Design Space Exploration for NoC-Style Bus Networks

Jin-Sung Kim and Jaesung Lee

With the number of IP cores in a multicore system-on-chip increasing to up to tens or hundreds, the role of on-chip interconnection networks is vital. We propose a networks-on-chip-style bus network as a compromise and redefine the exploration problem to find the best IP tiling patterns and communication path combinations. Before solving the problem, we estimate the time complexity and validate the infeasibility of the solution. To reduce the time complexity, we propose two fast exploration algorithms and develop a program to implement these algorithms. The program is executed for several experiments, and the exploration time is reduced to approximately 1/22 and 7/1,200 at the first and second steps of the exploration process, respectively. However, as a trade-off for the time saving, the time cost (TC) of the searched architecture is increased to up to 4.7% and 11.2%, respectively, at each step compared with that of the architecture obtained through full-case exploration. The reduction ratio can be decreased to 1/4,000 by simultaneously applying both the algorithms even though the resulting TC is increased to up to 13.1% when compared with that obtained through full-case exploration.

Keywords: VLSI, System-on-chip, On-chip bus, SNP, Bus matrix.

I. Introduction

With the number of IP cores in a multicore system-on-chip (SoC) increasing to up to tens or hundreds, the role of on-chip interconnection is now vital. Networks-on-chip (NoC) has evolved as a solution to a diverse set of challenges ranging from synchronization and long-distance communication in deep submicron technologies to system-level design methodologies necessary to effectively and safely design complex SoCs with a large number of IP cores [1]–[6].

However, the NoC architecture is still very expensive to adopt for real design process. It has numerous routing schemes and poses deadlock problems due to multiple communication routes. Therefore, switch components require large memory space to accommodate routing tables and buffer space for every port. The attached IP cores also require significant memory space to implement packetization facilities. Accordingly, the raw architecture workstation (RAW) prototype [6] allocated nearly 50% of the die area to its on-chip network.

Meanwhile, a wide range of multilayered bus architectures from a simple segmented bus to a bus matrix with numerous spread layers is also being studied. However, careless use of bus layers can result in excessive use of wires and low resource utilization [7]. For a full bus matrix, the same number of layers as the number of master IP cores is used. Namely, there exist as many bus channels as the number of masters multiplied by the number of slaves, thus leading to significant wire congestion.

As a compromise, we have devised an NoC-style bus network. As shown in Fig. 1(a), it has a mesh-type interconnected architecture similar to an NoC. However, it uses a bus protocol instead of a network protocol when the IP cores communicate with one another. Therefore, it can provide a bandwidth as wide as an NoC does without the wire congestion problem. In addition, neither a switch nor a buffer is

Manuscript received Jan. 20, 2016; revised Aug. 20, 2016; accepted Aug. 25, 2016.

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-H8601-16-1008) supervised by the IITP (Institute for Information & communications Technology Promotion).

Jin-Sung Kim (jinsungk@sunmoon.ac.kr) is with the Department of Electronic Engineering, Sun Moon University, Asan, Rep. of Korea.

Jaesung Lee (corresponding author, jaesung.lee@ut.ac.kr) is with the Department of Electronic Engineering, Korea National University of Transportation, Chungju, Rep. of Korea.

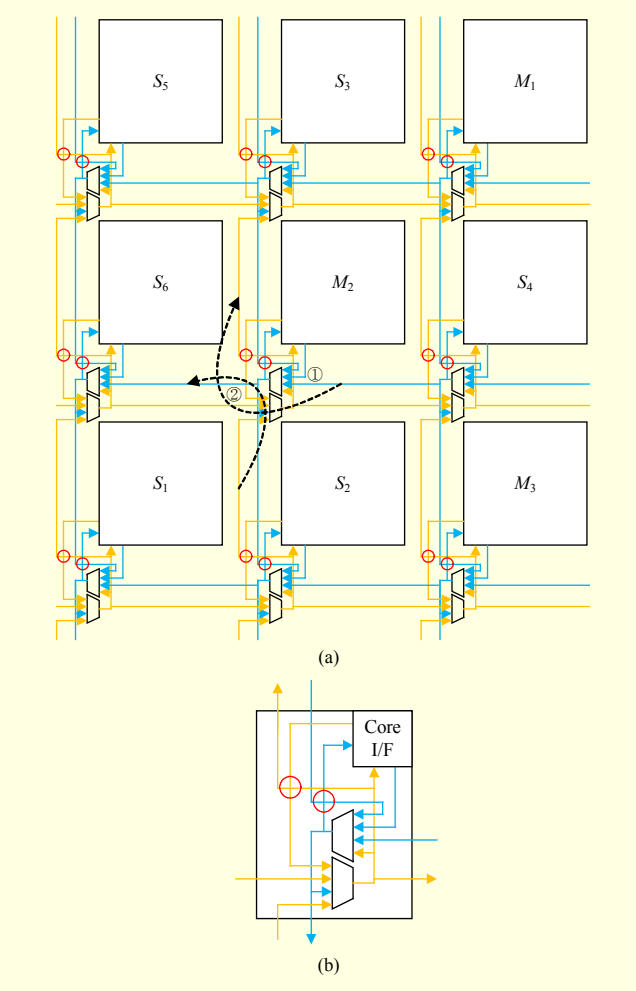

Fig. 1. Example of an NoC-style bus network: (a) mesh structure and (b) bridge component.

required because it is a bus interconnection; hence, it is not expensive.

In the bridge component of Fig. 1(b), the blue channel heads for the bottom or left-hand side, while the orange channel heads for the top or right-hand side. The transmitted data can change its channel color during data transfer through the two 4×1 multiplexers. Channel switching is allowed only when data is presented from the incoming port and is transmitted to the outgoing port. It should be noted that the two red circles indicate "no connection."

