APPLYING GENETIC ALGORITHM TO SOLVE PARTITIONING AND MAPPING PROBLEM FOR MESH NETWORK-ON-CHIP SYSTEMS

WalidMoktharSalh[1] and Azeddien M. Sllame[2]

[1]Faculty of Science University of Tripoli Tripoli, Libya
[2]Faculty of Information Technology University of Tripoli Tripoli, Libya

ABSTRACT

This paper presents a genetic based approach to the partitioning and mapping of multicore SoC cores over a NoC system that uses mesh topology. The proposed algorithm performs the partitioning and mapping by reducing communication cost and minimizing power consumption by placing those intercommunicated cores as close as possible together. A program developed in C++ in which the provided specification of the multicore MPSoC system captures all data dependencies before any start of the design process. Experimental results of several multimedia benchmarks demonstrates that the genetic-based approach able to find different satisfied implementations to the problem of partitioning and mapping of MPSoC cores over mesh-based NoC system that satisfies design goals.

KEYWORDS

Network-on-Chip, Multicore, Partitioning, Mesh topology, Genetic algorithm

1. INTRODUCTION

System-on-Chip (SoC) is an electronic circuit which usually realizes the maximum tasks needed practically to perform an intended system's behavior. Current state-of-the-art SoCs contain memory modules/banks, instruction-set processors (central processing units, CPUs), (Intellectual Property) IP processing cores, specialized logic, busses and interconnection networks, multimedia cores, digital signal processing (DSP) units, and wireless transmitters/receivers. In addition, current SoC implementations may include also micro-electric-mechanical systems (MEMS) and microsystems cores including a set of hierarchical cores [1] [2]. The most essential issue with current SoC systems is the communication subsystem. However, current practices are using scalable interconnection networks as an alternative to old design style which uses different buses or wires along the chip to interconnect the cores of SoC system. Thus, as a result with technology advances, the SoC design approach is moved to multiprocessing system-on-chip (MPSoC) systems which may contain multi-thousands cores with hierarchical cores memories, sending and receiving devices and microsystems; which is now named as "Network-On-Chip" (NoC) design paradigm [2] [3] [4].However, Figure (1) illustrates the big picture of NoC system design with different interconnection networks which currently used in developing efficient MPSoC designs that are very useful and efficiently working with intensive multiprocessing applications.

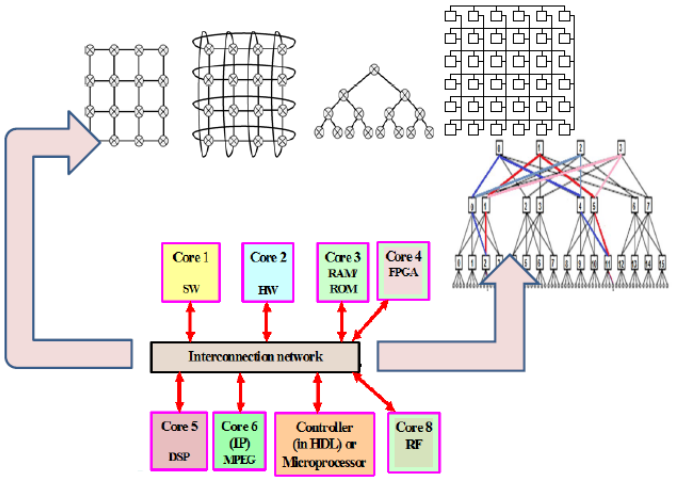


Figure (1): SoC generic architecture with different NoC interconnection topologies

NoC is anon-chip communication subsystem built to facilitate communication between cores composing of SoC systems. NoC design approach applies diverse networking algorithms, processes, theories, and methods to on-chip communications which bringsoutstanding improvements over typical bus or hierarchical buses and point-to-point links. NoC uses packet switching technique as found in TCP/IP networking model; instead of old fashioned bus techniques that communicate by wires routed around the chip [5]. More precisely, in NoC model, when one IP processing core is idle, other IP blocks (cores, components) continue to make use of the network resources. Consequently, NoC design approach has the benefits of being modular, scalable, well-structured, power efficient, employ reusable switches (routers), and flexible with higher bandwidth usability. However, Butterfly fat tree (BFT) and mesh topologies are considered essential representative prototypes of standard interconnection networks [1] [5], see Figure (1). However, the use of interconnection networks as a means of on-chip communication leads to use the principle of separation between computation and communication that has been used in supercomputing in the past.

This paper aims to implement a genetic algorithm which systematically changing its internal mechanisms to find a solution to the partitioning and mapping problem in NoC designs after reading the specification and use Hamilton distance to map the intercommunicated processing cores as near as feasible to reduce routing and power consumption overhead.

The paper is organized as follows: the Section 2 report the related work, Section 3 summarizes outlines multiprocessing SoC with core-based design approach. Section 4 explains the properties of mesh topology, while Section 5 describes the proposed design methodology with the genetic algorithm's structure. Finally results are reported in Section 6.

2. RELATED WORK

There are many research explorations about NoC design and multicore mapping techniques with diverse approaches carried out in a lot of universities and research centers internationally. The following are representative set from that research.

