are reconstructed to decrease the cost. In the second step, we implement the minimum spanning tree algorithm in the reconstructed network to decrease the cost by further trimming the redundant connections between the network nodes. In the last step, different switches are merged together to improve the performance, and the optimized floorplan in each layer are determined. We apply a Genetic Algorithm based algorithm to search the optimal merging way and definite positions for the switches in each layer. An example of the optimization result is illustrated in **Fig. 2**. Figure 2 (a) is the input core floorplan in two layers, and (b) is the optimized application specific NoC architecture for (a). As shown in Fig. 2 (b), the linkage between cores and switches and the switch positions in each layer are finally determined. The reason why we adopt a customized architecture is as follows. If we use a traditional regular topology like 3D mesh, every core will need a switch, and more physical links will also be added. By using our customized architecture instead of the above classical regular topologies, switches and links may be saved, so that system performance including power and latency can be improved greatly.

3.1 Preliminaries

The methodology and algorithms are designed based on the graph. We use the core communication graph as the input of our system, representing the packets transmission relations between any two cores.

Definition 1 The core communication graph $CCG(V, E)$ is a

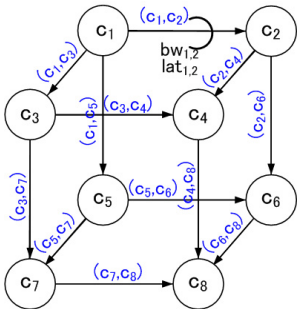


Fig. 3 Core communication graph CCG.

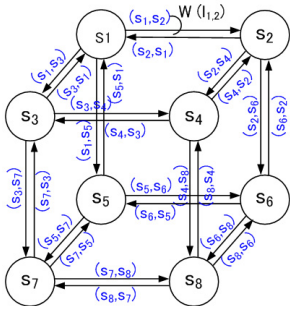


Fig. 4 Switch cost graph SCG.

directed graph, with each vertex $c_i \in V$ representing a core and the directed edge $(c_i, c_j) \in E$ representing the communication from the core c_i to c_j. The bandwidth of traffic flow from core c_i to c_j is represented by $bw_{i,j}$ and the latency for the flow is represented by $lat_{i,j}$. An example of the CCG for 3D mesh topology is illustrated in **Fig. 3**.

We start our algorithm by initially assigning a switch for each core, and the core communication graph can be transformed to switch cost graph. We use this graph as the foundation of subsequent calculation and analysis.

Definition 2 The switch cost graph $SCG(S, L)$ is a directed graph, with each vertex $s_i \in S$ representing a switch and the directed edge $l_{i,j} = (s_i, s_j) \in L$ representing the connection from the switch s_i to s_j. A weight $w(l_{i,j})$ is attached to each edge representing the cost of establishing the switch link. An example of the SCG for 3D mesh topology is illustrated in **Fig. 4**.

The cost $w(l_{i,j})$ on each link is set to be the combination of latency and power consumption of the traffic flow from switch s_i to s_j, which depends on the size of switches, the connectivity of the switches to other switches, and the length of physical links. The parameter definitions of the cost functions are listed in **Table 1**. The cost function for establishing a physical link between s_i and s_j can be calculated as:

$$w(l_{i,j}) = \alpha * p_{i,j} + (1 - \alpha) * t_{i,j} \tag{1}$$

Where $p_{i,j}$ is the power consumption from s_i to s_j ($p_{i,j} = p_i + p_j$), $t_{i,j}$ is the latency from s_i to s_j ($t_{i,j} = t_i + t_j$), and α is a weight parameter. p_i (p_j) is a power consumption at switch node s_i (s_i) and t_i (t_j) is a latency at s_i (s_i). They are calculated by the following equations:

$$p_i = p_{si} + p_{hi} + p_{vi} \tag{2}$$

$$p_{si} = p_{dsi} + p_{ssi} + p_{lsi} \tag{3}$$

Table 1 Notations of the model.

