PFMAP: Exploitation of Particle Filters for Network-on-Chip Mapping

Salih Bayar and Arda Yurdakul, *Member, IEEE*

***Abstract*— In this paper, we propose a mapping algorithm called particle filter mapping (PFMAP); PFMAP is able to map task nodes onto the cores of tile-based network-on-chip (NoC) architectures, such as regular, irregular, and custom 2-D or 3-D topologies. PFMAP is inspired from systematic resampling algorithm for particle filters, in which all particles can run parallel and independently from each other. Based upon the experimental results from applying PFMAP for various real life and synthetic applications onto the different topologies and architectures, the performance of the 2-D mesh architectures in terms of communication cost increased by up to 51% for irregular topologies, and by up to 31% for custom architectures. Similarly, total travel distance obtained by PFMAP is reduced by up to 45% for custom 2-D mesh architectures. In addition to these, average clock cycles per flit and total network power are reduced by up to 17% and 15% for regular 2-D mesh architectures, respectively. Finally, communication cost is diminished by up to 34% for 3-D regular NoC architectures.**

***Index Terms*— Communication system traffic, digital signal processing, greedy algorithms, multithreading, network-on-chip, parallel algorithms, routing, system-on-chip.**

I. Introduction

COMMUNICATION architectures, such as point-to-point and shared bus are poorly scalable as the number of cores and the communication volume increase [1]. In the last decade, networks-on-chip (NoC) have been proposed to reduce power consumption and have been widely adopted by the System-on-Chip community. Because of its scalability, NoC is a promising solution for the communication in a multicore architecture.

There are various parameters that affect the performance of NoC communication architecture directly or indirectly. Most important ones are mapping and routing algorithms, network topology, switching method, router architecture, and link bandwidth. If the mapping process is not carried out properly, then improving other parameters will not improve the performance of NoC significantly. Static mapping is done at compile time. Dynamic mapping runs on the fly and requires more complex components, such as observer and reconfiguration engine. Yet, dynamic mapping also requires an initial mapping. Our objective is the static mapping of cores on a regular, irregular, and custom 2-D or 3-D mesh architecture by minimizing the communication cost. This is the first step to minimize overall packet latency and network power [2]–[4]. Hence, instead of *static mapping*, we will use *mapping* in the rest of this paper.

In the literature, there are various studies in the field of node to core mapping of regular NoCs. However, none of these studies utilize particle filtering algorithm to solve the mapping problem for NoCs, which is a nondeterministic polynomial time-hard problem [5]. Yet, particle filters are widely used in applications, such as positioning, localization, tracking, and navigation in robotic automotive industry [6]–[8]. Particle filtering is a sequential Monte Carlo technique for the solution of the state estimation problem [9]. The original particle filtering algorithm is called sequential importance resampling and used frequently [10]. The main point in particle filtering is representing the required posterior density function by a set of random sample particles with corresponding weights, and to compute the estimates based on these samples and weights. As the number of samples and resampling iterations to generate new samples become very large, the solution approaches the optimal Bayesian estimate. There are various resampling methods for particle filtering algorithm [11]. Systematic resampling algorithm [12] is widely used because it is easy to implement and it outperforms other resampling approaches in most scenarios. Moreover, in terms of resampling quality, systematic resampling has the minimal variance [13]. Hence, we have preferred to use systematic resampling in particle filter mapping (PFMAP).

PFMAP is very suitable for solving mapping problem for the following reasons.

1) A configuration on any type of regular, irregular, and custom 2-D or 3-D NoC topology can be represented by particles.
2) Particles do not require a fixed computation time; instead, accuracy increases with the available computational resources [7].
3) Implementation of particle filters is extremely easy, especially systematic resampling algorithm.
4) Particles give much better results than their counterparts in the solution of mapping problem in most of the time.
5) Particles in PFMAP are totally independent from each other and therefore they all can run in parallel. Hence, it is easy to implement PFMAP on parallel computational platforms, such as multithread, graphics processing unit (GPU), and video processing unit (VPU).

The organization of this paper is as follows. Section II presents a short survey of related works on mapping of task nodes onto the cores of a tile based NoC. In Section III, our PFMAP algorithm is given in detail. In Section IV, we present

Manuscript received August 22, 2013; revised January 7, 2014, April 23, 2014, and July 28, 2014; accepted September 15, 2014. Date of publication October 22, 2014; date of current version September 23, 2015. This work was supported by the State Planning Organization of Turkey through the TAM Project under Grant 2007K120610.

The authors are with the Department of Computer Engineering, Boğaziçi University, Istanbul 34470, Turkey (e-mail: salih.bayar@boun.edu.tr; yurdakul@boun.edu.tr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2014.2360791

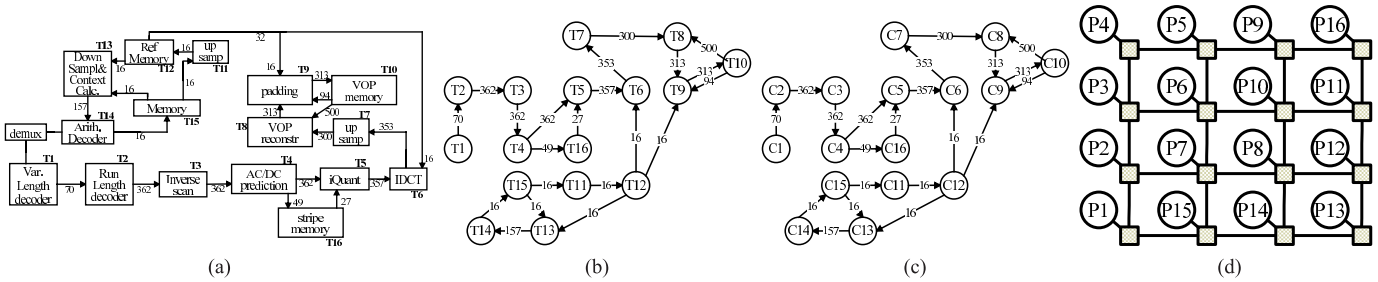


Fig. 1. VOPD application with 16 cores [26]. (a) Block diagram. (b) TTG. (c) CTG. (d) Mapping with NMAP [2].

our case studies and experimental results on various NoC topologies. Section V concludes this paper.

II. Related Works on Mapping

A vast amount of methods has been proposed to solve mapping problem. In Physical Mapping Algorithm (PMAP) [14], two-phase mapping algorithm for placing clusters onto processors is used. In NMAP [2], Dijkstra's shortest path on quadrant graph is applied to solve mapping problem. Both NMAP and PMAP are fast heuristic methods based on the approach of placing the most communicating nodes neighbor to each other by mapping heavy weight nodes at first. In [15] is a greedy algorithm, which uses n-ary search tree. The final configurations are given in the leaf nodes of their search tree. Although this algorithm uses branch and bound, space complexity of this paper is in the factorial range. SUNMAP [16] extends NMAP to support new NoC topologies, such as torus and hypercube. In chaos-genetic-based algorithm [17], a genetic algorithm using chaotic systems is presented. In Onyx [18] and Crinkle [19], priority lists are utilized. In [20], an optimized simulated annealing approach is proposed and tested on various task graphs. Two different algorithms, named A3MAP-genetic algorithm (A3MAP-GA) and A3MAP-successive relaxation (A3MAP-SR), have been presented in architecture-aware analytic mapping [3], [4]. A3MAP-GA is a genetic algorithm, while A3MAP-SR is a successive relaxation algorithm. There is an intensive survey on application mapping strategies for NoC design [5]. In this paper, besides giving classification of mapping algorithms, communication cost of some benchmark applications is compared for various known algorithms. There are also studies trying to find optimum solution of the mapping problem using integer linear programming (ILP) [21]. Since the mapping problem is intractable, it is not possible to find a solution for medium and large size problems using ILP. In Kernighan–Lin partitioning and mesh topology mapping algorithm (LMAP) [22], Kernighan–Lin-based partitioning is used to solve the mapping problem for regular architectures. In particle swarm mapping (PSMAP), Sahu *et al.* [23] propose particle swarm optimization. Similar to [22], the scalability of this algorithm is also ambiguous, since it gives only a few sample applications mapped on a 2-D regular mesh architecture.

In most of these previous studies, video applications, such as video object plane decoder (VOPD), MPEG4, and high-end video applications, such as picture in picture, multiwindow application (MWA), MWA with graphics, dual screen display, multimedia system (MMS) including H263 Dec. and Enc., MP3 Dec. and Enc. are used. In addition to them, embedded system synthesis benchmarks suite (E3S) [24] and for synthetic task graphs for free (TGFF) [25] are used in most of these studies.