- Lei and Kumar in [6] described a genetic algorithm optimization technique to map different applications into a NoC system based on 2D mesh structure, which represented in graph representation. The applied genetic algorithm maps cores onto graph vertices with the aim

of reducing time consumed by application execution with efficient a delay model developed for NoC communication over mesh topology.

- Teich et al in [7] presented a technique that concentrates on the partitioning of system level specification using evolutionary algorithms. The methodology explores the effect of the partitioning process with different design implementations only at the system level. Following this, scheduling of each specification partition is made by a list-based scheduler.
- Henkel et al in [8] presented a high-level assessment methodology to discover the design space with respect to different design constraints in order to guide HW/SW partitioning at system level. The methodology considers only high-level decisions and does not consider any implementation details of the HW.
- Amit Kumar et al. in [9] outlined a state-of-the-art comprehensive analysis review and cataloguing of mapping techniques which focused on evolving developments for multicore design systems. An evaluation study is delivered comparing the analyzed techniques according to target optimization objectives.
- G. Ascia et al. in [10] addressed the problem of topological mapping of IP cores on the tiles of a mesh-based NoC architecture. The target was to get a mapping which enhances the design performance while reducing the consumed power over a mesh topology.
- Glenn Leary et al. in [11] described automation design tool for NoC designs based on genetic algorithm. The designed tool provides also applies communication synthesis method over interconnection networks.

The work in this paper is different than the one mentioned above works in such a way that; the system specification is read by the algorithm to determine the data dependencies among the statements and define their execution order.

After that, the core methods of the proposed genetic algorithm are systematically evaluated, modified and biased in the direction of gaining their power of finding proficient solution to the partitioning and allocation problem in NoC designs by reducing the communication between associated SoC cores by placing them as close as possible in order to minimize power consumption of the designed system.

3. MULTIPROCESSING SYSTEM-ON-CHIP

SoC systems are supported by a wide variety collection of microprocessors, processor architectures, IP cores, CPUs, application specific integrated circuits (ASIC), application specific instruction set processor (ASIP), digital signal processing cores (DSP), etc. At present, more than a hundred different microprocessors are available from more than 40 semiconductor vendors. SoC systems have been migrated into MPSoC design paradigm due to two reasons; first one is the introduction of interconnection networks as an on-chip communication means; the second one is the latest developments and advancements in deep submicron technologies which allowed producing billions of transistors into a single chip. Thus, allowed implementing multiple-instruction multiple-data streams (MIMD) processing systems into a single chip. On another hand, SoC designs have maturated to apply reusability by employing the cores-based design approach which facilitated designing powerful SoC systems with hundreds of heterogeneous cores that need to communicate efficiently into a single chip. However, core-based approach enabled designers to concentrate on single core/hierarchical or microsystem designs which lead to produce reusable, tested, well-documented cores with proper interfacing. Consequently, this made making NoC possible which resulted into realizing power efficient, modular, flexible, and scalable MPSoC systems for current intensive multiprocessing devices which overcomes the design problems encountered with SoC traditional design techniques such as: variety of dedicated

interfaces, design and verification complexity, unpredictable performance, and many underutilized wires while using buses and wires inside the chip [1] [4] [5].

However the heterogeneity on core types leads to the following characteristics in MPSoC:

a) MPSoC system with heterogeneous cores is modeled and described by interconnected executable processes which can be implemented as hardware, software, and communication components (routers and switches) that are employed to facilitate communication between system cores and external environment.
b) Heterogeneous MPSoC architecture encourages the separation between computation and communication.
c) The MPSoC heterogeneous architecture provides highly concurrent computation and flexible programmability.
d) The heterogeneous MPSoC are composed of different kinds of PEs which need different interfacing unis.
e) Homogeneous MPSoC are made of the same type of element which is instantiated several times.
f) Heterogeneous MPSoCs can include similar cores, hierarchal cores and divergent cores analog and digital which allows MPSoCs architecture to include large number of processing elements integrated into a single chip.
g) Next generation of heterogeneous MPSoC will be designed from few heterogeneous subsystems (hierarchy feature of multicore design methods), where each subsystem includes a massive number of identical processors (cores) to run a specific SW, or HW task.

4. MESH INTERCONNECTION TOPOLOGYAS AN ON-CHIP COMMUNICATION SYSTEM

With NoC implementation an aggregate bandwidth grows while the BW of the bus is shared and the speed goes down as the cores added to the bus implementation. In addition, NoC system designs are based on the interconnection networks which exhibit pipelining concurrency in their design as they have been developed to work with parallel computing and supercomputers. Bus implementations did not have any concurrency, whereas pipelining is difficult to implement with buses. Furthermore, the NoC designs demonstrate the separation of abstraction layers as well as separation of computation and communications which enhances the scalability issue of the NoC systems, in contrary bus designs suffer from such separation feature. However, unlike NoC designs, the bus designs are fairly simple to build with low cost, while NoC designs need new design techniques and methodologies, such as network interfacing and buffering and router (switch) designs. However, 2D mesh is a popular interconnection structure is currently the preferred choice for large-scale MPSoC implementations. With mesh structure, the links is only interconnecting directly neighboring nodes which highlights a regular layout of the mesh topology as a key advantage over other networking structures such as fat trees. On another hand, the key disadvantage of the mesh structure is the large number of hops (routers) traversed through the topology to reach their final destination. However, each router imposes a minimum latency and it is thus; a potential point of contention. A large number of hops have a direct impact on the energy consumed in interconnecting the communication path for buffering, transmission, and flow control. As a result, mesh topology will have performance degradation on performance, and scalability problems as the size of the mesh interconnection matrix grows quickly by increasing the number of nodes [12] [13] [14] [15].