For interconnections between adjacent bridges, symmetric bus signals are chosen. Most of the on-chip buses proposed thus far separate their signals (including address, control, write data, and read data) from one another. Among these, only the read data signal carries the data from the slave IP cores to the master IP cores, while the other signals present data in the reverse direction. Therefore, the master cores must always be located at the top right-hand side corner in the mesh structure, and the slave cores must always be located at the bottom left-hand side corner. The advanced extensible interface (AXI) [8], which is one of the most popular bus protocols, still defines address, control, write data, and read data signals separately, as the legacy buses did. Likewise, it is impossible to build the NoC-style bus network and connect IP cores freely even with the latest popular bus standard.

Therefore, we adopted the SoC network protocol (SNP) [9] because it does not distinguish between a master and slave when interfacing them to a bus. Two sets of identical SNP signals (hereafter referred to as Up and Down channels denoted by orange and blue arrows, respectively, in Fig. 1) form symmetric communication paths for all the IP cores and bridges. The SNP then follows a phase-based approach and uses a unidirectional communication channel that is shared by address, control, and data information. Namely, all the information is transmitted sequentially in a time-multiplexed manner. Additional three-bit phase signals are used to distinguish the different types of information transmitted through the channel. Therefore, both the Up and Down channels can be used to write and read data. Accordingly, in contrast to existing buses, the proposed bus can connect an IP core to either the top or bottom side of the NoC-style structure whether it is a master or a slave.

If the bandwidth requirement for write-data transfer differs from that of read-data transfer, the SNP can improve its channel efficiency by placing each IP core at an advantageous point. In addition, if necessary, the NoC-style network additionally employs a buffer for register slicing or a synchronizer for globally asynchronous locally synchronous interfaces at the output of a multiplexer in a bridge.

The optimal design methodology for the bus network must be investigated. Design space exploration or systematic synthesis is necessary to minimize wastage of communication resources and efficiently utilize bus bandwidth. However, design space will be significant as the number of IP cores increases because there exists a geometric number of IP tiling combinations and communication routes between master and slave cores.

In this regard, we first define the exploration problem to consider all possible IP tiling combinations and all possible communication routes. Before solving the problem, we estimate the time complexity and validate its infeasibility. In order to reduce time complexity, we propose fast exploration algorithms and write a program to implement the algorithms. The program is executed for several examples, and the results (that is, IP tiling pattern and inter-IP communication route set) obtained using the proposed algorithms and through full-case exploration are compared.

The rest of the paper is organized as follows: Section II presents related works in the area of design automation for bus

architectures and NoCs. In Section III, we define the time cost (TC), and formulate the problem to find the best architecture. In Section IV, the scale of exploration space and time complexity required for exploring the space are discussed, and a fast search method to reduce time complexity is presented in Section V. Numerous examples and performance evaluation of architectures searched by the proposed method are discussed in Section VI. Finally, conclusions are drawn in Section VII.

II. Related Works

In the past, chip designers intuitively made a decision regarding the type of bus layer that must be used to connect an IP core. However, an optimal architecture cannot be achieved with that method. Such manual traversal of the vast design space cannot satisfy the requirements for best chip performance. After recognizing the importance of optimal architecture synthesis, automated on-chip bus design was widely studied. In the industry, FastForward for SiliconBackplane and Connection Kit for CoreFrame provide some guidelines for the designer to integrate IP cores more comfortably. In academia, Seceleanu and others [10] presented an optimal IP allocation method on segmented buses. They divided a single but long bus into numerous pieces and then examined where each IP core must be allocated to achieve the best performance. Hsieh and Pedram [11] conducted a similar study while focusing on low power rather than performance. They used the maximum matching algorithm to solve the hunting problem. However, both studies focus only on segmented bus architectures and apply old-fashioned communication protocols such as AMBA AHB. In the meantime, Pasricha and others proposed a synthesis methodology for optimal bus-matrix communication architectures using both AMBA AHB and AXI [12], [13]. They considered only a bus-matrix structure. Thereafter, Wang and others [14] proposed a synthesis scheme for multilayered on-chip buses and bus matrices to minimize power consumption using the same bus protocols. Some other studies considered only some specific architectures (or topologies) such as mezzanine, piggyback, and matrix using only AMBA [15]–[18].

Further, Ho and Pinkston [19] and Bertozzi and others [20] proposed some methods for optimizing NoC. Subsequently, the authors of [21]–[23] also conducted similar studies. However, the NoC synthesis methodologies are fundamentally different from those of bus architecture because the basic algorithm for NoC synthesis is constrained by the number of ports in a standard switch component and the overall performance is affected by packet routing algorithm rather than by design methodology, that is, IP tiling. Above all, the hardware cost of NoC is significantly higher than that of a bus-based SoC (generally 15 times more [24]). Nevertheless, they do not consider the cost issue as a significant one.

In summary, the previous studies related to bus synthesis consider only AHB- or AXI-type communications that involve asymmetric bus signals. Therefore, they examined only a few specific types of topologies to achieve optimal bus architectures. Lack of consideration for SNP ignores the NoC-style bus that can build a mesh structure. On the other hand, the previous studies on NoC synthesis are out of the scope of this study because they are fundamentally different from bus synthesis and too expensive to compare with the latter with the same criteria. Therefore, we must redefine and solve the exploration problem to optimally synthesize NoC-style on-chip bus networks, which have symmetric bus signals, using SNP.

III. Problem Statement

In this section, we redefine the design space exploration problem for designing NoC-style bus networks. First, the TC for data communication is formulated for a bus segment between two adjacent bridges in the mesh structure, and its value is calculated for every combination of IP tiling and every communication route (or path) between an arbitrary master and an arbitrary slave. The exploration problem is then presented at the end of this section.