Most studies are heuristic due to complexity. In most of these algorithms, for a fixed problem size, the running time of the algorithms are fixed. However, in PFMAP, running time of the algorithm depends on the user. User can set a desired time to finish the algorithm. In PFMAP, as the number of samples and resampling iterations to generate new samples become larger, the solution approaches the optimal Bayesian estimate. In addition, in PFMAP, particles representing a configuration can run fully in parallel. Thus, PFMAP is very suitable for current and next generation multithreaded or real parallel platforms, such as multicore, GPU, and VPU.

III. Proposed Algorithm

Two graphs are the inputs to PFMAP. The first one is task traffic graph (TTG), where the task nodes and the communication flows between them are defined. The second graph is core traffic graph (CTG) in which processor or computational cores and their communication relationships are given. Another input is the topology of the NoC, which is called router configuration topology (RCT). We define configuration as the placement of cores on tile-based NoC architecture. As an example, the block diagram of VOPD application is given in Fig. 1(a). Each block in this application can be considered as a task node. In Fig. 1(b), task graph of VOPD application is given. This task graph is generated according to the block diagram in Fig. 1(a). Here, weighted directed edges represent the average communication volume in MB/s from one node to another node. The task graphs used in this paper characterize the partitioning, task assignment, scheduling, communication patterns, and task execution time of a given application [15]. Similarly, Fig. 1(c) shows the core graph of target application. In Fig. 1(d), a solution is found for mapping problem of VOPD application to a 2-D regular mesh NoC architecture using NMAP algorithm. Here, rectangular shapes represent routers, and circular shapes represent processor cores attached to routers.

Mathematical formulation of the mapping problem can be given as follows.

Definition 1: TTG is a directed graph, TTG(N, T) with each vertex $n_i \in N$ representing a task node, and the directed edge between n_i and n_j indicated by $f_{i,j} \in T, i \neq j$ represents the traffic flow. The weight of $f_{i,j} \in T$ is the traffic amount from n_i to n_j and denoted by $t_{i,j}$. In TTG, $|N|$

represents the number of task nodes, while $|T|$ is the number of nonzero directed edges between n_i and n_j, s.t. $i \neq j$. $|T| \leq |N| \times |N-1|$.

Definition 2: CTG is a directed graph, CTG(C, T) with each vertex $c_i \in C$ representing a processor or a computational core. The directed edge between c_i and c_j indicated by $l_{i,j} \in T$ represents the link between source and destination nodes. The weight of each link $l_{i,j} \in T$ is denoted by $t_{i,j}$ representing the traffic amount on the current link.

One-to-one mapping of the TTG onto the CTG can be defined by

$$\text{map} : N \mapsto C, \ni \text{map}(n_i) = c_i \quad \forall n_i \in N, \quad \exists c_i \in C. \quad (1)$$

We assume that, each single task node $n_i \in N$ is mapped onto a single processor or computational core $c_i \in C$, on which there is no any other task node $n_j \in N$ is mapped yet. Hence, TTG and CTG are identical, i.e., $|N| = |C|$. In some applications, however, more than one task node can be mapped to a processor core or in the similar way, one task can be partitioned into subtasks, and these subtasks can be mapped onto multiple processor cores. We assume that these operations are carried out prior to PFMAP.

Definition 3: RCT(R, C) is a 2-D mesh NoC topology with each ordered pair $P_{r,c} \in \text{RCT}(R, C)$ representing the physical location of a processor core attached to a router on the target architecture. R and C indicate the number of rows and columns in the topology, respectively (i.e., $R \times C$ is NoC size). In $P_{r,c}$, r and c are the respective horizontal and vertical indices.

One-to-one mapping of the CTG onto the RCT can be defined by

$$\text{map} : C \mapsto \text{RCT}, \quad \ni \text{map}(c_i) = P_{r,c} \quad \forall c_i \in C, \exists P_{r,c} \in \text{RCT}(R, C). \quad (2)$$

Note that the mapping is valid if the number of cores to be placed ($|N|$) is less than or equal to the number of nodes on the target architecture ($|R \times C|$), $|N| \leq |R \times C|$.

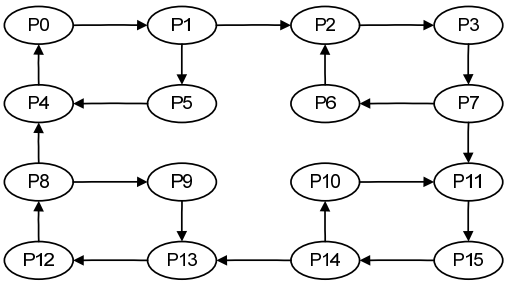
RCT is composed of only routers, and there are some limitations for routers in terms of their number of inputs and outputs: they have only n-bit single input and single output in one side (North–East–South–West). Each processor or computational core is attached to a router. Apart from the processor interface, both the maximum number of inputs and outputs for a router is four.

Definition 4: Manhattan distance (MDist) is the minimum number of hops from source node n_i to destination node n_j in RCT. The formula of the MDist for the nodes $P_{r1,c1}$ and $P_{r2,c2}$ in RCT is given by

$$\text{MDist} = |r1 - r2| + |c1 - c2|. \quad (3)$$

The communication cost of a configuration for regular 2-D mesh architectures is calculated using MDist between each node pairs. Communication cost for a single edge (CC_{se}) between $P_{r1,c1}$ and $P_{r2,c2}$ in a configuration is given by

$$CC_{se} = t_{P_{r1,c1}, P_{r2,c2}} * \text{MDist}(r1, c1, r2, c2). \quad (4)$$



(a)

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
P0	0	1	2	3	3	2	5	4	10	11	8	5	9	8	7	6
P1	3	0	1	2	2	1	4	3	9	10	7	4	8	7	6	5
P2	10	11	0	1	9	12	3	2	8	9	6	3	7	6	5	4
P3	9	10	3	0	8	11	2	1	7	8	5	2	6	5	4	3
P4	1	2	3	4	0	3	6	5	11	12	9	6	10	9	8	7
P5	2	3	4	5	1	0	7	6	12	13	10	7	11	10	9	8
P6	11	12	1	2	10	13	0	3	9	10	7	4	8	7	6	5
P7	8	9	2	3	7	10	1	0	6	7	4	1	5	4	3	2
P8	2	3	4	5	1	4	7	6	0	1	10	7	3	2	9	8
P9	5	6	7	8	4	7	10	9	3	0	13	10	2	1	12	11
P10	8	9	10	11	7	10	13	12	6	7	0	1	5	4	3	2
P11	7	8	9	10	6	9	12	11	5	6	3	0	4	3	2	1
P12	3	4	5	6	2	5	8	7	1	2	11	8	0	3	10	9
P13	4	5	6	7	3	6	9	8	2	3	12	9	1	0	11	10
P14	5	6	7	8	4	7	10	9	3	4	1	2	2	1	0	3
P15	6	7	8	9	5	8	11	10	4	5	2	3	3	2	1	0

(b)

Fig. 2. Irregular processor communication topology and its distance matrix. (a) Irregular processor topology. (b) Distance matrix.

CommCostReg is the total communication cost of a configuration for a regular 2-D mesh topology

$$CommCostReg = \sum_{r1=0}^{R} \sum_{c1=0}^{C} \sum_{r2=0}^{R} \sum_{c2=0}^{C} CC_{se}(r1, c1, r2, c2) \quad (5)$$

where R and C indicate the number of rows and columns in the RCT, respectively. Instead of using MDist, distance value of each computation core pairs for irregular, custom, and 3-D architectures is obtained by utilizing Dijkstra's shortest path algorithm [27] in the preprocessing step, where a distance matrix is generated. Then, this matrix is used as an input to PFMAP. In Fig. 2, a random processor communication topology and its corresponding distance matrix are given. Source and destination processor cores are given in the rows and columns of the distance matrix. For example, the distance from source node $P6$ (in row $P6$) to destination node $P5$ (in column $P5$) is given as 13. The path from $P6$ to $P5$ is as follows: $P6- > P2- > P3- > P7- > P11- > P15- > P14- > P13- > P12- > P8- > P4- > P0- > P1- > P5$.

In (6), *CommCostIrreg* is the total communication cost of a configuration for an irregular custom 2-D mesh or regular, irregular, and custom 3-D mesh topologies

$$CommCostIrreg = \sum_{i=0}^{E} \sum_{j=0}^{E} t_{i,j} * \text{dist}_{i,j} \quad (6)$$

where $\text{dist}_{i,j}$ is Dijkstra's shortest path.