4.1. Mesh Structure Concepts

The 1-D meshes are organized from linear array of processing cores, while the most practical mesh's topologies are, of type, 2D, and 3D meshes. In a mesh topology structure, the nodes are organized in am dimensional lattice of width w, producing a result of wm nodes; commonly m=1 (linear array) or m=2 (2D array). Therefore, the communications in mesh topology are allowed only between neighboring processing nodes. All interior nodes are connected to 2m other nodes and all the communication links are short and balanced. However, in 2D mesh architecture the routers or switches are found in each intersection between columns and rows; where switches (routers) are low radix with up to C+4 input and output ports. Consequently, this considered as a disadvantage because a large number of hops is needed for the packets to reach their final destination, which is considered a proportional to "N"; for N routers. For the routing purposes with mesh designs, the well-known XY routing is considered which uses a typical minimal turn deterministic algorithm. The XY routing algorithm decides the routing path to all current packets in every routing step, by firstly routing packets in X (horizontal direction) to the desired column, then finally performing the routing in Y (vertical direction) to reach the targeted destination. However, XY routing is performing efficiently on mesh topology and routers' addresses are their XY-coordinates, as seen in Figure (2) [14] [15].

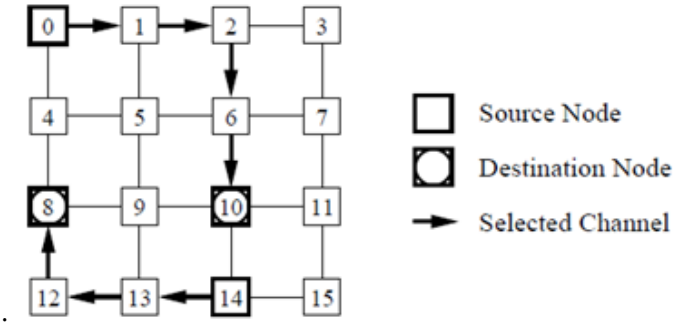


Figure (2): Routing inside mesh networks

4.2. Features of the Mesh Topology [12] [13] [14] [15] [16]:

- **Topology:** Mesh network has a fixed regular topology and belongs to direct type interconnection networks, in which switches with point-to-point interconnections are set up between the network nodes to establish the mesh structure. Though, as the dimension of mesh increases the number of switches (routers) increases, which increases the cost overheads, such as in the 2D mesh topology there are as many switches as processing cores. Mesh networks commonly use the same design for the switches included in the topology, each of them has connections between four neighboring cores. Yet, because of the structure regularity; the mesh topology has the feature of straightforward distance calculation between the sending and receiving nodes by summing up the offsets along mesh topology dimensions.
- **Network traffic balance:** In mesh interconnection networks deterministic routing in 2D mesh has in-order packet transfer which produces simple designs and efficiently working under uniform traffic only.
- **Deadlock, livelock, starvation:** Mesh uses XY deterministic routing that is thought-out as deadlock and livelock free.
- **Routing:** Mesh uses XY deterministic routing; in which all packets take the same path between any source-destination pairs. Hence, it is typically selects the shortest path associated with in-order flow control. In addition, the traffic with the traditional XY

routing; does not spread regularly over the entire network because the algorithm causes the biggest load flows in the middle of the network. In addition, under non congestion state, the mesh network which applies deterministic routing can achieve reasonable reliability with low latency overhead.

- **Fault tolerance:** as a result of its static interconnections among nodes which look as a matrix structure, the mesh topology possesses low fault tolerance characteristics.
- **Congestion control:** Since, mesh network has fixed regular structure so it is impossible for it to respond dynamically to network congestion. That will affect the efficiency and network's throughput negatively.
- **Latency and throughput:** In mesh networks, as the size of the network increases the latency and throughput will increase.
- **Network utilization:** XY deterministic routing in mesh networks causes the network resources to be underutilized. However, XY deterministic routing has a tendency to transfer packets in the direction of the mesh center which increase the network contention because the early network capacity saturation. Consequently, the performance will be reduced. On another hand, XY routing works well with uniform distributed traffic.
- **Scalability:** With scalability the performance increases with the addition of new processing cores consistently, which lead to more utilization of the network' bandwidth (BW). However, the BW in mesh networks is not scalable where it is static by the number of employed cores.
- **Energy dissipation:** The energy dissipation increases linearly as the number of channels increase in the operational switches in the mesh network.

Physical realization: The main practical feature of the mesh network is its simple mapping into the physical space of the chip using uniform short wiring between processing cores. However, more simplicity is expected as the size of the mesh found to be similar to the packaging technology's dimensions.