1. Time Cost for Communication on a Bus Segment

In Fig. 2, the mesh structure of Fig. 1 is rotated counterclockwise by 45° and simplified with all the bridges hidden for convenience. As mentioned earlier, blue channels transfer data or commands from the top to bottom to initiate a bus transaction, while orange channels transfer data or commands from the bottom to top to initiate a bus transaction. During the transfer, the channel color can be changed by the two multiplexers inside a bridge. Namely, data or commands can change their channel from blue to orange, and vice versa, through the paths indicated by ① and ②, respectively, as shown in Fig. 1.

The mesh structure may consider different paths such as ①, ②, and ③ shown in Fig. 2 to transfer data from M_3 to S_6. For one architectural candidate, however, the corresponding bridges are synthesized to have fixed paths such that M_3 writes data to S_6 on only one path. Because the NoC-style bus network is not a real NoC but a pure bus (that is, not packet switched but circuit switched), it will use only one fixed route for data transfer between an arbitrary pair of master and slave. Here, only the shortest path is considered. If one bridge is

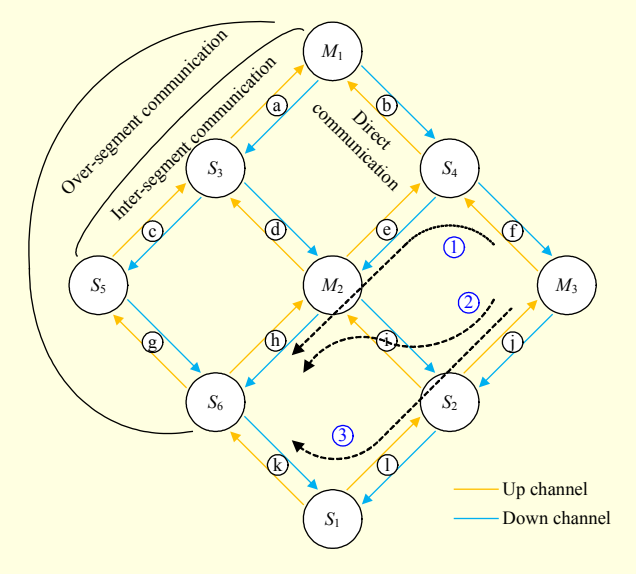

Fig. 2. Time cost consideration in a 3 × 3 mesh structure.

Table 1. Example write-/read-data communication matrix.

	M_1		M_2		M_3	
S_1	6	3	1	1	3	1
S_2	0	2	9	1	1	0
S_3	2	1	7	1	1	1
S_4	7	1	1	1	3	1
S_5	10	2	10	2	1	1
S_6	2	1	2	5	1	1

occupied, any other pairs of IP cores that must use that bridge to transfer their data cannot communicate with one another. Therefore, only paths that use the minimum number of bridges must be considered.

In [10], [11], and [12], the term communication traffic is used to represent the total number of data transfers between two IP cores. In these studies, the traffic characteristics of a given application are expressed in a tabular form, referred to as a communication matrix, that is filled with the amount of communication traffic between every pair of IPs. The term is often used along with some other terms. However, they fundamentally represent the same or similar properties, and any type of data related to traffic characteristics can be used as table values.

Let us suppose that an example application requires the communication traffic between each pair of IP cores, as shown in Table 1. In the table, masters and slaves are represented respectively by M and S with indices. For each master, there exist two columns in the table. The left-hand side column includes the amount of write-data traffic from the master to the slave, whereas the right-hand side column includes the amount of read-data traffic from the slave to the master.

For example, consider the communication TC between M_1 and S_4, which are directly connected to both sides of segment ⓑ in Fig. 2. The TC for M_1 to send (that is, write) data to S_4 through that segment is 7, whereas the TC for M_1 to receive (that is, read) data from S_4 is 1. Because write data and read data are managed independently, the representative TC is decided by the larger of the two costs.

Therefore, the TC on a segment g, $TC(g)$, is calculated as follows:

$$TC(g) = \max\left(tc_{\mathrm{W}}(g), tc_{\mathrm{R}}(g)\right), \tag{1}$$

where g is a unique number assigned to that segment, and the max function returns the larger one of two arguments.

However, not only M_1 and S_4 but other IPs may also use that segment. Let us assume that an architectural candidate, in which all communication routes are fixed, allows M_1 to communicate with S_1, S_2, and S_6 through segment ⓑ as well as with S_4. We must then consider inter-segment communication between M_1 and S_1, S_2, and S_6 as well as direct communication between M_1 and S_4.

Meanwhile, let us assume that an architectural candidate allows only segment ⓒ between S_3 and S_5 when communications occur between M_1 and S_6 or between M_1 and S_1 even though there is no direct communication between S_3 and S_5 through the segment. The total TC on that segment must then include the TC for over-segment communication as well as the TC for inter-segment communication through the segment (between S_5 and M_1, S_5 and M_2, and S_5 and M_3).

In this manner, the total TC at an arbitrary segment g is calculated considering the three factors as follows:

$$TC(g) = tc_{\mathrm{direct}}(g) + tc_{\mathrm{inter}}(g) + tc_{\mathrm{over}}(g). \tag{2}$$

In other words, for an arbitrary segment g, the communication cost $TC(g)$ includes the (1) direct communication cost that accounts for communication between two IPs connected directly to both sides of the segment g, (2) inter-segment communication cost that accounts for communication between an IP connected directly to the segment and the other IPs that are farther away from that segment by more than one hop but have to use the segment to communicate with that IP, and (3) over-segment communication that accounts for communication between two IPs whose communication path must pass through segment g even though they are all farther away from that segment by more than one hop. The concept of (2) is based on that of [10].