Definition 5: CostLowerBound is the minimum communication cost that can be achieved in a given configuration. *CostLowerBound* calculation is given by

$$CostLowerBound = \sum_{i=0}^{E} \sum_{j=0}^{E} t_{i,j}. \quad (7)$$

Algorithm 1 Main Part of Configuration Mapping Algorithm

Input: Set of cores and nodes, TTG, CTG, RCT(R,C)
Output: Mapping of cores to nodes

1: $BestFitness = -1$
2: **for** $i = 0$ to IT **do**
3: **for** $j = 0$ to PN **do**
4: **if** $(i == 0)$ **then**
5: $randomConf(particles_j)$;
6: **else**
7: $randomNodeSwap(particles_j)$;
8: **end if**
9: $particleFitness_j \Leftarrow CalcFitness(particles_j)$;
10: $CurrFitness \Leftarrow particleFitness_j$;
11: **if** $CurrFitness > BestFitness$ **then**
12: $BestFitness \Leftarrow CurrFitness$
13: $BestConf \Leftarrow particles_j$
14: $BestCost \Leftarrow CommCost$ for $particles_j$
15: **end if**
16: **end for**
17: $sysResamp(particleFitnesses, chosens, PN)$
18: $chooseResampleds(particles, chosens, PN)$
19: **end for**

Algorithm 2 Resample Particles Systematically

Input: Particle fitnesses and set of particles
Output: Set of re-sampled particles

1: **Function** sysResamp(particleFitnesses, particles)
2: $fitnessSum = 0$
3: $curFitnessSum = 0$
4: $uni = 0$
5: **for** $i = 0$ to P **do**
6: $fitnessSum+ = particleFitnesses_i$
7: **end for**
8: $uni = ([0..1] \times (fitnessSum/P))$
9: $k = -1$
10: **for** $j = 0$ to P **do**
11: **while** $curFitnessSum < uni$ and $k < P - 1$ **do**
12: $k++$
13: $curFitnessSum+ = particleFitnesses_k$
14: **end while**
15: $resampledParticles_j \leftarrow k$
16: $uni+ = (fitnessSum/P)$
17: **end for**
18: **RETURN** $resampledParticles$
19: **EndFunction**

The main part of our PFMAP mapping algorithm is given in Algorithm 1. In this algorithm, each particle represents a mapping configuration. The algorithm generates PN random configurations in the first iteration (lines 4–6). The related function ($randomConf(particles_j)$) is an implementation of Fisher–Yates shuffle [28], which is used for the sake of performance. With this function, we randomly place the processor or computational cores on mesh NoC architecture initially for each configuration. Here, PN is the number of particles, and IT is the number of iterations, which we define before the beginning of running our algorithm. It should be noted that the running time of our algorithm is proportional to the PN and IT.

In the first iteration, we calculate the fitness for each randomly generated configuration (lines 9 and 10). The calculation of fitness function for mapping is given in (8). Here, *CommCost* is identical to *CommCostReg* if the architecture is regular 2-D mesh; otherwise, (i.e., architecture is irregular and custom 2-D mesh or regular, irregular, and custom 3-D mesh topologies) it is equal to *CommCostIrreg*

$$\text{Fitness}_{\text{mapping}} = 1/CommCost. \tag{8}$$

According to initial configurations, we calculate the fitness value for each configuration. Among these configurations, the largest fitness value is set as the minimum $BestFitness$ (lines 11–15).

After generating initial configurations and finding the *BestFitness*, we resample these configurations in each iteration (lines 17 and 18). Sampling from the distribution and checking those samples with largest fitness values can be utilized for a *low-cost configuration selection algorithm*. In the remaining IT-1 iterations, we apply pairwise swap among randomly selected nodes (line 7). At the initial steps, the algorithm might not represent the fitness function. However, after a *burn-in* period, it starts to converge to the distribution. The *burn-in* period is directly proportional to the application and NoC architecture size.

For all random function generations (lines 5 and 7), we used thread-safe single instruction multiple data-oriented fast Mersenne twister (MT) pseudorandom number generator [29] because of the following reasons.

1) It has larger period (up to $2^{216\,091} - 1$) than the original MT ($2^{19\,937} - 1$) [29].
2) It is roughly twice faster than the original MT and has a better equidistribution property as well as a quicker recovery from zero-excess initial state [29].
3) It is faster than other statistically reasonable generators (very useful when huge quantities of random numbers are required) [30].
4) Original MT is proven to be equidistributed (up to 623-D) for 32-bit values. It passes many stringent statistical tests, including the diehard test of Marsaglia and the load test of Hellekalek and Wegenkittl [31].
5) It is very common; it has strong support from the people knowledgeable in the same field.

In Algorithm 2, method for resampling of particles is given. Here, configurations can be considered as a probability distribution where each configuration's probability is given by (8). Therefore, as the communication cost of a configuration increases, the probability of its selection decreases for the next iteration.

In systematic resampling, new configurations (i.e., particles) are derived from the previous ones. A number of

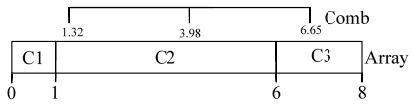


Fig. 3. Residing of three configurations on an array for resampling step using comb.

configurations at the input and output are the same. While the configurations with high *CommCost* values are mostly discarded, each configuration with lower *CommCost* value is reproduced a few times. Thus, at the end of resampling, probably there will not be only a single instance of a configuration with low *CommCost*.

For resampling step, fitness of each configuration is calculated. Then, the fitness values are located in a 1-D array. The total size of this array is the sum of fitness values (*fitnessSum* in line 6 of Algorithm 2) of P configurations. Here, the size of occupied area of each configuration is proportional to its size. Assume that we have three configurations $C1$–$C3$ and they have *CommCost* values as $1, 0.2$, and 0.5, respectively. Therefore, the corresponding fitness values are $1, 5$, and 2, respectively, according to (8). Residing of these configurations in the array is shown in Fig. 3.

In our resampling scheme, we define a comb with P teeth, which selects (i.e., resamples) the new configurations from the array. For our example, our comb has three teeth as shown in Fig. 3. Teeth of comb select the appropriate configurations. Here, the length of comb is shorter than the length of array, and the interval between teeth is equal. For P configurations, the length of comb can be given as

$$CombLength = \frac{\sum_{i=1}^{|P|} p_{\text{configuration}_i}}{P} \times (P-1). \quad (9)$$

For our example, the length of comb, $CombLength = [(1 + 5 + 2)/3] * 2 = 16/3$ and array length is eight. To locate the comb over the array, we generate a number from the uniform distribution on the interval $[0, ArrayLength - CombLength]$. This number, uni, is the leftmost tooth of the comb (line 8). For our example, this interval is $[0, 8/3]$ and assume that uni is 1.32. The process of locating the comb over the array and selecting new candidates are done in lines 10–17 and shown in Fig. 3. In lines 11–14, decision about the current configuration is given. If the sum of cumulative sum of fitness values (*curFitnessSum*) and fitness of the current configuration ($particleFitnesses_k$) is less than current value of uni (i.e., $k * (fitnessSum/P)$), k is incremented. Thus, the kth particle is not resampled; instead, $(k-1)$th particle is reproduced.

Our resampling method generates only a single real number. Hence, in the next iteration, the probability of having better configurations is increased while still keeping some of the configurations with higher costs as well.

IV. Case Studies

We implement our PFMAP algorithm in C++ with Open Multi-Processing (OPENMP) library [32]. All tests have been carried out on a 32-bit Windows-7 PC with a i5 CPU-750@2.67 GHz and 3-GB RAM. We performed our experiments with various video applications, such as VOPD, MPEG4-decoder, MWD, MMS-suite (H263-decoder, H263-encoder, MP3-decoder, MP3-encoder), and E3S benchmark suite [24] [Auto-Industry (AI), Consumer, Telecom]. In addition, we have generated various synthetic task graphs using TGFF [25]. These applications are mapped onto regular, irregular, and custom 2-D and 3-D NoC architectures.

For the simulation purposes, we have used NIRGAM NoC simulator [33]. NIRGAM is a SystemC-based cycle-accurate NoC simulator for 2-D regular NoC architectures. Yet, it does not support multicasting. Moreover, user cannot give the mapping as an input to the system. Hence, we modified NIRGAM to support these features. In NIRGAM, we selected the simulation frequency as 1 GHz and set simulation time to 1 ms. All other settings are left at their default values. It is enough to use hop count (*CommCost*) to evaluate quality of a mapping in terms of consumed energy [34] for regular architectures. The average energy consumption of sending one bit of data from one node (t_i) to another one (t_j) is determined by the MDist for regular architectures

$$E_{\text{bit}}^{t_i,t_i} = n_{\text{hops}} \times E_{S_{\text{bit}}} + (n_{\text{hops}} - 1) \times E_{L_{\text{bit}}}. \quad (10)$$

In (10), n_{hops} is the number of routers the bit traverses from tile t_i to t_j. $E_{S_{\text{bit}}}$ and $E_{L_{\text{bit}}}$ are the energy consumed by the switches and links between tiles, respectively. Since $E_{S_{\text{bit}}}$ and $E_{L_{\text{bit}}}$ are dependent on NoC architecture, n_{hops} determines the energy consumption for regular architectures, and it is directly related to mapping process.