5. THE PROPOSED DESIGN METHODOLOGY

Current state-of-the-art research in MPSoC design domain is concentrating on defining specific features of infrastructure such as partitioning the MPSoC system specification amongst its available components (cores) and scheduling their execution in the selected components, selecting the suitable on-chip communication network subsystems, and defining interfacing. In this paper we will consider the partitioning and mapping of the specification (cores) of the MPSoC system into NoC mesh structure. The proposed methodology is described in Figure (3). Thus, the main work will be as follows:

(1) To model the MPSoC system in a text-like C++ specification, this captures the data dependencies among cores, which usually represented as data flow graph (DFG).
(2) To develop a genetic algorithm based solution to partition and map the cores of the MPSoC system into mesh type NoC system.
(3) To produce a software program for the developed genetic algorithm in C++ programming language to model the partitioning solution over the mesh NoC system.

However, the developed genetic algorithm is to make sure that all cores during the partitioning process are mapped efficiently as close as possible to their interconnected cores and all cores whose have an intensive communication and data transfer between them is placed closer than others in order to reduce the routing timing and communication overheads between them. Therefore, the employed genetic program is made in such a way that it easily finds one or more "optimized" implementation(s) for a given MPSoC system over mesh NoC architecture by

respecting the data flow dependencies and precedence constraints found in the supplied specification between the SoC' cores.

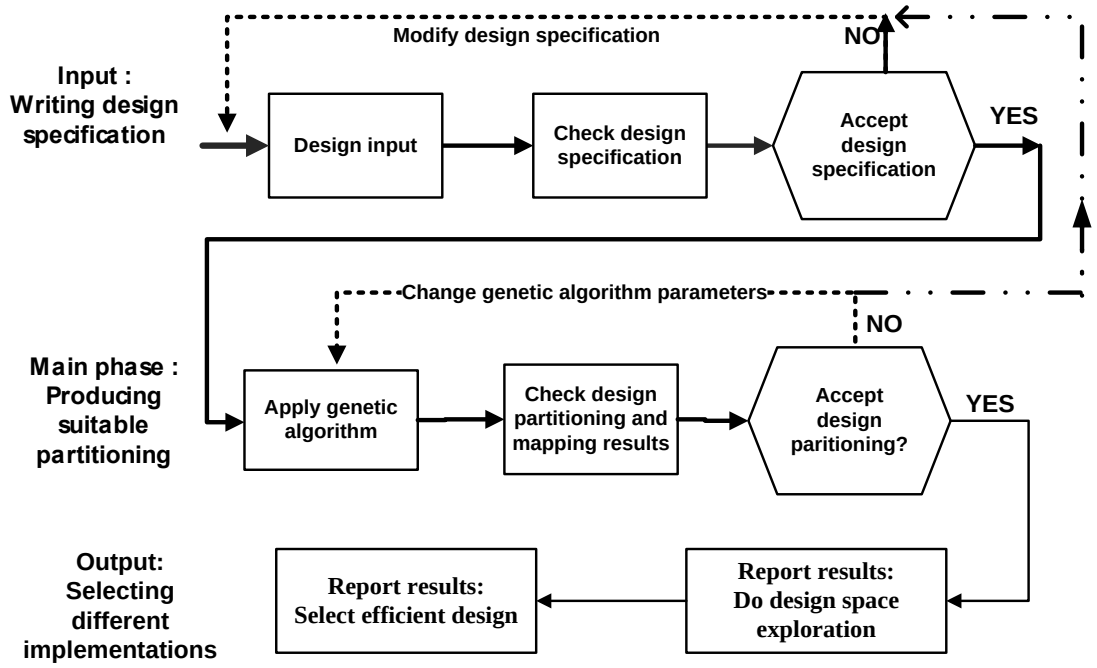


Figure (3): The proposed partitioning and mapping methodology

5.1. Formal Definitions

The following definitions are provided that may help formally understand the partitioning and mapping problem.

Definition 5.1.1:

MT is a set of operation types. Note that *MT* = { *, -, +, <>, /} in this paper examples.

Definition 5.1.2:

Core (component) graph (DFG) is a directed acyclic graph G (V, E1, ε)

Where:

V: is a set of vertices, each labeled with an operation using a mapping $\varepsilon : V \rightarrow MT$.
E_1: is a relation of the operations.

Note that a DFG does not have to be a single connected graph and a number of disconnected graphs by design imply multiple, parallel data flows.

Definition 5.1.3:

A precedence relation between two operations $v_i \in V$ and $v_j \in V$ is denoted by $v_i \rightarrow v_j$, where v_i is the immediate predecessor of v_j and v_j uses the result of operation v_i. This data dependence will

be defined as a precedence constraint during the mapping and scheduling process between the two operations of the core which must be satisfied.

Definition 5.1.4:

η: MT $\rightarrow N^+$ is a resource constraint mapping, which assigns the number of functional units to every operation type.

Definition 5.1.6:

A core library CL is a set of K cores,

Where $\forall i, j : i, j \in \{1..K\}$ *and* $i \neq j$ *holds* *if* $t_i = t_j$ *and* $d_i < d_j$ *then* $a_i > a_j$.