p_{si}	The power of crossbar switch i
p_{hi}	The power of horizontal channel i
p_{vi}	The power of vertical channel i
p_{dsi}	The dynamic power of crossbar switch i
p_{ssi}	The short-circuit power of crossbar switch i
p_{lsi}	The leakage power of crossbar switch i
t_{ai}	The arbitration logic delay of switch i
t_{si}	The switch delay of switch i
t_{hi}	The delay of horizontal channel i
t_{vi}	The delay of vertical channel i
N_F	The number of flows in CCG

$$p_{hi} = p_{dhi} + p_{shi} + p_{lhi} \tag{4}$$

$$p_{vi} = p_{dvi} + p_{svi} + p_{lvi} \tag{5}$$

$$t_i = t_{ai} + t_{si} + t_{hi} + t_{vi} \tag{6}$$

The total power P and latency T in the whole network can be represented by:

$$P = \sum_{j=1}^{N_F} \left(\sum p_{si}|_{all\ switch\ i\ in\ flow\ j} + \sum p_{hi}|_{all\ horizontal\ link\ i\ in\ flow\ j} + \sum p_{vi}|_{all\ vertical\ link\ i\ in\ flow\ j} \right) \tag{7}$$

$$T = \max_{j=1,2,...,N_F} \left(\sum (t_{ai} + t_{si})|_{all\ switch\ i\ in\ flow\ j} + \sum t_{hi}|_{all\ horizontal\ link\ i\ in\ flow\ j} + \sum t_{vi}|_{all\ vertical\ link\ i\ in\ flow\ j} \right) \tag{8}$$

From the above definitions and functions, we can formulate our cost model as:
Minimize

$$\alpha * P + (1 - \alpha) * T \tag{9}$$

Subject to

$$n_i \le N_{\max\_sw}, \forall i = 1, 2, ..., |S| \tag{10}$$

$$N_{3D} \le N_{\max\_ill} \tag{11}$$

Where n_i represents the number of ports of switch i, $N_{\max\_sw}$ is the maximum size of a switch and $N_{\max\_ill}$ is the maximum number of vertical links. The objective is to improve both power consumption and latency. The parameter α can be used to make trade-offs between power and latency. Equation (10) ensures the size of each switch will not surpass the maximum size according to the frequency, and Eq. (11) ensures that the total number of vertical links will not surpass the maximum vertical links constraints.

In the initialization step, we input the communication parameters, such as the application bandwidth, latency, message types etc.. The system will then generate the corresponding SCG according to the input CCG. The weight in each edge of SCG $w(l_{i,j})$(can be seemed as item cost) is calculated by using the cost function (1). The total cost for every flow in CCG which is calculated by Eq. (9) is the summation of each item cost. It can be used for cost evaluation in every optimization algorithm.

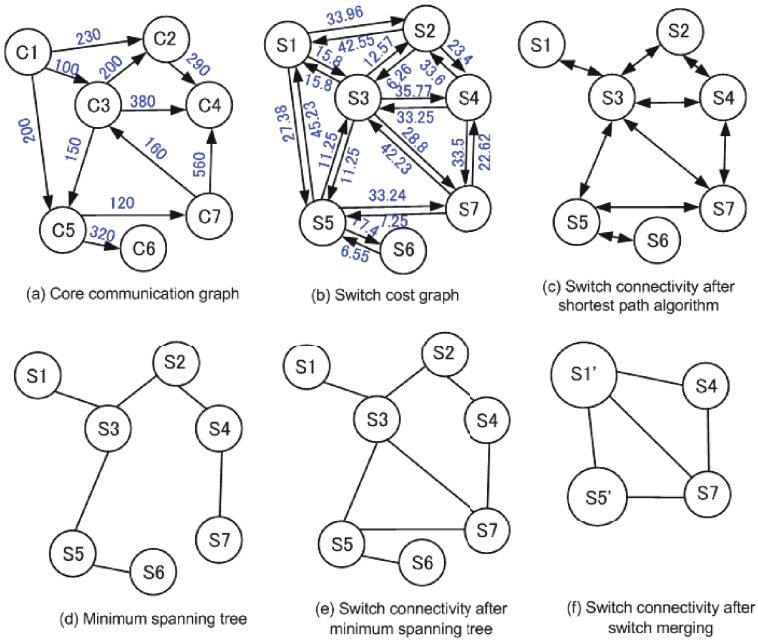


Fig. 5 Example of the design algorithm.