A. 2-D Regular Mesh Architectures

In Table I, various applications are mapped with PFMAP onto regular 2-D mesh architectures. The results show both running time of our PFMAP algorithm and the communication cost of each scenario for different number of iterations (IT) and particles (PN). In Table I and in the following illustrations, while V represents the number of task nodes, and E shows the number of edges. For each application, the lower bound communication cost (LB) is the *CostLowerBound* in (7). However, sometimes it is impossible to reside all communicating task nodes as neighbor to each other. Hence, optimum solutions (OPs) need not be equal to lower bound cost values.

The halting methodology of our algorithm is as follows: if the resulting communication costs in the best case and worst case are close to the value of average case, we terminate the execution. In Table I, running time ($Run\ T.$) of our PFMAP algorithm is given in milliseconds. The best ($Best\ C.$), worst ($W.\ C.$), and average case ($Avg.\ C.$) communication costs are also presented. We tested each benchmark 100 times for given IT and PN values. In general, as IT and PN increase, we find better solutions. However, for some applications (i.e., for small and simple applications), it does not make any sense to run it for large values of IT and PN. For example, for the H263 encoder application, we found the solution for IT $= 100$ and PN $= 100$ in 1.42 ms. Hence, there is no need to run this application for larger IT and PN values anymore. For the medium-size problems (e.g., VOPD with 16 cores

TABLE I
ALGORITHM RUNNING TIME AND COMMUNICATION COST RESULTS OF PFMAP ON DIFFERENT APPLICATIONS

Nr. Of Tests= 100								
Application	IT PN	10 10	10 100	100 100	100 1000	1000 1000	1000 10000	10000 10000
H263 dec (V=12, E=14) LB: 213175 OP: 213372	Run T. [ms]	1.21	1.41	13.35	28.53	295	2036	21002
	Best C.	218378	213372	213372	213372	213372	213372	213372
	Avg C.	243168	219494	213785	213464	213444	213374	213372
	W. C .	295814	228920	216991	213660	213660	213660	213372
H263 enc (V=12, E=11) LB: 403817 OP: 404014	Run T. [ms]	1.19	1.42	13.14	-	-	-	-
	Best C.	406306	404014	404014	-	-	-	-
	Avg C.	523072	433431	404014	-	-	-	-
	W. C .	664716	522249	404014	-	-	-	-
MP3 dec (V=12, E=4) LB: 42420 OP: 42420	Run T. [ms]	1.32	1.55	-	-	-	-	-
	Best C.	42420	42420	-	-	-	-	-
	Avg C.	44197	42420	-	-	-	-	-
	W. C .	56550	42420	-	-	-	-	-
MP3 enc (V=12, E=8) LB: 70679 OP: 77740	Run T. [ms]	1.20	1.33	13.39	-	-	-	-
	Best C.	77740	77740	77740	-	-	-	-
	Avg C.	79557	77780	77740	-	-	-	-
	W. C .	100268	78380	77740	-	-	-	-
MWD (V=12, E=12) LB: 1120 OP: 1216	Run T. [ms]	1.25	1.37	13.59	31.12	316.22	1980	-
	Best C.	1408	1312	1216	1216	1216	1216	-
	Avg C.	1800	1558	1364	1264	1224	1216	-
	W. C .	2080	1792	1504	1344	1248	1216	-
MPEG4 Dec (V=12, E=21) LB: 3466 OP: 3633	Run T. [ms]	1.22	1.36	13.42	29.19	297	1995	-
	Best C.	4272.5	3752	3633	3633	3633	3633	-
	Avg C.	4900.7	4030.1	3693.3	3643.9	3637.6	3633	-
	W. C .	5275.5	4602	3772.5	3672	3672	3633	-
VOPD (V=16, E=21) LB: 3731 OP: 4119	Run T. [ms]	1.23	1.47	14.51	36.07	329	2516	25033
	Best C.	4950	4561	4167	4125	4119	4119	4119
	Avg C.	6608	5583	4670	4225	4136	4129	4124
	W. C .	7946	6321	5178	4469	4157	4135	4135

TABLE II
ALGORITHM RUNNING TIME AND COMMUNICATION COST RESULTS OF VARIOUS STUDIES

		Algorithms				
Application		NMAP	LMAP	PSMAP	ILP	PFMAP
MPEG4(4x4)	Run T. [s]	0.024	0.040	0.040	21.53	0.005
	Comm C.	3672	4006	3567	3567	3567
MWD(4x4)	Run T. [s]	0.016	0.030	0.020	200.5	0.015
	Comm C.	1184	1248	1120	1120	1120
VOPD(4x4)	Run T. [s]	0.024	0.040	0.260	21.53	0.329
	Comm C.	4265	4189	4119	4119	4119

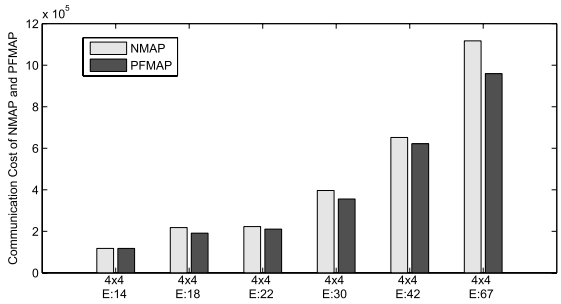


Fig. 4. Communication cost comparison of PFMAP and NMAP on a 2-D NoC with fixed size (4 × 4) with increasing communication demand.

on 4 × 4 2-D mesh NoC), it is not easy to apply our halting methodology. For that reason, after some burn-in period (if the resulting worst case solution is near to the best solution), we stop the running of our PFMAP algorithm. In such a situation, a designer can set a threshold value (e.g., 105% of the best communication cost) and check the average and worst values. If they are less than the threshold value, the algorithm might be halted.

We also compared the communication cost and running time of PFMAP algorithm with NMAP, LMAP, PSMAP, and ILP studies in Table II. In this table, ILP shows the optimum communication cost values, and it is remarkable that PFMAP also finds optimum results for given applications in a short period of time. Although NMAP and LMAP are fast algorithms, they do not find optimum results for given medium-size applications. In average, PFMAP seems to be the best algorithm among the other ones in terms of both communication cost and algorithm running time.

We have tested the performance of PFMAP on a fixed, 4 × 4 2-D regular mesh network with increasing communication demand between cores. To implement this, we have generated five task graphs with fixed number of vertices but different number of communication demands using TGFF. In Fig. 4, we see that PFMAP outperforms NMAP when communication demand increases.

Running times of NMAP and PFMAP algorithms for the networks in Fig. 4 are given in Fig. 5. Except the simplest one (i.e., graph with 14 edges), PFMAP finds a better result than NMAP with a little time overhead.

B. 2-D Irregular and Custom Mesh Architectures

We have compared PFMAP with other studies, such as NMAP, CMAP, A3MAP-SR [4], and A3MAP-GA [4] for VOPD application on some irregular 2-D mesh architectures given in Fig. 6. Here, solid lines represent links with full bandwidth, while dashed lines show the links width

TABLE III
COMMUNICATION COST OF VOPD APPLICATION ON SIX DIFFERENT IRREGULAR MESH ARCHITECTURES (4×4)

Total Communication Cost									
Network	NMAP	CMAP	A3MAP-SR	A3MAP-GA	PFMAP	Imp. Over NMAP(%)	Imp. Over CMAP(%)	Imp.Over A3MAP-SR(%)	Imp.Over A3MAP-GA(%)
Figure 6a	4215	4911	4205	4189	4189	+0.62	+14.70	+0.38	0.00
Figure 6b	8704	6544	5820	5345	4253	+51.14	+35.01	+26.92	+20.43
Figure 6c	6405	7194	6185	5257	4900	+23.50	+31.89	+20.78	+6.79
Figure 6d	4923	5507	4374	4199	4199	+14.71	+23.75	+4.00	0.00
Figure 6e	4950	4259	4191	4189	4281	+13.52	-0.52	-2.15	-2.20
Figure 6f	7424	6497	4832	4081	4351	+41.39	+33.03	+9.95	-6.62
Average	6104	5819	4935	4543	4362	+24.14	+22.98	+9.98	+3.07
Ratio	1	0.953	0.808	0.744	0.715				

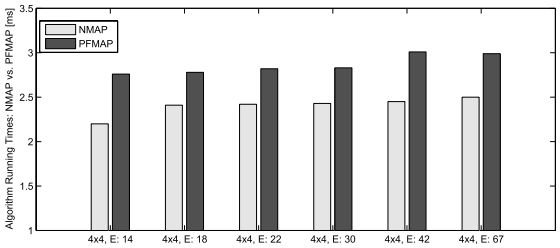


Fig. 5. Algorithm running time of NMAP and PFMAP on a 2-D NoC with fixed size (4×4) with increasing communication demand (IT = 10, PN = 10 for PFMAP).