Definition 5.1.7:

A system-on-chip is modeled as a directed acyclic graph SoC (VM, E_2, CL)

Where
VM is a set of cores v_i, $v_i \in$ CL.
E_2 represents connections of the cores.
CL is a component library used for implementation.

Definition 5.1.8:

An implementation of a DFG G (V, E_1, ε) is a system-on-chip SoC (VM, E_2 , CL) if mappings exist:

$$\alpha : V \rightarrow VM$$
$$\beta : E_1 \rightarrow E_2$$

Such that the DFG and SoC produce the same result for any input data.

5.2. Genetic Algorithm Design

Genetic algorithms are well-known global probabilistic search methods initially begins by building an initial population of generated potential solutions to a problem, and progressively advances towards better candidate solutions by applying genetic operators of crossover, mutation, and potential elections of competent solutions reiteratively. Furthermore, the genetic algorithms approach of have been employed to find optimized solutions on different design problems in many engineering fields such as chip designs and networking [17][18][19][20]. However, genetic algorithms technique doesn't work with a single solution at a team, but it works concurrently with a large group of candidate solutions which enable finding efficient solutions by discovering the design space globally. Crossover is a reproduction technique that takes parent chromosomes and mates them to new child chromosomes. Mutation is employed to search for different designs in the design space in order to keep the range and the variations of the design population adequate to find satisfactory solutions. In addition, fitness function is used to control and measure the quality of the population progression toward better solutions, since the fitness of any obtained solution during the design process that is a gained score marks how-much suitable the produced solution while satisfying design goals. However, it is also known from these references that the

evolutionary algorithm does not guarantee an optimal solution being reached in every run, but, if the problem domain is ingeniously captured in the genetic algorithm, then it can outperforms traditional solutions [20]. However, during the design process of the SoC using the genetic program the designers seek to find one or more "optimized" implementation(s) for a given system such that all cores are distributed and mapped efficiently so that the performance is maximized and power dissipation is minimized by mapping all cores as close as possible to their parents and interrelated cores which will reduce the routing process overhead of the targeted NoC architecture.

Toward explaining the idea of the proposed genetic algorithm an example of video object plane decoder is selected; as shown in Figure (4); as one of the standard benchmarks in the field to demonstrate the results obtained by the applied technique. Figure (5) describes the data dependencies relations between the nodes of the video object plane decoder.

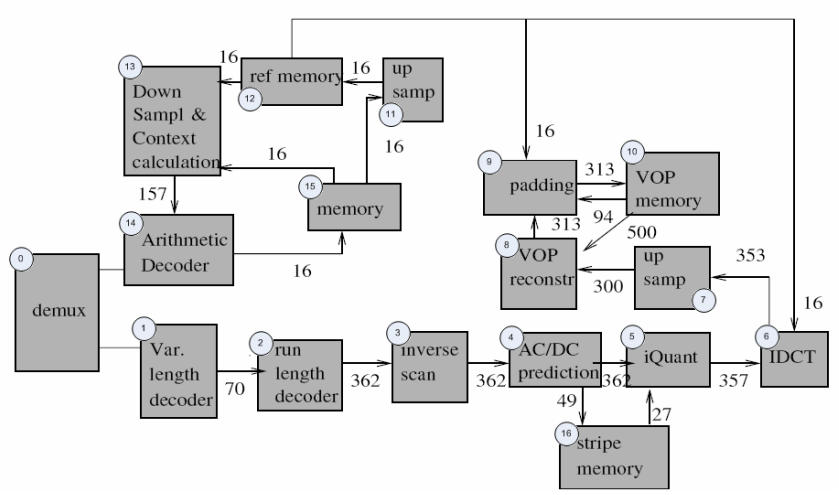


Figure (4): Block diagram of video object plane decoder

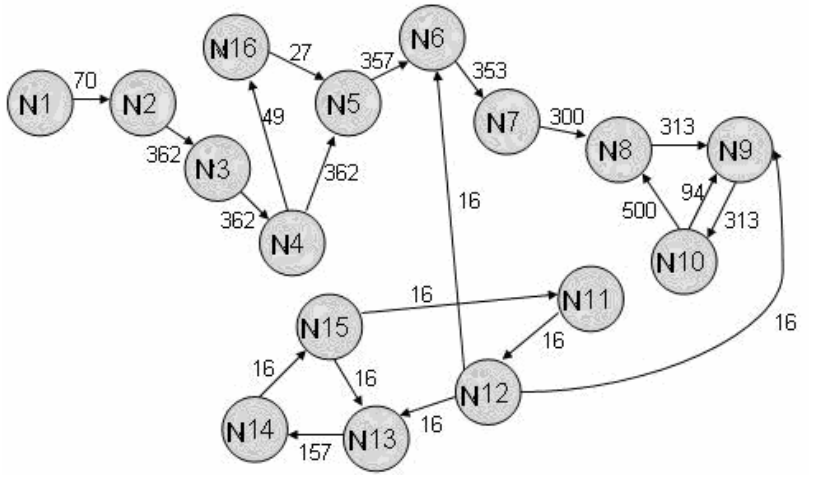


Figure (5): Precedence relations of the cores of the SoC example

The designed algorithm starts by reading the following inputs, as illustrated in Figure (3): (1) the specification of the MPSoC; (2) the size of the targeted NoC architecture (mesh size). Whereas the expected output is to produce an optimized SoC design in such a way that the communication