3.2 Overview of the Design Algorithms

The overview of the design algorithms are detailed in Algorithm 1. An example of the illustration for the progressive optimization procedure is shown in **Fig. 5**. In the initialization, each core is assigned a switch, and we can get SCG from the input CCG (Fig. 5 (a)) and cost computation functions (shown in Fig. 5 (b)). In Fig. 5 (a), the weight in each edge represents the bandwidth between the core pairs. Let's derive $w(l_{1,2})$ and $w(l_{2,1})$ as an example. The parameters and $bw_{1,2} = 230$ are firstly input to Eqs. (3), (4), (5) and (2) to get $p_1 = 28.23$ and $p_2 = 25.47$, and then input to Eq. (6) to get $t_1 = 8.58$ and $t_2 = 5.64$. Please refer Refs. [1], [10] and [20] for detailed calculation, but it is omitted here due to the paper length limitation. We set $\alpha = 0.5$, and input the resulting $p_{1,2}$ and $t_{1,2}$ to Eq. (1) to calculate $w(l_{1,2}) = 33.96$. To calculate $w(l_{2,1})$, since there is no traffic flow from C_2 to C_1, we assign $bw_{2,1} = 1$, so that $w(l_{2,1}) = 42.55$. SCG is a mesh architecture which is bidirectional with channels in both directions between connected nodes. However, in the application specific network, not all the channels are actually in use for message transmission. Deleting additional links which take a great impact on the switch size is a major factor for decreasing the interconnect cost. Therefore, we are focusing on constructing the least cost architecture by limiting the least cost directed path for every source and destination node of the flows in CCG. At the first step, we apply Dijkstra's algorithm [17] to find the minimal cost path for each flow in CCG. For each pair of nodes in the network, we delete the links that take higher cost but do not influence the required data flow transmission, and the minimum cost paths are reserved to formulate the new switch connectivity graph ($NSCG$) (shown in Fig. 5 (c)). For example, according to the shortest path calculation, for the flow from s_1 to s_2, the cost of path $\{s_1, s_3, s_2\}$ is smaller than that of $\{s_1, s_2\}$. Then we select the minimum cost path s_1, s_3 and s_2 and delete the high cost link (s_1, s_2). After shortest path selection, there still exist links that can be removed for further optimization. Take Fig. 5 (c) for example, the link (s_3, s_4) is on a minimal cost path between s_3 and s_4, but we can find another alternative path $\{s_3, s_2, s_4\}$ or $\{s_3, s_7, s_4\}$ in the current topology. So this link (s_3, s_4) can be removed. For the deep exploration, we try to find the minimum spanning tree (shown in Fig. 5 (d)) by use of Prim's algorithm [18] and then project it back to the required data flow in CCG. By this projection to CCG, the links necessary for flow transmission but not in the spanning tree are reconstructed in the switch connectivity, such as the link (s_3, s_7) and (s_5, s_7) in Fig. 5 (e). This process is the second step. As a result, the switch connectivity with the minimum communication cost is obtained (shown in Fig. 5 (e)). In the third step, we apply a GA based algorithm to further optimize the topology by merging the appropriate switches together and allocate each merged switch in the floorplan. Finally we output the optimal customized 3D architecture (shown in Fig. 5 (f)).

Let m and n be the number of edges and nodes in the network re-

Algorithm 1 3D NoC architecture optimization
Input: $CCG(V, E)$, $SCG(S, L)$, cost function, library of network components
Output: optimized network topology and floorplan
begin
  SCG = InitialSwitchalLocation (CCG)
  cost = cost function (SCG)
  for i = 1 to number of flows in CCG **do**
    ShortestPath (SCG, cost)
  end for
  delete the paths with higher cost
  update the links and weights in SCG
  $NSCG$ = ConstructNewGraph (SCG)
  MST = MiniSpanningTree ($NSCG$, cost)
  project MST to CCG
  delete unnecessary links according to MST
  update the links and weights in NSCG
  $NSCG'$ = ConstructNewGraph ($NSCG$)
  Switch Merging & Allocation ($NSCG'$, cost)
  update the nodes and links in $NSCG'$
  output the best network topology and floorplan
end

Algorithm 2 Switch Merging & Allocation by GA
Input: $NSCG'(S, L)$, cost function, GA parameters
Output: optimized network topology and floorplan
begin
  $t \leftarrow 0$
  initialize P_t by encoding method
  evaluate P_t by decoding method
  while (not $gen = Maxgen$) **do**
    create $C(t)$ from $P(t)$ by crossover routine
    create $C(t)$ from $P(t)$ by mutation routine
    evaluate $C(t)$ by decoding routine using (9)
as the evaluation function
    select $P(t+1)$ from $P(t)$ and $C(t)$ by using
Tournament Selection
    $t \leftarrow t + 1$
  end
  output the best network topology
end

spectively. The computational complexity of Dijkstra's algorithm and Prim's algorithm is $O(n^2)$ in worst case, and the complexity of GA is $O(m * n)$.