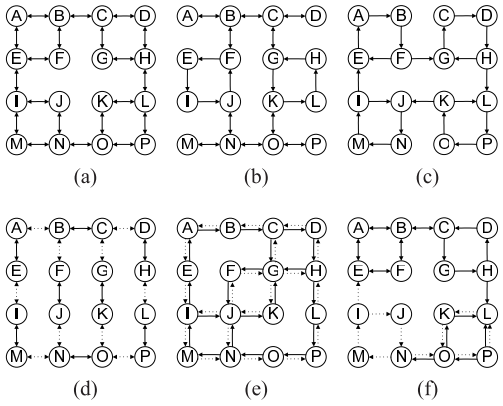


Fig. 6. Irregular mesh architectures [4]: (a) only bidirectional links, (b) both bidirectional and unidirectional links, (c) only unidirectional links, (d) both directions having the same bandwidth, (e) each direction of links with different bandwidth, and (f) links with all irregularities.

half bandwidth. PFMAP tries to place heavy weight communications onto the links with full bandwidth and the remaining smaller weight edges onto the links with half bandwidth irregular 2-D mesh topologies. As it is observed in Table III, although PFMAP finds a worse communication cost in a few scenarios (rows 7 and 8), it gives much better results than all its counterparts in average for each scenario. Even though the PFMAPs resampling algorithm works very well, it might give good results only for large number of IT and PN values for a given application. We fixed both IT and PN values to 1000 for this set of experiments. If we increase these values, the PFMAP algorithm will probably find better results with the time.

We have also compared PFMAP with other studies for VOPD application on some custom architectures given in Fig. 7. Comparison results are given in Tables IV and V.

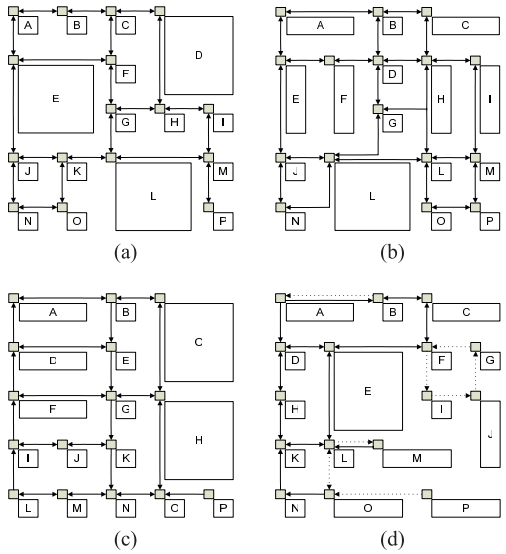


Fig. 7. Custom mesh architectures [4]: (a) three PEs having four times larger area than others, (b) PEs with three different sizes, (c) both bidirectional and unidirectional links, and (d) links with different bandwidth.

Communication cost of PFMAP is much better than other algorithms in average in all scenarios. Total travel distance of PFMAP might be worse than the other algorithms for a few scenarios (row 4 in Table V) due to fixed IT and PN values.

Hop count may not be sufficient to qualify the mapping quality of irregular and custom architectures [35]. Mapping quality also depends on the communication energy and latency for such architectures. To examine this issue, an irregular 3×3 mesh architecture and a custom 11-core architecture are given in Fig. 8. Number pairs on the edges denote relative communication energy and latency of each link, respectively. The communication latency from $Core\ i$ to $Core\ j$ is denoted by $l_{i,j}$ and obtained by the sum of relative communication latencies on the shortest path from $Core\ i$ to $Core\ j$. The total communication latency (L_{Comm}) for an application mapping is given by

$$L_{Comm} = \sum_{i=0}^{E} \sum_{j=0}^{E} t_{i,j} * l_{i,j}. \tag{11}$$

Similarly, the communication energy from $Core\ i$ to $Core\ j$ is denoted by $e_{i,j}$ and obtained by the sum of relative communication energies on the shortest path from $Core\ i$ to $Core\ j$.

TABLE IV
COMMUNICATION COST OF VOPD APPLICATION ON FOUR DIFFERENT CUSTOM MESH ARCHITECTURES (4×4)

Total communication cost									
Network	NMAP	CMAP	A3MAP-SR	A3MAP-GA	PFMAP	Imp. Over NMAP(%)	Imp. Over CMAP(%)	Imp.Over A3MAP-SR(%)	Imp.Over A3MAP-GA(%)
Figure 7a	4488	4752	4531	4087	4076	+9.18	+14.23	+10.04	+0.27
Figure 7b	4264	4119	4248	4199	3859	+9.49	+6.31	+9.16	+8.09
Figure 7c	6296	5598	5867	5150	4290	+31.86	+23.37	+26.88	+16.70
Figure 7d	5524	5735	4263	4263	4236	+23.32	+26.14	+0.63	+0.63
Average	5143	5051	4727	4425	4115	+18.46	+17.51	+11.68	+6.42
Ratio	1.000	0.982	0.919	0.860	0.800				

TABLE V
TOTAL TRAVEL DISTANCE (WIRELENGTH) BY ALL PACKETS

Total travel distance									
Network	NMAP	CMAP	A3MAP-SR	A3MAP-GA	PFMAP	Imp. Over NMAP(%)	Imp. Over CMAP(%)	Imp.Over A3MAP-SR(%)	Imp.Over A3MAP-GA(%)
Figure 7a	5879	6300	5332	4543	4108	+30.12	+34.79	+22.96	+9.58
Figure 7b	5505	4135	5049	4215	4486	+18.51	-8.48	+11.15	-6.42
Figure 7c	7835	6842	7434	5613	4862	+37.95	+28.94	+34,60	+13.38
Figure 7d	9196	9627	5170	5170	4971	+45.94	+48.36	+3.85	+3.85
Average	7104	6726	5746	4885	4606	+35.16	+31.52	+19.84	+5.71
Ratio	1.000	0.947	0.809	0.688	0.648				

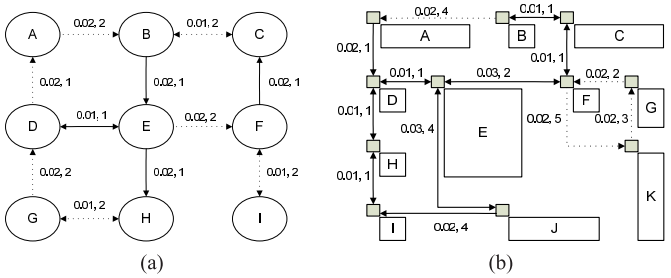


Fig. 8. Energy and latency representations of irregular and custom architectures. (a) Irregular mesh 3×3. (b) Custom with 11 cores.

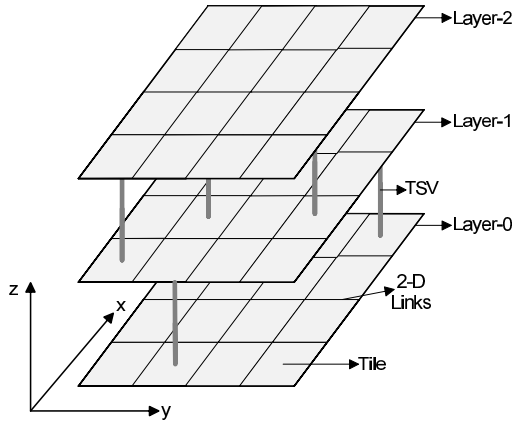


Fig. 9. 3-D NoC architecture with the size of $4 \times 4 \times 3$.

The total communication energy (E_{Comm}) for an application mapping can be calculated by

$$E_{Comm} = \sum_{i=0}^{E} \sum_{j=0}^{E} t_{i,j} * e_{i,j}. \tag{12}$$

In this set of experiment, various benchmarks with different sizes generated by TGFF are mapped onto miscellaneous irregular and custom NoC architectures similar to Fig. 8 but with different dimensions. Table VI shows the communication energy and communication latency. Here, PFMAP gives always much better results than NMAP in terms of both communication latency and energy with a small running time overhead.