For write-data traffic from a master to a slave, the three components are expressed as follows:

$$
\begin{aligned}
tc_{\mathrm{W,direct}}(g) &= \sum_{C_{\mathrm{m}}(i)\ni g, C_{\mathrm{s}}(j)\ni g} W_{M_i,S_j},\\
tc_{\mathrm{W,inter}}(g) &= \sum_{C_{\mathrm{m}}(i)\ni g, C_{\mathrm{s}}(j)\not\ni g, P_W(i,j)\ni g} W_{M_i,S_j}\\
&\quad + \sum_{C_m(i)\not\ni g, C_{\mathrm{s}}(j)\ni g, P_W(i,j)\ni g} W_{M_i,S_j},\\
tc_{\mathrm{W,over}}(g) &= \sum_{C_{\mathrm{m}}(i)\not\ni g, C_{\mathrm{s}}(j)\not\ni g, P_W(i,j)\ni g} W_{M_i,S_j}.
\end{aligned} \tag{3}
$$

For read-data traffic from a slave to a master, the three components are expressed as given below.

$$
\begin{aligned}
tc_{\mathrm{R,direct}}(g) &= \sum_{C_{\mathrm{m}}(i)\ni g, C_{\mathrm{s}}(j)\ni g} R_{M_i,S_j},\\
tc_{\mathrm{R,inter}}(g) &= \sum_{C_{\mathrm{m}}(i)\ni g, C_{\mathrm{s}}(j)\not\ni g, P_R(i,j)\ni g} R_{M_i,S_j}\\
&\quad + \sum_{C_m(i)\not\ni g, C_{\mathrm{s}}(j)\ni g, P_R(i,j)\ni g} R_{M_i,S_j},\\
tc_{\mathrm{R,over}}(g) &= \sum_{C_{\mathrm{m}}(i)\not\ni g, C_{\mathrm{s}}(j)\not\ni g, P_R(i,j)\ni g} R_{M_i,S_j}.
\end{aligned} \tag{4}
$$

In these formulas, W_{M_i,S_j} and R_{M_i,S_j} represent the amounts of write-data and read-data traffic between the two IPs M_i and S_j, respectively. Equations (3) and (4) are established on the basis of analysis of [9].

Each summation of the three factors is performed depending on which segment each IP core belongs to. Let the function C be defined as a set function that returns a set of segment numbers where an IP core is connected directly. The functions C_{m} and C_{s} then represent the set functions for master and slave IP cores, respectively. An index i is used for masters (that is, $1 \le i \le$ total number of masters), and another index j is used for slaves (that is, $1 \le j \le$ total number of slaves). For example, M_2 is directly connected to four segments ⓓ, ⓔ, ⓗ, and ⓘ in Fig. 2; therefore, $C_{\mathrm{m}}(2)$ returns {ⓓ, ⓔ, ⓗ, ⓘ}.

As shown in (3) and (4), the direct communication cost can be easily calculated; however, the intra- and over-segment communication cost calculations are slightly complex. An additional function P (which is also a set function) is required to decide whether a pair of IP cores needs to use segment g for their communication. The function returns a set of segment numbers that the traffic between two IPs (i and j) must pass through. The subscripts W and R are defined for write traffic and read traffic, respectively: P_{W} and P_{R}. For example, if a master i with $C_{\mathrm{m}}(i) \not\ni g$ and a slave j with $C_{\mathrm{s}}(j) \not\ni g$ satisfy the condition $P_{\mathrm{R}}(i,j) \ni g$, then they must use segment g for their read-data communication. Therefore, their TC must be included in the formula for over-segment communication.

2. Exploration Problem

The design space exploration problem for synthesis of an NoC-style bus network is discussed in this subsection.

Problem 1: The exploration problem for an NoC-style bus network involves finding the set functions C and P such that the maximum among TC values for all bus segments g ($1 \le g \le s$; s is the total number of segments) is minimum.

It should be noted that the term 'architectural candidate' in this study refers to just one case in which one pattern of IP tiling is chosen among all possible IP tiling patterns (that is, the function C is defined once) and, for that pattern, the communication paths between every pair of master and slave are defined (that is, the function P is defined once). Therefore, a large number of candidates can exist as the number of IP cores increases.

IV. Exploration Space and Time Complexity

In this section, we present the exploration space that comprises the architectural candidates and time complexity to explore the space. Among all the possible IP tiling patterns, the exploration process searches for the best pattern whose maximum TC value is lowest. For better understanding, this section presents an example where the shape of layout is a square and the number of IP cores is n^2.

1. Design Space Exploration

First, the number of IP tiling patterns is given by the permutation $n^2!$ because it is the same as the number of ways of arranging n^2 IP cores in a row. The permutation must then be divided by four because the layouts when flipped up and down or turned left and right are all the same (see Fig. 2). Once one IP tiling pattern is obtained, the next step defines the communication paths for every master–slave pair in that pattern. The number of all possible communication paths for a pair is as follows. For example, the number of shortest paths from A to B in Fig. 3 is the combination ${}_5C_2$ (= $5!/(2!\times 3!)$) because it is the same as the number of ways of arranging a, a, a, b, and b in a row.

It should be noted that there exist numerous communication paths for another pair as well. Therefore, the combination values calculated for every pair must be multiplied one by one.

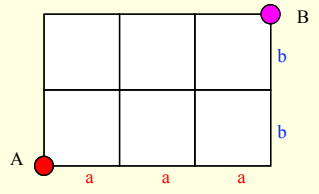

Fig. 3. Shortest communication paths between two cores A and B.

http://dx.doi.org/10.4218/etrij.16.0116.0037

Moreover, because the read-data and write-data paths will be managed separately, the number of multiplications can be doubled. We can see that the exploration space is very large. For each architectural candidate, the *TC* values at every segment must be calculated to find the maximum and then the maximum values of all the candidates must be sorted to find the best candidate.

2. Time Complexity

Next, we investigate the time complexity needed to explore the design space. The exploration program is written in C++ and executed on a Phenom II X4 945 processor-based desktop with 16 GB memory. First, the program tiles IP cores to create an IP tiling pattern, that is, it defines the functions C_m and C_s, and for that pattern, all possible communication paths between every master–slave pair are searched, that is, the functions P_W and P_R are defined for every case. The program repeats this procedure for all possible IP tiling patterns. After program runs on numerous examples, we found that approximately 1.5×10^7 candidates can be explored in 2 min on average. On the basis of this rate, the time complexity is estimated by increasing the number of IP cores, and the results are shown in Fig. 4. When the square root of the number of IPs is not an integer, rectangle type of mesh structures are used.