3.3 GA Based Optimization

The switch merging and allocation problem is NP-hard [22], and the searching space for this problem is extra-large, which makes it intractable for deterministic algorithms. In this step, we design a GA based heuristic algorithm for efficient and deep search. The details of application of GA are shown in Algorithm 2, where P_t is the set of parents at the t-th generation for switch merging and allocation, $C(t)$ is the offspring in current generation t, and gen and $Maxgen$ be the current generation and the maximum generation respectively. In the switch merging part, we assume that only the switches on the same layer can be merged, so the algorithm implements every search cycle in each layer separately. The switch allocation is based on the result of switch merging and the encoding procedure is implemented after the encoding of the merging part. We use the function from the cost model to simultaneously evaluate the results from switch merging and allocation in the decoding procedure. The output of the algorithm is the optimized 3D architecture including both the topology and floorplan.

3.3.1 GA Representation

We use two kinds of chromosomes to represent the switch merging way and the positions for merged switches respectively. For switch merging, we design a two-section chromosome: Section 1 is a series of random numbers from 1 to the number of switches, representing the switches for merging; Section 2 consists of random numbers from 0 to the number of unmerged switches, representing which switches are merged together. The switches are partitioned into several groups according to the representing numbers in Section 2. When we generate an item, the number of unmerged switches is decreased correspondingly. **Figure 6** illustrates an example of the representation method for switch merging. In this example, there are 5 switches, and the series number in Section 1 means switch 1, 4, 3, 2 and 5. In Section 2, we firstly generate 2 from 1 to 5, which means the first two switches (switch 1 and 4) are merged together. Then the number of unmerged switches is decreased to 3, and we generate 2 from 1 to 3, meaning the next two switches (switch 3 and 2) are

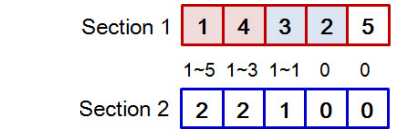


Fig. 6 Example of chromosome for switch merging.

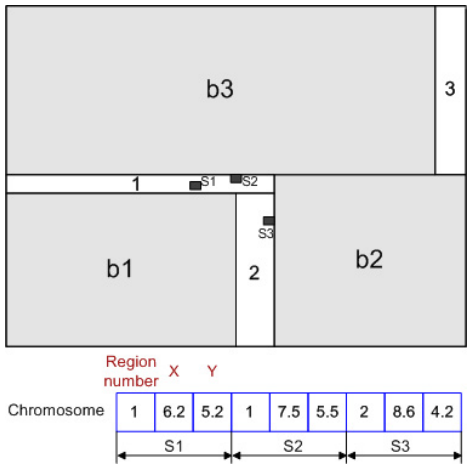


Fig. 7 Example of switch allocation and its chromosome.

merged. The third item equals to 1, representing switch 5 is in a separate group, without being merged with others. At this time, the number of unmerged switches is decreased to 0, which means the switch merging process has been completed. The remaining items are all set to 0. From this chromosome, we get the solution as "switch 1 and 4, switch 3 and 2 are merged respectively."

Chromosomes for switch allocation are generated after the merging chromosomes, and the lengths are determined according to the number of switches after merging. The optional areas for allocation are partitioned into several rectangle regions. These regions are defined by the coordinates from the lower left to the upper right, and they are numbered in sequence from left to right. We use the region number, x-coordinate, and y-coordinate to represent one optional position for the merged switch as illustrated in **Fig. 7**. The x-coordinate and y-coordinate are generated randomly in the available range of the corresponding region. In this example, there are 3 merged switches and 3 regions for switch allocation (1, 2, and 3 in this figure). We in turn generate the region number, x-coordinate, and y-coordinate for the 3 switches respectively and form a 9-item chromosome. The corresponding positions for each switch are shown in the floorplan.