C. Three-Dimensional NoCs

A 3-D NoC interconnection architecture is composed of 2-D layers connected to each other through vertical links. In Fig. 9, a 3-D NoC architecture with dimensions $X = 4$, $Y = 4$, and $Z = 3$ is given. Most attractive way to connect these layers is utilizing through silicon vias (TSVs). However, TSV pads between layers occupy significant chip area and lead to congestion delays [36]. Hence, finding a good mapping algorithm, which decreases the number of TSVs, may increase the system performance by saving chip area and reducing communication delay. From this point of view, we applied PFMAP algorithm to applications, such as AI, Telecom, DMC, MMS with 25 cores (MMS25), and MMS with 40 cores (MMS40) onto different sizes of 3-D NoCs.

Initially, we determine the 3-D NoC network dimensions according to the number of application task nodes. Given N as the number of task nodes for an application. The calculation of 3-D NoC dimensions is given in Algorithm 3.

After determining dimensions of target 3-D NoC, we assume all mutual tiles in neighbor layers are connected TSVs. Since TSVs are more costly than 2-D links, we set the cost of a TSV as five times of a 2-D link's cost. As the resampling iterations increase, we prune unused TSVs. After a burn-in period, the algorithm ends with a smallest number of TSVs of the target 3-D NoC.

TABLE VI
COMMUNICATION ENERGY AND LATENCY COMPARISON OF NMAP AND PFMAP ON BOTH IRREGULAR AND CUSTOM ARCHITECTURES

		NMAP			PFMAP					
Network	Benchmark				IT=100, PN=100			IT=100, PN=1000		
		L_{comm}	E_{comm}	$Run\ T.[ms]$	L_{comm}	E_{comm}	$Run\ T.[ms]$	L_{comm}	E_{comm}	$Run\ T.[ms]$
4x4 irregular	TGFF16	466.44	34626	2.63	395.72	28026	107	372.51	26170	173
Ratio		1	1	1	0.85	0.80	40	0.79	0.75	65
5x5 irregular	TGFF25	10438.77	838272	5.76	9173.19	715883	111	9085.77	673411	318
Ratio		1	1	1	0.87	0.85	19	0.87	0.80	55
6x6 irregular	TGFF36	156244.55	14980839	11.50	23336.87	1812894	149	21142.23	1582556	841
Ratio		1	1	1	0.14	0.12	13	0.13	0.10	73
11-core custom	TGFF11	337.83	44657	1.47	230.78	27747	84	225.23	26672	129
Ratio		1	1	1	0.68	0.62	57	0.66	0.59	87
17-core custom	TGFF17	15667.71	1675886	2.82	6232.33	665284	105	6038.62	656285	174
Ratio		1	1	1	0.39	0.39	37	0.38	0.39	61
28-core custom	TGFF28	18909.16	1992570	7.24	15934.52	1778346	128	14880.78	1508527	437
Ratio		1	1	1	0.84	0.89	17	0.78	0.75	60

Algorithm 3 Determine X, Y, and Z Dimensions for 3-D NoC

Input: Number of task nodes for an application (N)
Output: X, Y, Z for 3-D NoC design

1: **Function** getDimensions (N)
2: $X = Y = Z = \lfloor \sqrt[3]{N} \rfloor$
3: **while** $X * Y * Z < N$ **do**
4: $X++$;
5: **if** $X * Y * Z \geq N$ **then**
6: $break$;
7: **end if**
8: $Y++$;
9: **if** $X * Y * Z \geq N$ **then**
10: $break$;
11: **end if**
12: $Z++$;
13: **end while**
14: **RETURN** X, Y, Z
15: **EndFunction**

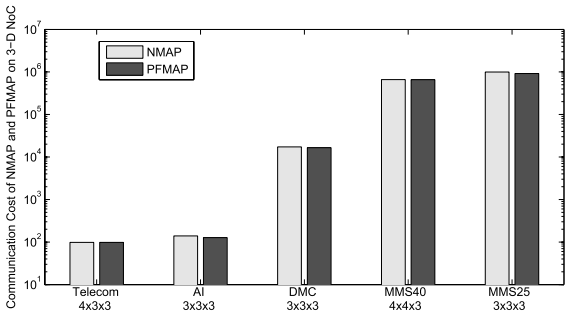


Fig. 10. Communication cost of PFMAP and NMAP on a 3-D NoC.

The communication cost of an application onto regular, irregular, or custom 3-D NoC is represented as in (6). We consider any type of a 3-D NoC as an irregular or custom 2-D NoC; we extract the distance matrix of the target 3-D NoC as explained in Fig. 2. Then, we calculate the communication cost of a current configuration using (6).

Fig. 10 shows the communication costs of PFMAP and NMAP algorithms on different sizes of 3-D NoCs for various real life applications. Here, each application is independent from each other. However, it is remarkable that PFMAP tends to give much better results than NMAP when the density of the corresponding task graph is high.

TABLE VII
COMMUNICATION ENERGY COMPARISON OF NMAP AND PFMAP ON 3-D NOCS

	NMAP		PFMAP: IT=100, PN=100	
Benchmark	E_{comm3D}	$Run\ T.[ms]$	E_{comm3D}	$Run\ T.[ms]$
TGFF3x3x3	5190	14.35	4767	109
TGFF4x4x4	10820	45.84	10319	382
TGFF5x5x5	24937	713	22950	3027
TGFF6x6x6	53463	2660	44538	16477
Ratio	1	1	0.87	5.82

Mapping on 3-D NoCs is also done according to communication energy consumption. For this purpose, energy model given in [37] is used. Here, the average energy consumption of sending one bit of data from $tile\ t_i$ to $tile\ t_j$ is represented as follows:

$$E_{\text{bit}}^{t_i,t_j} = nE_{R\text{bit}} + n_H E_{LH\text{bit}} + n_V E_{LV\text{bit}} \tag{13}$$

where n is the number of routers, n_H number of horizontal links, and n_V number of vertical links, all passed by packets. n, n_H, and n_V change with the mapping. $E_{R\text{bit}}$ is the energy consumed by a router, and $E_{LH\text{bit}}$ and $E_{LV\text{bit}}$ are the energy consumed on the horizontal and vertical links. All $E_{R\text{bit}}$, $E_{LH\text{bit}}$, and $E_{LV\text{bit}}$ values are technology dependent; they can be used as constants as in [37]. We define $E_{comm3\text{-D}}$ as the sum of all communicating node pairs with the communication energy consumption of E^{t_i,t_j} in a benchmark.

In Table VII, we compared NMAP and PFMAP for four 3-D NoCs with different sizes. Even with a small number of IT and PN values, PFMAP outperforms NMAP algorithm.

D. Large-Scale NoCs

For large-scale NoCs, we have observed that creating initial configurations randomly (line 5 in Algorithm 1) causes to increase in IT and PN values to find a good solution. Instead of using pure random initial configurations for both large-scale 2-D and 3-D NoCs, we apply an initialisation step as in NMAP. The pseudocode for our initialize method is given in Algorithm 4.

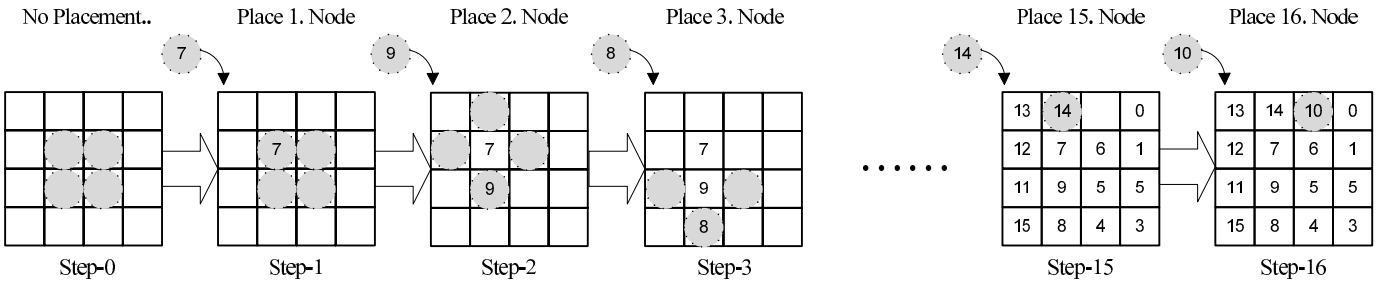


Fig. 11. PFMAP initialization steps for large-scale NoCs.