As shown in the figure, the time complexity increases geometrically as the number of IP cores increases because the graphs of mathematical permutation and combination functions increase as their arguments increase. The time complexity for the case of more than 16 IP cores is not shown in the graph because it is time consuming. For example, the time complexity is estimated to be 47 days and 218 days for 25 and 36 IPs, respectively. It should be noted that these long execution times are disqualified considering the modern chip design cycle and that type of exploration can be regarded as being impossible in the case of applications having more than 16 IP cores.

Consequently, a method must be developed to reduce the time complexity.

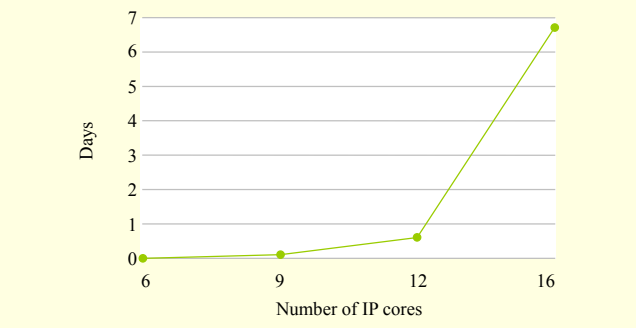

Fig. 4. Time complexity.

V. Fast Exploration Algorithms

As mentioned in the previous section, all possible architectural candidates cannot be explored to find the best architecture with the lowest max. *TC* within a reasonable period of time. Accordingly, we propose a heuristic method to reduce time complexity. Figure 5 shows the max. *TC* values calculated for all possible architectural candidates for nine IPs; the number of master and slave cores are four and five, respectively. The *x*-axis represents an ID number assigned to each architectural candidate, and the *y*-axis represents the calculated and obtained max. *TC* value for each candidate. The communication matrix for the master and slave cores is randomly generated following a standard normal distribution. The created mesh structure is small, and the layouts are similar whether they are flipped up or down or rotated left or right. In addition, the max. *TC* values are primarily decided by a few IP pairs that are far from one another. Therefore, there can be many cases that have the same or similar max. *TC* values.

From the figure, we can see that good candidates having small *TC* values occur when the masters and slaves are uniformly tiled mixed with each other. When plotting the *TC* values with numerous additional examples, the same phenomenon occurs because data transfer does not occur between two homogeneous IPs, that is, master-to-master or slave-to-slave. Therefore, data traffic cannot be distributed without a significant traffic jam. Hence, the exploration

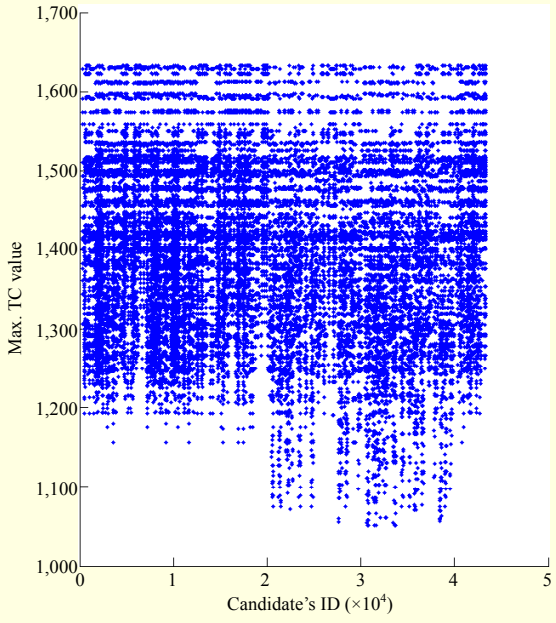

Fig. 5. Exploration of full cases.

program must focus on some patterns of IP tiling where the masters and slaves are uniformly tiled mixed.

In order to investigate the mixture type of patterns, the first step of exploration process involves the following algorithm.

Algorithm 1 (mixture-type first exploration).

For a given communication matrix table such as Table 1,

1. Among the different IP tiling patterns, find the patterns where, for many nodes (that is, vertices of the mesh graph in Fig. 2), many heterogeneous IP cores are connected to adjacent nodes (that is, up to max. 4 connected). This procedure is performed before actual IP tiling and by coloring the two colors (master and slave) such that the abovementioned conditions are satisfied.
2. Then, tile the IPs and find the best *TC* value using *Algorithm 2* below or full sets of communication paths, that is, all possible combinations of paths between every pair of IPs.
3. In order to change the current IP tiling pattern, swap two randomly selected homogeneous IPs, that is, the function *C* is modified.
4. Again, find the best *TC* value using *Algorithm 2* below or full sets of paths.
5. Repeat steps 3 and 4 for a certain period up to iteration bound $b1$ (from many experiments, the saturation of enhancement is found to occur near the point where $b1 = 45$).
6. If the current best *TC* for that period is enhanced when compared with the previous best *TC*, return to Step 5 and continue to examine. Otherwise, stop.

The pseudo-code for Algorithm 1 is as follows:

Mesh_structure[1.. $n_m + n_s$] : = COLORING(n_m, n_s); // n_m is the number of masters, n_s is the number of slaves
Function_C[1.. n_m, 1.. n_s] : = TILE_IP(n_m, n_s, *Mesh_structure*); //tiling IPs on the mesh generated
goto *Algirithm2* or *full_cases_of_paths*; // Algorithm 2 on or off
j: = 0;
while($i < b1$) { // $b1$ is iteration bound
    Function_C[1.. n_m, 1.. n_s] : = SWAP_IP(n_m, n_s, *Mesh_structure*);
    goto *Algirithm2* or *full_cases_of_paths*;
    j: = j + 1;}
return *Function_C*, *Function_P*;

For the second step, that is, the path selection step, the greedy approach is the most suitable. Initially, this method randomly selects one set of inter-IP paths among all possible sets of inter-IP paths. Thereafter, it calculates *TC* for each case by swapping the set with a different set one after the other. If the current *TC* is improved when compared with the previous value, then the *TC* value is updated with the current value. Otherwise, the *TC* for the next set is calculated. In this manner, the procedure continues until enhancement does not occur further within a certain iteration bound ($b2$). In order to change the initial random seed, the same procedure is repeated s times. Here, both $b2$ and s are user-defined parameters, and from many experiments, their conservative values are found to be approximately 50 and 200, respectively.