3.3.2 GA Operators

We apply Insertion Mutation for Section 1 of the merging chromosome. For Section 2, we use Exchange Mutation in part of the population, and regenerating new individuals in the other part. Through this way, both the stability and diversity are maintained. Regenerate Probability is used to set the percentage for regenerating mutation. Order crossover is applied for Section 1 of the merging chromosome after mutation operator. An example of application of mutation and crossover for the merging chromosome is shown in **Fig. 8**.

For the allocation chromosome, we apply two different mutation methods. One is to fix the region number and change the x, y coordinate, the other is to regenerate a new chromosome. The po-

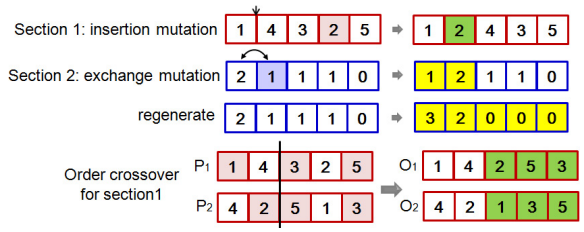


Fig. 8 Example of mutation and crossover for the merging chromosome.

sition points must be changed within the valid ranges of regions, and infeasible solutions are trimmed in the mutation procedure.

3.3.3 Evaluation and Selection

We utilize the cost model as the fitness function to evaluate the solutions from decoding both the merging and allocation chromosomes. We try to find the solution that consumes the minimum cost including both power and latency, and the solutions that violate the constraints are trimmed by being set to an extra large cost value. Tournament Selection is used to retain the best solution in every generation.

4. Numerical Experiments and Analysis

For the experiments, the NoC component library [19] is used. The parameter values for the power and latency models are determined by 45-nm technology low power libraries [20]. The vertical interconnects using TSVs are implemented based on the models in Ref. [21]. The proposed algorithms have been implemented in C# .Net language on a 2 GHz Windows workstation. The parameters for implementing the GA based algorithm are listed in **Table 2**.

We design three different experiments to test the effectiveness and efficiency of our proposed algorithm. In the first experiment, standard benchmarks on different scales are used to test the efficiency of our algorithm. In the second experiment, we make a comparison between the NoC application in 2D and 3D layout. In the third experiment, the effectiveness is tested by comparing the optimized architecture results through our proposed algorithm with the results by other algorithms.

4.1 Experimental Results on Standard Benchmarks

The proposed algorithm has been tested on three benchmarks on different scales: DSP Filter with 6 cores [23], MPEG4 Decoder with 12 cores [24] and a realistic multimedia SoC benchmark (D_26_media) with 26 cores [22]. The architecture is designed on to 3 layers in 3D, and the maximum number of vertical links, $N_{\max\_ill} = 25$. The weight parameter α between power and latency in Eq. (9) is initially set to 0.5. Then we regulate this parameter from 0–1, which means the preference is towards power or latency, and corresponding optimized topologies are different. For each α value, we normalize each power and latency to the average power and latency respectively and get the normalized value as the fitness value for GA optimization. We take 100 runs, and get the average best values as the final result. According to the results, we select the typical α values and list the implementation results in **Table 3**, and the final optimized architecture in each layer for application on MPEG4 Decoder with α equals to 0.5 is illustrated in **Fig. 9**. As there are various communication

Table 2 GA implementation parameters.

ID	Parameter	Value		
		Switch merge Section 1	Switch merge Section 2	Switch allocation
1	*Popsize*	200	200	200
2	*MaxGen*	1000	1000	1000
3	Crossover Probability	0.7	#	#
4	Mutation Probability	0.7	0.8	0.4
5	Regenerate Probability	#	0.6	0.5

flows and network scales, the performance improvement differs from benchmark to benchmark. As shown in Table 3, in each respective benchmark, the optimized number of switches almost remains the same with various α values, and this number is always smaller than the corresponding number of cores, which means that the application of regular topologies is not a good choice in irregular core architectures. For example, MPEG4 benchmark has 12 cores but 3 switches are enough in our proposed architecture. The optimized switch locations in each layer are distinct by changing the value of α, which means when the preference of cost is changing from power to latency, we will get different optimal floorplans even with the same topologies. From the execution time, we can find our proposed algorithm is very efficient even with large scale applications.