Algorithm 4 Initialization Function

Input: Set of cores and nodes, TTG, CTG, RCT
Output: Mapping of cores to nodes

1: **Function** initialConf(RCT(R,C))
2: *Select node* N_{max} *with max. comm. demand*
3: *Select one of processor* P_{max} *with max. connection link*
4: *Map* N_{max} *onto* P_{max}
5: **while** *no more node to be mapped* **do**
6: N_{max} = *one of most comm. nodes with placed nodes*
7: P_{max} = *one of procs. which causes min. comm. cost*
8: *Map* N_{max} *onto* P_{max}
9: **end while**
10: **RETURN** $modifiedRCT$
11: **EndFunction**

The main difference between NMAPs initialization method and ours is that in each step we select the placement of a node randomly among best candidates with equal costs. In NMAP, the selection operation always finds the same best location. As we find different best candidates with equal cost, we run *Initialization Function* as the number of particles times. Thus, we are able to create cost efficient initial configurations. In Fig. 11, the initialization steps of a configuration for VOPD application are represented. Yet, we apply initialization only for large-scale NoCs. According to Algorithm 4: in the first step (line 2), we select the task node with maximum communication demand (node 7 in Fig. 11). Then for the placement, one of the best candidate locations is selected randomly (line 3). The best candidate locations for node 7 are shown as shaded circles in step-0 in Fig. 11. For each initial configuration (i.e., particle), we select one of these best candidate locations randomly. In each step of the initialization phase, there might multiple choices, which results in different configurations. For example, in step-3, while for one configuration the best location for node 8 is selected as the bottom neighbor of the node 9, for a different configuration it can be selected as right/left neighbor of the node 9. By selecting best candidates randomly in this way, we might come up with different configurations. These configurations form our initial configuration set. After obtaining initial configuration set for the given number of particles (first iteration in Algorithm 1), we can apply systematic resampling on this set for the given number of iterations.

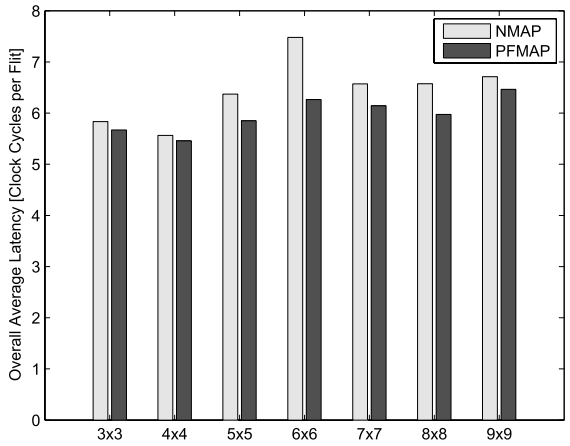


Fig. 12. Average network latency comparison of NMAP and PFMAP for different sizes of synthetic task graphs.

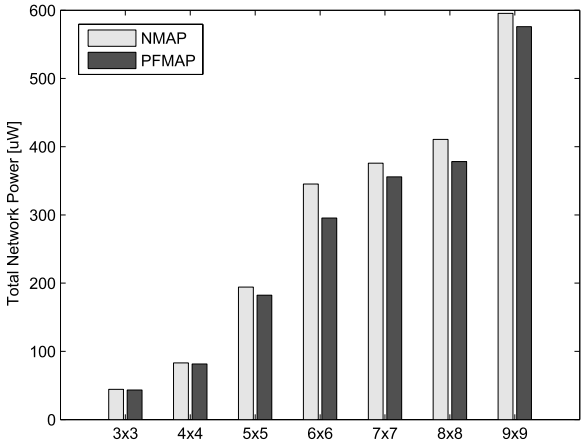


Fig. 13. Total network power comparison of NMAP and PFMAP for different sizes of synthetic task graphs.

We have evaluated the scalability of our PFMAP algorithm on fully synthetic task graphs (generated by TGFF) with various NoC sizes from 3×3 to 9×9 for 2-D regular mesh networks. In Figs. 12 and 13, timing and power results of seven different synthetic task graphs are presented. These synthetic task graphs are mapped onto regular 2-D mesh architectures using NMAP and our PFMAP algorithm. As already mentioned, we have used NIRGAM NoC simulator for the simulation of each mapping.

As is evident from Figs. 12 and 13, PFMAP gives lower average packet latency and total power than NMAP independent from the network size. As the network size increases,

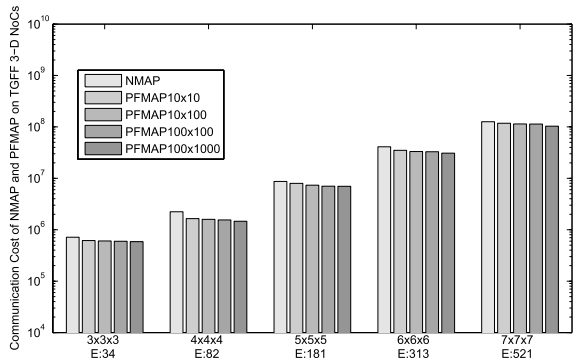


Fig. 14. Communication cost of PFMAP and NMAP algorithms on 3-D NoC for different sizes of TGFF applications.

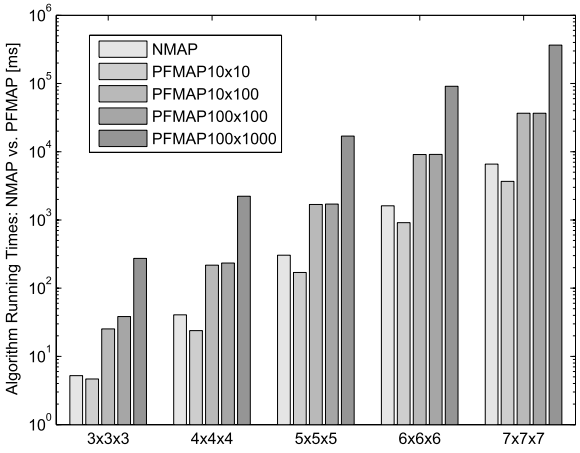


Fig. 15. Running time of NMAP and PFMAP algorithms with different sizes of IT and PN for different sizes of TGFF applications.

the number of generated packets will increase proportionally. Hence, the gap between network delays of NMAP and PFMAP will rise up significantly. As a result of this, the gap between energy consumptions of NMAP and PFMAP will also increase as the network is getting larger. Finally, we may come up with the result that PFMAP is more scalable than NMAP, since it gives much better results in terms of both total latency and energy consumption as the network getting larger.

In our final set of experiments, we have examined the scalability factor of our PFMAP algorithm on 3-D NoCs. In this set of experiments, we have also used fully synthetic task graphs (generated by TGFF) with different number of task nodes (i.e., 27–343) and edge weights (i.e., 34–521), as we did for 2-D NoCs. In Fig. 14, communication costs of both PFMAP and NMAP algorithms on 3-D NoC for different sizes of TGFF applications are available. Here, we also compare PFMAP with itself by setting different ITs and PNs. For example, PFMAP 10 $\times$ 100, given in Fig. 14, means corresponding PFMAP solution is found with 10 iterations (IT) and 100 particles (PN). As it is obvious from Fig. 14, all PFMAP solutions give much better results than NMAP in any network size. It is also clear that communication cost of PFMAP decreases with increasing IT and PN for any network size. The main point here is that PFMAP outweighs NMAP even with a very low IT and PN values in all network sizes.

The corresponding solution finding times of applications in Fig. 14, are shown in Fig. 15. It is definite that running times of both NMAP and PFMAP algorithms increase as the problem size getting larger. Similarly, running time of PFMAP algorithm increases for larger IT and PN values in any network size. Although NMAP is a very fast heuristic algorithm, PFMAP10 $\times$ 10 runs faster than NMAP while giving better results than it in all network sizes. We cannot deny that NMAP is a very fast algorithm, but the running time of PFMAP algorithm actually depends on the designer. PFMAP finds a solution in 500 s for a huge application with 343 nodes and 541 edges, which gives a better result than NMAP by 20%.

In addition to these, either running time of PFMAP can be reduced or IT and PN values can be increased to find a better mapping solution using parallel platforms such as GPU.

V. Conclusion

In this paper, we have discussed mapping of task cores onto the nodes of regular, irregular, and custom tile-based 2-D and 3-D NoC architectures. To solve mapping problem on tile-based NoCs, we have utilized systematic resampling algorithm for particle filters. To the best of our knowledge, we are the first, who apply this algorithm to solve mapping problem on NoC architectures.

According to the various experimental results, our proposed algorithm, PFMAP, gives better results than aforementioned studies. We have also given the mathematical representations and definitions for the developed algorithm.

Timing results show that PFMAP is able to find an optimum or near optimum solution in a few milliseconds for medium size commercial applications. We also show that PFMAP is a suitable algorithm for mapping on any type tile-based NoC architectures such as regular, irregular, and custom 2-D or 3-D topologies. We have proven that PFMAP is scalable enough to solve mapping problem on huge networks.