overheads are between cores are minimized; by placing interrelated cores as close as possible to required design performance; if one exists, otherwise either the mesh size is changed or the specification is revised or the genetic algorithm parameters is updated and applied again. What can be said here, that designers able to change design parameters, employ their experience, perform discussions within design teams in order to improve the reported design results. Since genetic algorithm can report many suitable implementations; but one of them might be optimal.

(A) Creation of the first population

The developed genetic program begins with the creation of the parents from an initial population, which are made up by two-dimensional two-array matrices (4 X 4) so that each matrix is a generation of 16 chromosomes, and each matrix is represented by a number representing each number of components (cores) of the SoC system. However, the chromosome looks as follows:
n1=1 , n2=2, n3=3, n4=4, n5=5, n6=6, n7=7, n8=8, n9=9, n10=10, n11=11, n12=12, n13=13, n14=14, n15=15, n16=16.
Here, the distribution of chromosomes is random in both matrices, which named as parent1 and parent2, which are the two base processes by which the new generations will be produced. The generation of first population is stored into two random arrays par1, par2 as seen in tables below.

		Parent1	
01	14	02	03
11	04	08	16
13	05	09	15
07	10	06	12

		Parent2	
10	11	07	06
14	02	01	05
12	13	15	16
09	04	03	08

(B) Fitness calculation

The fitness function calculates the fitness of each generation, which is a two-dimensional matrix based on measuring the distance between each core (HW component, processing node) and the cores it needs to communicate with them in order to reduce the routing process overhead between them using the Hamilton distance using the formula ($H\ i, j = |Xi - Xj| + |Yi - Yj|$).

For example in parent1 data, the Hamilton distance between core 12 and core 10 is equal to = 2, whereas the distance between core 12 and core 11 = 3, the distance between core 12 and core 14 = 1, the distance between core 3 and core 13 = 2, the distance between core 3 and core 1 = 2, the distance between core 3 and core 12 = 3, the distance between core 9 and core 13 = 3, the distance between core 9 and core 7 = 5, the distance between core 9 and core 8 = 3, and the distance between core 6 and core 5 = 1 and the distance between core 6 and core 4 = 5 and the distance between core 6 and core 14 = 4 to be the total fitness output is the sum of the distances between each core and the core that is needed here in this chromosome (matrix) = 33.

Parent2 has a distance between core 12 and element 10 = 2, the distance between element 12 and element 11 = 5, the distance between element 12 and element 14 = 5, the distance between element 3 and element 13 = 5, the distance between core3 and core1 = 3 and the distance between core3 and core2 = 1, the distance between core9 and core13 = 2, the distance between core9 and core7 = 3, the distance between core9 and core8 = 1, the distance between core6 and core5 = 2, the distance between core6 and core4 = 3, and the distance Between core6 and core14 = 4. Therefore the total fitness result is the sum of the distances between each core and its related cores, which for this chromosome = 36.

The best generation is elected by comparing Parent1with Parent2, which shown in below. In this example the second father whose fitness is equal to 46 is selected because it is less than the first father' fitness value which equals to 63, then it is stored as the solution of the best father to enter the new cycle of generations to elect who is better than him if it is found in future generations, as shown below.

(C) Rotary processing

Here the rotary is entered and we have said that it represents the number of generated generations in this algorithm and the first thing inside this rotary is to convert the two binary matrices into two monolithic matrices so that the program performs the crossover and mutation and other operations within a rotary that we initially defines as to be 10, 20, 100, 1000, or 10000. It represents the number of generated generations that will be created and the comparison between them and who will be selected represents the best optimization solution.

		Parent1	
01	14	02	03
11	04	08	16
13	05	09	15
07	10	06	12

		Parent2	
10	11	07	06
14	02	01	05
12	13	15	16
09	04	03	08

Parent 1

10	11	7	6	14	2	1	5	12	13	15	16	9	4	3	8

Parent2

1	14	2	3	11	4	8	16	13	5	9	15	7	10	6	12

(D) Crossover

The proposed genetic algorithm performs the crossover process by selecting one common point between the two matrices of the parents which need to make an exchange of their two matrices parts between them at the specified crossover point in order to form two new children namely child1 and child2. In this step there are some iterations in some values i.e. there are repeated values in each matrix, and these are resolved by the program directly, where the developed algorithm excludes any repeated values in the matrix and put a value that does not exist to prevent the process of repetition in one generation and this repair is done in the same part of the crossover process. The output from the program execution of the crossover is shown below.