4.2 Comparison Between 2D and 3D Application

We implement 2-layer and 3-layer architecture in 3D application on the MPEG4 Decoder benchmark, and make comparison with the 2D architecture. In the 2D implementation, the same CCG is used as an input communication parameter, but placed in a 2D floorplan. The same algorithms are applied to the 2D without iteration in multi-layers, and we don't consider the vertical links constraints. The variation of performance values along with different switch counts in 3D and 2D implementations are shown in **Fig. 10**. From this figure we can see the best architecture with minimum power and latency in 3D application appears when the switch count equals to 3, and the 3D architecture outperforms the 2D counterpart in the evolving process. The average power and latency reduction evaluated by the cost function is 33.06% for 2-layer and 35.09% for 3-layer when comparing 3D to 2D implementations.

4.3 Comparison between Different Algorithms

We compare the application specific architecture generated by our algorithm implemented on D_26_media benchmark with the architecture on traditional 3D mesh topology and then with the results by using the algorithm in [22] and the results by Simulated Annealing (SA) algorithm. In the SA implementation, we use the same representation and evaluation as our GA. The perturbations are selected from the GA mutation, insertion mutation, exchange mutation and regenerate mutation. We use a temperature schedule of the form $T_k = \gamma T_{k-1}, k = 1, 2, 3, ...$The starting parameters T_0 and γ are set by taking account of input size and

Table 3 Implementation results on benchmarks.

Bench	Number of switches				Switch allocation in each layer (layer, x, y)				Cost $\alpha * P + (1-\alpha) * T$				Time (s)
	$\alpha = 0.5$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0.3$	
DSP	5	5	5	5	(0,185,117) (0,182,110) (1,161,136) (2,180,98) (2,179,65)	(0,167,116) (0,173,102) (1,166,130) (2,179,125) (2,167,106)	(0,137,111) (0, 121,104) (1,140,101) (2,154,127) (2,144,109)	(0,174,126) (0, 185,121) (1,185,111) (2,184,99) (2,185,114)	37.55	9.26	65.81	26.22	135
MPEG4	3	3	3	3	(0,94,64) (1, 92,66) (2,91,62)	(0,57,70) (1,54,71) (2,53,70)	(0,67,70) (1,59,68) (2,59,69)	(0,55,70) (1,63,64) (2,60,65)	45.45	11.18	79.73	31.75	544
D_26	4	4	3	4	(0,351,193) (1,436,2) (2,319,71) (2,314,85)	(0,15,113) (0,312,10) (1,218,5) (2,103,154)	(0,16,105) (1,3,171) (2,20,155)	(0,7,9) (1,105,20) (2,66,14) (2,12,137)	66.05	29.18	122.86	78.40	1456

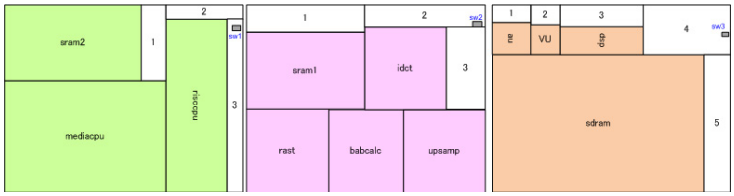


Fig. 9 Implementation results for 3 layers MEPG4 application.

Table 4 Comparison results between different algorithms.

Item	$\alpha = 0$		$\alpha = 1$		$\alpha = 0.3$		$\alpha = 0.5$		Time (s)
	Latency (ns)	Latency improved	Power (mw)	Power improved	Cost $\alpha * P + (1-\alpha) * T$	Cost improved	Cost $\alpha * P + (1-\alpha) * T$	Cost improved	
Mesh topology	33.8	#	245.2	#	97.22	#	139.5	#	#
Proposed algorithm	29.18	13.67%	122.86	49.89%	78.40	19.36%	66.05	52.65%	1456
Algorithm from Ref. [22]	30.23	10.56%	136.56	44.31%	81.94	15.72%	83.40	40.22%	3821
Algorithm by SA	28.78	14.85%	128.32	47.67%	82.76	14.87%	73.26	47.48%	1482

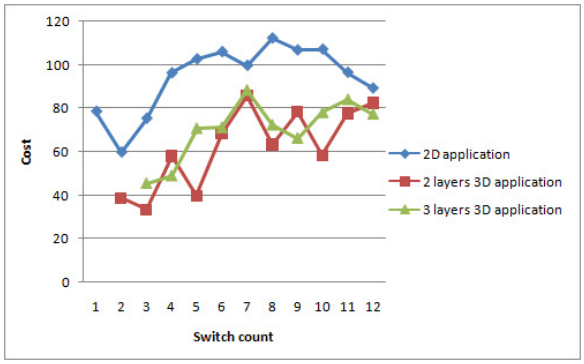


Fig. 10 Comparison results between 2D and 3D implementation.