Since there is no data dependency between the particles, we applied multithread approach to the parallel running particles. By exploiting C++ OPENMP library, we inserted thread-level parallelism to our mapping algorithm. We have already pointed out that the quality of mapping accuracy increases with the available computational resources. Moreover, particles can run in parallel and are very suitable for parallel computation platforms, such as GPU and VPU. As a result of these, the heuristic PFMAP algorithm can run much faster on fully parallel platforms and therefore gives better results in shorter times.

Acknowledgment

The authors would like to express our very great appreciations to D. Dinç for his valuable help in the development of systematic resampling algorithm. Advice given by Dr. C. Çelik and Dr. S. Tosun have been a great help in modifying the NIRGAM NoC simulator.

References

[1] H. G. Lee, N. Chang, Ü. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Design Autom. Electron. Syst.*, vol. 12, no. 3, pp. 1–23, Aug. 2007.

[2] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, vol. 2. Feb. 2004, pp. 896–901.

[3] W. Jang and D. Z. Pan, "A3MAP: Architecture-aware analytic mapping for networks-on-chip," in *Proc. 15th ASP-DAC*, Jan. 2010, pp. 523–528.
[4] W. Jang and D. Z. Pan, "A3MAP: Architecture-aware analytic mapping for networks-on-chip," *ACM Trans. Design Autom. Electron. Syst.*, vol. 17, no. 3, pp. 1–26, Jun. 2012.
[5] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *J. Syst. Archit.*, vol. 59, no. 1, pp. 60–76, Jan. 2013.
[6] F. Gustafsson *et al.*, "Particle filters for positioning, navigation, and tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 425–437, Feb. 2002.
[7] S. Thrun, "Particle filters in robotics," in *Proc. 18th Conf. Uncertainty Artif. Intell.*, 2002, pp. 511–518.
[8] I. M. Rekleitis, "A particle filter tutorial for mobile robot localization," Centre Intell. Mach., McGill Univ., Montréal, QC, Canada, Tech. Rep. TR-CIM-04-02, 2004.
[9] A. Doucet, N. De Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. New York, NY, USA: Springer-Verlag, 2001.
[10] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc. F, Radar Signal Process.*, vol. 140, no. 2, pp. 107–113, Apr. 1993.
[11] R. Douc and O. Cappé, "Comparison of resampling schemes for particle filtering," in *Proc. 4th Int. Symp. Image Signal Process. Anal. (ISPA)*, Sep. 2005, pp. 64–69.
[12] G. Kitagawa, "Monte Carlo filter and smoother for non-Gaussian nonlinear state space models," *J. Comput. Graph. Statist.*, vol. 5, no. 1, pp. 1–25, Mar. 1996.
[13] J. D. Hol, T. B. Schon, and F. Gustafsson, "On resampling algorithms for particle filters," in *Proc. IEEE Nonlinear Statist. Signal Process. Workshop*, Sep. 2006, pp. 79–82.
[14] N. Koziris, M. Romesis, P. Tsanakas, and G. Papakonstantinou, "An efficient algorithm for the physical mapping of clustered task graphs onto multiprocessor architectures," in *Proc. 8th Euromicro Workshop Parallel Distrib. Process.*, Jan. 2000, pp. 406–413.
[15] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proc. ASP-DAC*, Jan. 2003, pp. 233–239.
[16] S. Murali and G. De Micheli, "SUNMAP: A tool for automatic topology selection and generation for NoCs," in *Proc. 41st Design Autom. Conf.*, Jul. 2004, pp. 914–919.
[17] F. Moein-Darbari, A. Khademzade, and G. Gharooni-Fard, "CGMAP: A new approach to network-on-chip mapping problem," *IEICE Electron. Exp.*, vol. 6, no. 1, pp. 27–34, 2009.
[18] M. Janidarmian, A. Khademzadeh, and M. Tavanpour, "Onyx: A new heuristic bandwidth-constrained mapping of cores onto tile-based network on chip," *IEICE Electron. Exp.*, vol. 6, no. 1, pp. 1–7, 2009.
[19] S. Saeidi, A. Khademzadeh, and F. Vardi, "Crinkle: A heuristic mapping algorithm for network on chip," *IEICE Electron. Exp.*, vol. 6, no. 24, pp. 1737–1744, 2009.
[20] L. Zhong, J. Sheng, M. Jing, Z. Yu, X. Zeng, and D. Zhou, "An optimized mapping algorithm based on simulated annealing for regular NoC architecture," in *Proc. IEEE 9th Int. Conf. ASIC (ASICON)*, Oct. 2011, pp. 389–392.
[21] S. Tosun, O. Ozturk, and M. Ozen, "An ILP formulation for application mapping onto network-on-chips," in *Proc. Appl. Inf. Commun. Technol. (AICT)*, Oct. 2009, pp. 1–5.
[22] P. K. Sahu, N. Shah, K. Manna, and S. Chattopadhyay, "A new application mapping algorithm for mesh based network-on-chip design," in *Proc. Annu. IEEE India Conf. (INDICON)*, Dec. 2010, pp. 1–4.
[23] P. K. Sahu, P. Venkatesh, S. Gollapalli, and S. Chattopadhyay, "Application mapping onto mesh structured network-on-chip using particle swarm optimization," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2011, pp. 335–336.
[24] R. P. Dick. (2013). *Embedded System Synthesis Benchmarks Suites (E3S)*. [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s
[25] R. P. Dick, D. L. Rhodes, and W. Wolf. (1998). *TGFF: Task Graphs for Free*. [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/tgff
[26] E. B. van der Tol and E. G. T. Jaspers, "Mapping of MPEG-4 decoding on a flexible architecture platform," *Proc. SPIE*, vol. 4674, pp. 362–363, Dec. 2002.
[27] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.
[28] R. A. Fisher and F. Yates, *Statistical Tables for Biological, Agricultural and Medical Research*, 3rd ed. London, U.K.: Oliver & Boyd, 1948.
[29] M. Saito and M. Matsumoto, "SIMD-oriented fast Mersenne twister: A 128-bit pseudorandom number generator," in *Monte Carlo and Quasi-Monte Carlo Methods*, A. Keller, S. Heinrich, and H. Niederreiter, Eds. Berlin, Germany: Springer-Verlag, 2008, ch. 36, pp. 607–622.
[30] X. Tian and K. Benkrid, "Mersenne twister random number generation on FPGA, CPU and GPU," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul./Aug. 2009, pp. 460–464.
[31] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.
[32] OpenMP Architecture Review Board. (May 2008). *OpenMP Application Program Interface Version 3.0*. [Online]. Available: http://www.openmp.org/mp-documents/spec30.pdf
[33] *Nirgam (V2.0)*. [Online]. Available: http://www.nirgam.ecs.soton.ac.uk, accessed Dec. 20, 2013.
[34] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 4, pp. 551–562, Apr. 2005.
[35] O. He, S. Dong, W. Jang, J. Bian, and D. Z. Pan, "UNISM: Unified scheduling and mapping for general networks on chip," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1496–1509, Aug. 2012.
[36] C. Liu, L. Zhang, Y. Han, and X. Li, "Vertical interconnects squeezing in symmetric 3D mesh network-on-chip," in *Proc. 16th Asia South Pacific Design Auto. Conf.*, Jan. 2011, pp. 357–362.
[37] X.-H. Wang, P. Liu, M. Yang, M. Palesi, Y.-T. Jiang, and M. C. Huang, "Energy efficient run-time incremental mapping for 3-D networks-on-chip," *J. Comput. Sci. Technol.*, vol. 28, no. 1, pp. 54–71, Jan. 2013.

Salih Bayar received the B.S. degree in electronics and communication engineering from Yıldız Technical University, Istanbul, Turkey, in 2003 and the M.S. degree in electrical engineering from the Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2007. He is currently pursuing the Ph.D. degree with Boğaziçi University, Istanbul.

He was a Research Assistant with Boğaziçi University from 2007 to 2013. His current research interests include reconfigurable computing, multiprocessor and embedded multicore architectures, system-on-chip, and network-on-chip.

Arda Yurdakul (S'91–M'99) received the B.S., M.S., and Ph.D. degrees in electrical and electronics engineering from Bogazici University, Istanbul, Turkey, in 1992, 1994, and 1999, respectively.

She is currently an Associate Professor with the Department of Computer Engineering, Bogazici University. Her current research interests include design of novel architectures and accelerators, embedded systems, reconfigurable computing, system-level and high-level design automation, and Internet-of-Things.

Dr. Yurdakul served as an IEEE Turkey Section Chair from 2010 to 2011.