One- point crossover

Parent1

10	11	7	6	14	2	1	5	12	13	15	16	9	4	3	8

Parent2

1	14	2	3	11	4	8	16	13	5	9	15	7	10	6	12

After crossover and repair

child1

1	14	2	3	7	4	6	5	12	15	13	16	9	10	11	8

child2

10	11	7	6	14	2	8	16	13	5	9	15	3	1	4	12

(E) Mutation

In this step, the mutation of the generation process is done, which is made by choosing two random children in the same generation and is switched between them to create a genetic mutation, and this process is usually applied after a number of cycles and here we have chosen to be perform the mutation process after every five cycles during the program execution period. The following tables show the steps of the mutation with mutation results.

Before mutation

Child 1

1	14	2	3	7	4	6	5	12	15	13	16	9	10	11	8

Child 2

10	11	7	6	14	2	8	16	13	5	9	15	3	1	4	12

After mutation

Child 1

1	14	2	9	7	4	6	5	12	15	13	16	3	10	11	8

Child 2

10	11	7	3	14	2	8	16	13	5	9	15	6	1	4	12

(F) Election of the best generation

Now, in this step, the two single matrices must be returned to two pairs to complete the selection or distinction process, which is the election of the best generation that is made by calculation and record-keeping of fitness's value of each generation.

(G) Fitness recalculation

The fitness of each generation is calculated by a function of measuring the Hamilton distance between each node or core with all other components or cores that it needs to communicate with them, i.e. route packets between them, by the formula: H i, j = | Xi –Xj | + | Yi - Yj |. However, the process of electing the best generation between the two generations of new child1 and child2 is done by comparing the best generation of parent1or parent2, and electing the best one of them.

(H) Obtaining the optimized solution

The rotary returns if the number of revolutions (the generations) has not ended and the same previous operations are performed again so that the best matrix is registered and it is the best generation in terms of fitness. Thus, an optimized solution is found; which could be an optimal solution. Mapping results to 4X4 mesh NoC topology.

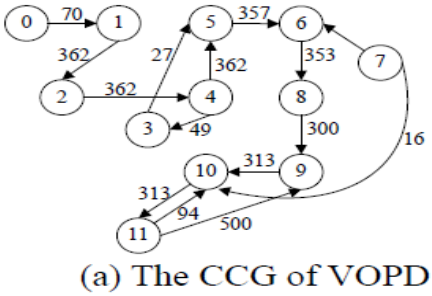
	Final	Mapping	
14n	13n	1n	9n
15n	11n	3n	10n
12n	2n	7n	8n
6n	5n	4n	16n

(I) Designer intervention

Note: The designer still can have the ability to change and move nodes mappings to achieve and realize his/her point of to the design.

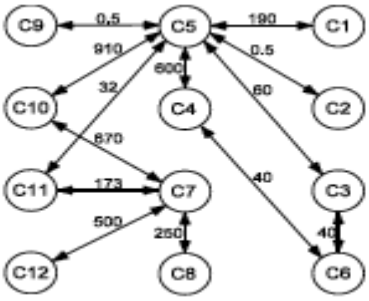
6. PRACTICAL RESULTS

Figure (6) describes the dependency graph of standard multimedia benchmark of CCG VOPD multimedia algorithm, and Figure (7) illustrates the MPEG4 multimedia algorithm, while Figure (8) shows the audio video multimedia algorithm, see [22][23][24].



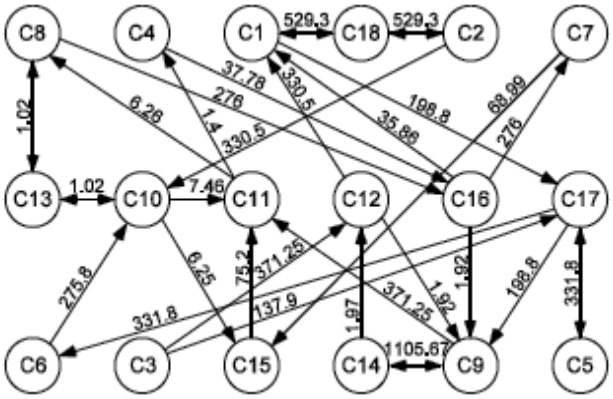
CCG	Final	mapping	
0	1	x	x
2	4	3	x
x	5	10	11
7	6	8	9

Figure (6): The mapping results of CCG multimedia algorithm



MPEG4	Final	mapping	
12	6	x	x
7	11	4	2
x	3	5	9
8	10	1	x

Figure (7): The mapping results of MPEG4 multimedia algorithm



	Final	mapping	Of	AV benchmark
4	x	11	8	x
18	1	9	16	x
2	6	10	x	14
3	12	15	13	x
x	17	5	7	x

Figure (8): The mapping results of audio video(AV) benchmark with 18 cores

7. CONCLUSION

We applied the presented approach on various multimedia benchmark applications with experimental results showing that it is possible to find efficient (not necessary optimal) MPSoC mapped designs using the genetic algorithm. Thus, from the results of implemented benchmark examples we can say that the realized genetic program of mapping-cores approach is producing practically efficient results which can aid to explore the MPSoC design space carefully. Consequently, it is clear that the results of mapping MPSoC cores using mesh as a NoC topology satisfies design specified goals in terms of reducing communication cost and power consumption.