extensive experimentation. In this experiment, we set $\gamma = 0.9$ and $T_0 = 20000$ comparing to the GA *MaxGen*. At each temperature we attempt the same times moves as the size of the population in our GA. The annealing process is halted when the temperature has been lowered 40 times since the last improvement was recorded in the best-so-far.The comparison results are reported in **Table 4**. When compared to the original 3D mesh topology in the 1'st row, if we simply consider the power in the cost function and ignore the impact of latency ($\alpha = 1$), we can find our custom based architecture can save the power by 49.89%. If the latency is concerned independently ($\alpha = 0$), we can see the latency reduction is 13.67%. If the power and latency are simultaneously taken into account, the cost improvement achieves 52.65% with $\alpha = 0.5$, and 19.36% with $\alpha = 0.3$. From the comparison results in Table 4, we can find our proposed algorithm is more power and latency effective. The algorithms in Ref. [22] include greedy search which is quite time consuming, and the time complexity is $O(|\ Z_{\max}\ ||\ V\ |^2|\ E\ ||\ \ln(|\ V\ |))$. Our proposed algorithm is based on heuristic searching, and we can see its time efficient from the execution time. When comparing with the architecture optimized by SA, in most cases our architecture results in better solutions by simultaneously optimizing the topology and location.

5. Conclusions

In this paper, we proposed an efficient algorithm for designing application specific irregular 3D NoC architectures. We adopt a three-step progressive optimization procedure to automatically explore the best solutions for improving power and latency in a large scale searching space. The numerical experiments show that custom based architecture can optimize the network performance, including power and latency. By comparing with 2D applications, the 3D system can improve the power and latency. When comparing with traditional 3D mesh and the architecture optimized by other algorithms, our proposed algorithm can achieve good implementing results within short execution time even with large scale applications.

Acknowledgments This research was partly supported by

JSPS KAKENHI 23500069 and Program for Fostering Regional Innovation by MEXT, Japan.

References

[1] Sheibanyrad, A., Ptrot, F. and Jantsch, A. (eds.): *3D Integration for NoC-based SoC Architectures*, Springer-Verlag (2010).

[2] Jantsch, A., Grange, M. and Pamunuwa, D.: The promises and limitations of 3-D integration, *Integrated Circuits and Systems*, pp.27–44 (2011).

[3] Pavlidis, V.F. and Friedman, E.G.: Via placement for minimum interconnect delay in three-dimensional (3D) circuits, *Proc. IEEE ISCAS*, pp.4587–4590 (May 2006).

[4] Ohta, H., Yamada, T., Kodama, C. and Fujiyosi, K.: The O-Sequence: Representation of 3D-floorplan dissected by rectangular walls, *Proc. Research in Microelectronics and Electronics*, pp.317–320 (2006).

[5] Berntsson, J. and Tang, M.: A slicing structure representation for the multi-layer floorplan layout problem, *Evolutionary Computing: Proc. EvoWorkshops 2004*, Vol.3005, pp.188–197 (2004).

[6] Hung, W.L., Link, G.M., Xie, Y., Vijaykrishnan, N. and Irwin, M.J.: Interconnect and thermal-aware floorplanning for 3D microprocessors, *Proc. the 7th International Symposium on Quality Electronic Design*, pp.98–104 (2006).

[7] Cong, J. and Zhang, Y.: Thermal via planning for 3-D ICs, *Proc. ICCAD 2005*, pp.745–752 (2005).

[8] Carloni, L.P., Pande, P. and Xie, Y.: Networks-on-chip in emerging interconnect paradigms: Advantages and challenges, *Proc. the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pp.93–102 (May 2009).

[9] Feero, B. and Pande, P.P.: Networks-on-chip in a three-dimensional environment: A performance evaluation, *IEEE Trans. Comput.*, pp.32–45 (Jan. 2009).

[10] Pavlidis, V.F. and Friedman, E.G.: 3-D topologies for networks-on-chip, *IEEE Trans. VLSI Systems*, pp.1081–1090 (Oct. 2007).

[11] Hu, J. and Marculescu, R.: Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures, *Proc. DATE*, pp.688–693 (2003).