Algorithm 2 (greedy exploration with iteration bound).

1. First, randomly select a set of inter-IP communication paths.
2. Calculate max. *TC* for the architectural candidate.
3. In order to change the set of paths, again randomly select a new set of paths; that is, the function *P* is modified.
4. Calculate max. *TC* for the new case.
5. If the new *TC* is improved when compared with the current *TC*, update the current *TC* value with the new value.
6. Otherwise, go to Step 3.
7. Continue this loop as long as the *TC* is improved within the iteration bound $b2$.
8. In order to change the initial random seed, the whole procedure is repeated s times.

The pseudo-code for Algorithm 2 is as follows:

for (i = 1 **to** s) **do** { // s is the random seed
    Function_P[1.. n_m, 1.. n_s] : = SELECT_PATH_SET_RANDOMLY(n_m, n_s, *Mesh_structure*, s);
    TC : = CALCULATE_MAX_TC(*Function_C*, *Function_P*);
    j: = 0;
        while($i < b2$) { // $b2$ is iteration bound
            Function_P[1.. n_m, 1.. n_s] : = SELECT_PATH_SET_RANDOMLY(n_m, n_s, *Mesh_structure*, s);
            TC': = CALCULATE_MAX_TC(*Function_C*, *Function_P*);
            if (*TC'* < *TC*) *TC*: = *TC'*;
            else j: = j + 1;}

VI. Performance Estimation

The exploration program presented in Section IV-B is modified to employ the two fast exploration algorithms presented in Section V. In addition, for comparison with existing bus architectures, a multilayered bus synthesis algorithm is also implemented on the basis of [7], [10], [12], and [13].

The programs are then executed for some example communication matrixes. The prepared matrixes (tables such as Table 1) commonly assume 12 IP cores that consist of 6 masters and 6 slaves. A 4 × 3 mesh structure is built to tile these IP cores. The table values are randomly generated on the basis of uniform distribution for a set of tables and non-central F distribution [25] for another set of tables using the statistics toolbox MATLAB [26]. Synthetic values are used for the table values instead of real data because the authors cannot afford to implement a sufficiently wide range of real applications to avoid error of generalization. In other words, the possibility of error can significantly increase if values extracted from very few specific real applications are used. Therefore, quantitative analysis is reasonable because such an analysis can consider

different traffic patterns.

Specifically, 10 tables are prepared per set, and a total of 20 tables are prepared. For the first set of tables, the column values are generated on the basis of uniform distribution ranging from 0 to 200 with an average of 100. The generated values do not discriminate between write and read traffic. Uniform distribution is chosen to simulate uniform traffic. Meanwhile, for the second set, the write-traffic columns are filled with the values generated using non-central F distribution. They also range from 0 to 200, and their average is also approximately 100. The read-traffic columns are filled with the values generated on the basis of normal distribution to simulate asymmetric traffic. It should be noted that real applications are most likely to have asymmetric traffic rather than uniform traffic.

First, we synthesized the existing multilayered bus architecture having the lowest *TC* value as a reference for comparison. The synthesized multilayered bus is based on AMBA AXI [8] because the AXI is the de facto standard in this area.

Thereafter, the NoC-style bus networks are synthesized. Full-case exploration with tables is performed to obtain the ground truths, that is, to find the architectural candidate of NoC-style bus having the best (that is, lowest) *TC*. As mentioned earlier, it takes 2 min for approximately 1.5×10^7 cases. An example table creates almost the same number of cases. Namely, there exist exactly 14,735,542 architectural candidates with 12 IPs (6 masters and 6 slaves), and the measured time is approximately 2 min on average. Indeed, this is a huge burden. In order to reduce the number of candidates, the proposed heuristic algorithms are applied to the corresponding steps in the exploration process. First, only *Algorithm 1* (*mixture-type first exploration*) is applied to the first step (IP tiling), while nothing is applied for the second step (path defining). The iteration parameter $b1$ is set to be 45. The measured execution time of the program was reduced to approximately 1/22. However, as a trade-off for the time saving, the *TC* values of the searched architectures are up to 4.7% higher than those of architectures found through full-case exploration.

Second, *Algorithm 2* (*greedy exploration with iteration bound*) is applied to the second step of the exploration process, while nothing is applied for the first step. The iteration bound $b2$ and s are set to be 50 and 200, respectively. Through the experiment, we found that 1.5×10^7 cases were fully searched in approximately 0.7 s, when compared with the average time of 2 min. Therefore, the reduction ratio becomes 7/1,200. However, as a trade-off for the time saving, the *TC* values of the searched architectures are up to 11.2% higher than those of architectures found through full-case exploration, that is, the ground truths. Figure 6 shows these results. Figure 6(a) shows the resulting graph for uniform distribution tables, while Fig. 6(b) shows the resulting graph for non-central F distribution plus normal distribution tables. When compared with the existing multilayered buses based on the de facto standard, the performance of NoC-style buses is significantly enhanced for both types of traffic distributions, specifically they achieved approximately 26% enhancement.