REFERENCES

[1] Ahmed A. Jerraya, Wayne Wolf (editors) [2005]. Multiprocessor Systems-ON-Chips, Morgan Kaufmann Publishers, San Francisco, USA.
[2] Resve Saleh, et al.: System-on-Chip: Reuse and Integration, Proceedings of the IEEE, Vol. 94, No. 6, June 2006.
[3] Jantsch A, Tenhunen H., 2003. Networks on Chip. Kluwer Academic Publishers, USA.
[4] De Michelli G, Benini L. Network on Chips. Morgan Kaufmann: Berlin, 2006.
[5] Dally, W. J., and B. Towles, “Route packets, not wires: On-chip interconnection networks,” Proc. of the DAC'38 Conference, Las Vegas, June 2001.
[6] T. Lei and S. Kumar. "A two-step genetic algorithm for mapping task graphs to a network on chip architecture", In Euromicro Symposium on Digital Systems Design, Sept. 1-6, 2003.
[7] J. Teich, T. Blickle, L. Thiele: An Evolutionary Approach to System-Level Synthesis, In IEEE Proc. of Codes/CASHE’97 5th Int. Workshop on Hardware/Software Codesign, Braunschweig, Germany, March 1997, pp. 167-171.
[8] J. Henkel, R. Ernst: High-Level Estimation Techniques for Usage in Hardware/Software Co-Design, In Proc. of IEEE/ACM Asia and South Pacific Design Automation Conference ASP-DAC’98, February Japan, 1998, pp. 353-360.
[9] Amit Kumar Singh ; Muhammad Shafique ; Akash Kumar ; Jörg Henkel: Mapping on multi/many-core systems: Survey of current and emerging trends, In Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE, NY, USA, May 2013.Page(s):1 - 10.
[10] G. Ascia, V. Catania ; M. Palesi: An evolutionary approach to network-on-chip mapping problem, 2005 IEEE Congress on Evolutionary Computation, Sept. 2005, Edinburgh, Scotland, Vol.1, pp.112 – 119.
[11] Glenn Leary, Krishnan Srinivasan ; Krishna Mehta ; Karam S. Chatha: Design of Network-on-Chip Architectures With a Genetic Algorithm-Based Technique, In IEEE Transactions on Very Large Scale Integration (VLSI) Systems ,Vol.17, Issue: 5) pp.674 – 687.
[12] Tobias Bjerregaardand Shankar Mahadevan: A Survey of Research and Practices of Network-on-Chip, ACM Computing Surveys, Vol. 38, Article no.1, March 2006.

[13] Kavaldjiev, Nikolay Krasimirov, and Gerardus Johannes Maria Smit, A survey of efficient on-chip communications for soc, Centre for Telematics and Information Technology, University of Twente, 2003.
[14] Dally, W. J., and B. Towles. 2004. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers, San Francisco, USA.
[15] Duato J., S.Yalamanchili, L. Ni. 2002. Interconnection Networks–An Engineering Approach. Morgan Kaufmann Publishers, San Francisco, USA.
[16] Azeddien M. Sllame, Amani Hasan: A Comparative Study between Fat Tree and Mesh Network-on-Chip Interconnection Architectures, In The 14th annual Middle Eastern Simulation and Modelling Conference (MESM'14), pp. 31-37, Muscat, Oman, 3-5 February 2014.
[17] T. Baack: Genetic Algorithms in Theory and Practice. Oxford University Press, New York, Oxford, 1996.
[18] D. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, 1989.
[19] J. Teich, T. Blickle, L. Thiele: An Evolutionary Approach to System-Level Synthesis, In IEEE Proc. of Codes/CASHE'97 5th Int. Workshop on Hardware/Software Codesign, Braunschweig, Germany, March 1997, pp. 167-171.
[20] Azeddien M. Sllame, L. Sekanina: An Evolutionary-Based Algorithm to the Module Selection Process in High-Level Synthesis, In Proc. of 8th Int. Conference on Soft Computing Mendel'2002, Brno, Czech Republic, June 2002, pp. 87-92.
[21] N.Janakiraman, P.ManoArunika, P.Nirmal Kumar, "An optimized Mapping of IP Core onto NoC using Multi-Objective Evolutionary Algorithms" , International Journal of Innovative Research in Science, Engineering and Technology Volume 3, Special Issue 3, March 2014.
[22] Antoine Jalabert Srinivasan Murali Luca Benini Giovanni De Micheli: ×pipes Compiler: A tool for instantiating application specific Networks on Chip, Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'04), IEEE, 2004.
[23] Fen Ge, Ning Wu, Xiaolin Qin and Ying Zhang: Clustering-Based Topology Generation Approach for Application-Specific Network on Chip, Proceedings of the World Congress on Engineering and Computer Science 2011 Vol II WCECS 2011, October 19-21, 2011, San Francisco, USA.
[24] Ahmed A. Morgan, Haytham Elmiligi, M. Watheq El-Kharashi, and Fayez Gebali: Networks-on-Chip Architecture Customization using Network Partitioning: A System-Level Performance Evaluation, Int. J. Com. Dig. Sys. 4, No.1 (Jan-2015), ISSN (2210-142X).