[12] Murali, S. and Micheli, G.D.: SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs, *Proc. 41st annual DAC*, pp.7–11 (2004).

[13] Murali, S., and De Micheli, G.: Bandwidth Constrained Mapping of Cores onto NoC Architectures, *Proc. DATE*, Vol.2, pp.20896 (2004).

[14] Bertozzi, D. et al.: NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip, *IEEE Trans. Parallel and Distributed Systems*, pp.113–129 (Feb. 2005).

[15] Murali, S., Seiculescu, C., Benini, L. and Micheli, G.D.: Synthesis of networks on chips for 3D systems on chips, *Proc. 2009 ASPDAC*, pp.242–247 (2009).

[16] Yan, S. and Lin, B.: Custom networks-on-chip architectures with multicast routing, *IEEE Trans. VLSI Systems*, pp.342–355 (2009).

[17] Johnson, D.B.: A note on Dijkstra's shortest path algorithm, *J. ACM*, Vol.20, No.3, pp.385–388 (July 1973).

[18] Kershenbaum, A. and Slyke, R.V.: Computing minimum spanning trees efficiently, *ACM '72 Proc. ACM annual conf.*, Vol.1, pp.518–521 (1972).

[19] Stergiou, S. et al.: X pipes lite: A synthesis oriented design library for networks on chips, *Proc. DATE*, Vol.2, pp.1188–1193 (2005).

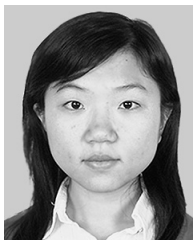
[20] Pavlidis, V.F. et al.: Via placement for minimum interconnect delay in three-dimensional (3D) circuits, *IEEE International System-on-Chip Conf.*, pp.4587–4590 (2006).

[21] Loi, I., Angiolini, F. and Benini, L.: Supporting vertical links for 3D networks-on-chip: toward an automated design and analysis flow, *Nano-Net '07 Proc. 2nd international Conf. on Nano-Networks*, (Sep. 2007).

[22] Seiculescu, C., Murali, S., Benini, L. and Micheli, G.De: SunFloor 3D: a tool for networks on chip topology synthesis for 3D systems on chips, *Proc. DATE*, pp.9–14 (April 2009).

[23] Jalabert, A., Murali, S., Benini, L. and Micheli, G.De: X pipesCompiler: A tool for instantiating application specific Networks on Chip, *Proc. DATE*, Vol.2, pp.884–889 (Feb. 2004).

[24] Gebhardt, D., You, J. and Stevens, K.S.: Link pipelining strategies for an application-specific asynchronous NoC, *Proc. 2011 5th IEEE/ACM International Symposium on NoCS*, pp.185–192 (May 2011).



Xin Jiang was born in Liaoning, China on July, 1981. She received her B.E. degree in Electronic Engineering in 2004, from Shanghai Marine University. In 2006, she received her M.S. degree in Shanghai Marine University. She then joined Shanghai Easipass International Co. Ltd., where she worked as a software testing engineer from 2006 to 2008. She is currently working toward a Dr. Eng. degree in Graduate School of Information, Productions and Systems, from Waseda University. Her current research interests include optimization algorithms in Electronics DA designs. She is a member of IEICE.

Ran Zhang was born in Dalian, China on April, 1985. She received her B.E. degree in Electronic Engineering in 2008, from Dalian University of Technology. In 2010, she received her M.S. degree in Graduate School of Information, Productions and Systems, from Waseda University. She is currently working toward a Dr. Eng. degree in Waseda University. Her current research is optimization algorithms for Electronics DA designs. She is a member of IEICE.

Takahiro Watanabe was born in Ube, Japan on October, 1950. He received his B.E. and M.E. in Electrical Engineering from Yamaguchi University, and Dr. Eng. from Tohoku University. In 1979, he joined Research and Development Center of TOSHIBA Corp., where he worked in the field of LSI design automation. In August 1990, he joined Yamaguchi University, the Department of Computer Science and Systems Engineering, and in April 2003, he moved to Waseda University, Graduate School of Information, Production and Systems. His current research interests are EDA algorithm, Microprocessor and MPSoC, NoC, FPGA and their applications. He is a member of IEICE, IPSJ and IEEE.

(Recommended by Associate Editor: *Yasuhiro Takashima*)