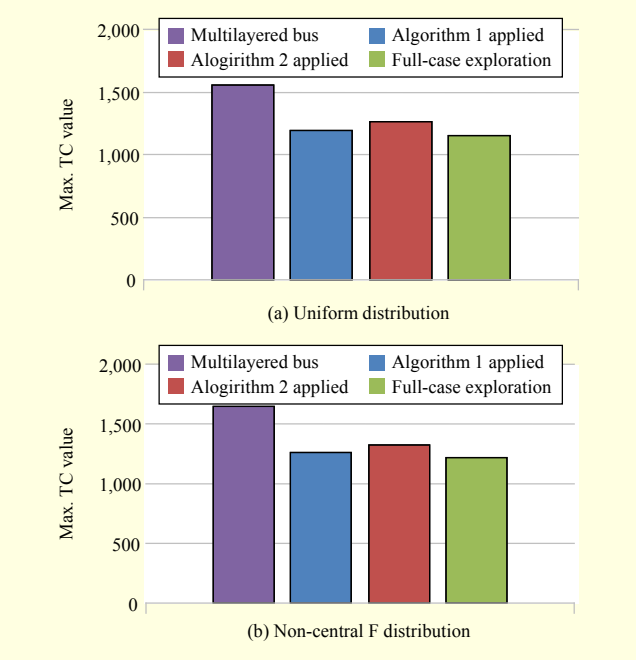

Fig. 6. Time cost with each applied algorithm: (a) uniform distribution and (b) non-central F distribution plus normal distribution.

Table 2. Reduction ratio of time complexity.

Algorithm / Algorithm-applied step	Mixture-type first exploration	Greedy exploration with iteration bound	Two algorithms applied at a time
IP tiling	$\frac{1}{22}$	N/A	$\frac{1}{3,200}$ – $\frac{1}{4,000}$
Communication path selection	N/A	$\frac{7}{1,200}$	

The exploration accuracy of *Algorithm 1* is relatively high even though it does not significantly save exploration time. Meanwhile, the time-saving effect of *Algorithm 2* is relatively good even though it is not very accurate. This phenomenon is consistent in both types of distributions.

The reduction ratios in the two steps are summarized in Table 2. Because the reduction in time complexity at each step is accumulated, the overall reduction ratio is the product of the two ratios. However, when the two algorithms are

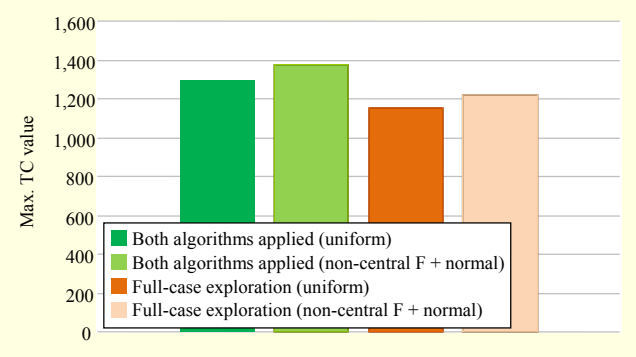

Fig. 7. Time cost with both algorithms applied simultaneously.

simultaneously applied, the total reduction ratio ranges from 1/3,200 to 1/4,000; the MS Windows on the desktop PC shows different execution times for the same program code even for the exactly same size tables. The running condition of the operating system has minimal effect. In addition, the resulting *TC* values are increased to up to 13.1% when compared with those from full-case exploration, as shown in Fig. 7. However, considering the total reduction in time complexity, it is a good trade-off for fast time-to-market.

As can be seen in Figs. 6 and 7, the searched TC values do not vary significantly when the same algorithms are applied, regardless of type of traffic distribution, that is uniform or non-uniform. Therefore, the exploration process finds the optimal point where an IP tiling pattern and communication paths are selected for the best performance by efficiently utilizing the overall network infrastructure.

Hence, the proposed algorithms can make it possible for the exploration program to cover up to 25 or 36 IP cores with 5 × 5 or 6 × 6 mesh structures within a reasonable time of approximately 25 min and 1 h 40 min, which are estimated to be 47 days and 218 days, respectively, as presented in Section IV-B when full-case exploration is assumed. Therefore, up to medium-scale SoCs can be rapidly designed with the proposed fast search algorithms.

VII. Conclusion

In this study, we redefined the design space exploration problem to find the best NoC-style bus network. All possible IP tiling patterns and combinations of inter-IP communication paths are considered. Further, the exploration space is investigated, and the time complexity to solve the problem is estimated. The space is found to be either permutational or combinatorial; therefore, the time consumed to explore increases geometrically as the number of IP cores tiled increases. Therefore, two fast exploration algorithms are proposed for each step of the exploration process to reduce time complexity.

A software program is written to implement the proposed algorithms, and it was executed for several experiments. By applying the algorithms, the exploration time is shortened to approximately 1/22 and 7/1,200 at the first and second step of the exploration process, respectively. However, as a trade-off for the time saving, the *TC* values of the searched architectures are up to 4.7% and 11.2%, respectively, higher than those of architectures found through full-case exploration. The reduction ratio can be reduced to 1/4,000 by simultaneously applying the two algorithms even though the resulting *TC* values are increased to up to 13.1% when compared with those for the full-case exploration case. Considering current chip design cycles, this appears to be a decent trade-off. Further, the exploration process found the optimal point where an IP tiling pattern and communication paths are selected to ensure best performance by effectively utilizing the overall network.

Therefore, SoCs having more than 36 IP cores can be designed within a reasonable time. However, considering recent large-scale SoC designs, this reduction is not sufficient because the number of IP cores that can be integrated into a single chip already exceeds a hundred and is continuously increasing proportional to the square. In the near future, we will consider probabilistic approaches for further reduction in time complexity.

References

[1] W. Dally, "Route Packets, not Wires: On-chip Interconnection Networks," *Proc. Design Autom. Conf.*, June 22, 2001, pp. 684–689.

[2] L. Benini and G. De Micheli, "Networks on Chips: a New SoC Paradigm," *Comput.*, vol. 35, no. 1, Jan. 2002, pp. 70–78.

[3] A. Jantsch and H. Tenhunen, "Networks on Chip," Boston, MA, USA: Kluwer Academic Publishers, 2003.

[4] W.J. Dally and B. Towles, "Principles and Practices of Interconnection Networks," New York, USA: Elsevier Science Publishers, 2003.

[5] G. De Micheli and L. Benini, "Networks on Chips: Technology and Tools," Los Altos, CA, USA: Morgan Kaufmann Publishers, 2006.

[6] T. Konstantakopoulos et al., "Energy Scalability of On-chip Interconnection Networks in Multicore Architectures," MIT CSAIL Technical Report, Nov. 2007.

[7] R. Wang et al., "Bus Matrix Synthesis based on Steiner Graphs for Power Efficient System-on-Chip Communications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, Feb. 2011, pp. 167–179.

[8] ARM, AMBA AXI protocol specification, 2003.

[9] J. Lee and H.-J. Lee, "Wire Optimization for Multimedia SoC

and SiP Designs," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 8, Sept. 2008, pp. 2202–2215.

[10] T. Seceleanu el al., "Resource Allocation Methodology for the Segmented Bus Platform," *IEEE Int. SOC Conf.*, Herndon, VA, USA, Sept. 25–28, 2005, pp. 129–132.

[11] C. Hsieh and M. Pedram, "Architectural Energy Optimization by Bus Splitting," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 4, Apr. 2002, pp. 408–414.

[12] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "BMSYN: Bus Matrix Communication Architecture Synthesis for MPSoC," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 8, Aug. 2007, pp. 1454–1464.

[13] S. Pasricha et al., "CAPPS: a Framework for Power–Performance Tradeoffs in Bus-Matrix-Based On-chip Communication Architecture Synthesis," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, no. 2, Feb. 2010, pp. 209–221.

[14] R. Wang et al., "Bus Matrix Synthesis based on Steiner Graphs for Power Efficient System-on-Chip Communications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, Feb. 2011, pp. 167–179.

[15] S. Sombatsiri et al., "An AMBA Hierarchical Shared Bus Architecture Design Space Exploration Method Considering Pipeline, Burst and Split Transaction," *Int. Conf. Electrical Eng. /Electron., Comput., Telecommun. Inform. Technol.*, Krabi, Thailnad, May 15–17, 2013, pp. 1–6.

[16] K. Rawat, K. Sahni, and S. Pandey, "RTL Implementation for AMBA ASB APB Protocol at System on Chip level," *Int. Conf. Signal Process. Integr. Netw.*, Noida, India, Feb. 19–20, 2015, pp. 927–930.

[17] K.E. Ahmed and M.M. Farag, "Enhanced Overloaded CDMA Interconnect (OCI) Bus Architecture for On-chip Communication," *IEEE Annu. Symp. High-Performance Interconnects*, Santa Clara, CA, USA, Aug. 26–28, 2015, pp. 78–87.

[18] Y. Liu and B.C. Schafer, "Adaptive Combined Macro and Micro-Exploration of Concurrent Applications Mapped on Shared Bus Reconfigurable SoC," *Electron. Syst. Level Synthesis Conf.*, San Francisco, CA, USA, June 10–11, 2015, pp. 11–16.

[19] W.H. Ho and T.M. Pinkston, "A Design Methodology for Efficient Application-Specific On-chip Interconnects," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 2, 2006, pp. 174–190.

[20] D. Bertozzi et al., "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, Feb. 2005, pp. 113–129.

[21] M. Jun, W.W. Ro, and E. Chung, "Exploiting Implementation Diversity and Partial Connection of Routers in Application-Specific Network-on-Chip Topology Synthesis," *IEEE Trans. Comput.*, vol. 63, no. 6, June 2014, pp. 1434–1445.

[22] C. Killian et al., "Smart reliable Network-on-Chip," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 22, no. 2, 2014, pp. 242–255.

[23] A. Mahdoum, "Architectural Synthesis of Networks on Chip," *IEEE Conf. Ind. Electron. Appl.*, June 2013, pp.1889–1894.

[24] K.-M. Lee, *Design and Implementation of Low-Power Network-on-Chip for Application to High-Performance System-on-Chip Design*, Ph.D dissertation, KAIST, Daejeon, Korea, 2005.

[25] N.L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*, Volume 2, Malden, MA, USA: Wiley-Inter Science, 1994.

[26] The MathWorks, Statistics Toolbox Documentation. http://www.mathworks.co.kr/access/helpdesk/help/toolbox/stats/index.html?/access/helpdesk/help/toolbox/stats

Jin-Sung Kim received his BS and MS degrees and his PhD in electrical engineering and computer science from Seoul National University, Rep. of Korea, in 1996, 1998, and 2009, respectively. From 1998 to 2004 and from 2009 to 2010, he was with the PDP Development Group, Samsung SDI Co., Ltd., Chonan, Rep. of Korea, as a Manager, where he was involved in driver circuits and discharge waveforms. From 2010 to 2011, he was a postdoctoral researcher at Seoul National University. In 2011, he joined the Department of Electronic Engineering, Sun Moon University, Asan, Rep. of Korea, where he is currently an associate professor. He is an area editor of the journal Displays. His current research interests include pattern recognition, video compression and image enhancement, and driving system for flat panel displays.

Jaesung Lee received his BS and MS degrees in electronic engineering from Ajou University, Suwon, Rep. of Korea in 1999 and 2001, respectively, and his PhD in electrical engineering and computer science from Seoul National University, Rep. of Korea in 2008. From 2001 to 2011, he was with the Cloud Computing Research Department of ETRI, Daejeon, Rep. of Korea. In 2011, he joined the Department of Electronic Engineering at Korea National University of Transportation, Chungju, Rep. of Korea, where he is currently working as an associate professor. His research interests include multimedia SoC design and computer architecture.