

PUMA: Placement Unification with Mapping and guaranteed throughput Allocation on an FPGA Using A Hardwired NoC

Muhammad Aqeel Wahlah¹ and Kees Goossens²

¹ *Computer Engineering, Delft University of Technology, Delft, The Netherlands*
¹ aqeel@ce.et.tudelft.nl

² *Electronic Systems, Eindhoven University of Technology, Eindhoven, The Netherlands*
² k.g.w.goossens@tue.nl

Abstract—Platform-based Field Programmable Gate Arrays (FPGAs) have gained popularity for implementing multiprocessor system on chips (MPSoCs). The applications in an MPSoC can have high complexities and stringent Quality-of-Service (QoS) demands. Consequently, the problem of binding an application on an FPGA has become more challenging. An application requires logic and communication resources for computing and transporting data among its IPs. This in turn divides an FPGA into two virtual planes, i.e., logic and communication. Therefore, the available resources in both the FPGA planes should be taken into account by an application binding solution.

Our proposed scheme performs placement unification with mapping and allocation (PUMA). This means PUMA accounts for the required (application) to the available (FPGA) resources in both the logic plane and the communication plane, simultaneously. A hardwired Network on Chip (HwNoC) serves as the communication plane for our FPGA, because of its scalable and isolated nature. Moreover, PUMA ensures that a successful binding solution fulfills an application QoS constraints.

PUMA is implemented by using cycle-accurate transaction-level SystemC. PUMA performance and scalability is evaluated by using a number of synthetic applications. The PUMA application binding success rate exists in between 35% and 90%. Additionally, the cost of PUMA is evaluated against a real-world *H.264* encoder.

Keywords-FPGA; Hardwired NoC; Mapping; Allocation

I. INTRODUCTION AND MOTIVATION

Field Programmable Gate Arrays (FPGAs) [19] due to their fast time-to-market, low non-recurring engineering (NRE) costs, and in-field product upgrades have gained popularity for multiprocessor system on chips (MPSoCs) [9], [10]. The FPGA-based MPSoCs can execute large number of computation- and communication-intensive applications by using a scalable communication architecture, e.g., *soft*¹ Network-on-Chip (NoC) [9]. In recent years, a *hard*² variant, i.e., a hardwired NoC (HwNoC) [4], [6] has also been proposed. Before proceeding further, we define terminology.

An application is said to be: i) *placed* when its intellectual properties (IPs) are placed on an FPGA logic plane, ii) *mapped* when its IP ports are connected to an FPGA communication plane, iii) and allocated when its IPs can communicate with each other as per QoS constraints. In concise we term the whole process of placement, mapping, and allocation as *binding*.

¹A *soft* IP is implemented on an FPGA reconfigurable blocks, e.g., lookup tables (LUTs).

²An IP is *hard* when it is implemented in silicon, e.g., PowerPC.

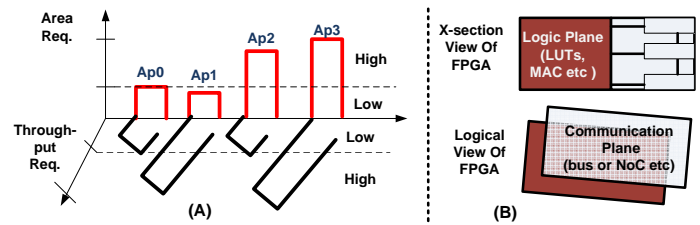


Figure 1. (A) Possible Application Categories, (B) FPGA With Two Views

An application requires logic and communication resources for computing and transporting data among its IPs. Moreover, applications depending upon their area and throughput demands can be abstracted into four categories, as shown in Figure 1A. Therefore, an FPGA is logically divided into two planes (communication and logic) to fulfill an application requirements, Figure 1B. This in turn raises the need for a solution that can unify all the three processes, i.e., placement, mapping, and allocation for executing an application on an FPGA. On the contrary, a binding solution (as we shall explain in Section I-A) that decides on the basis of resource availability across a single FPGA plane, can cause reconfiguration and/or degraded performance for the input application. Even though an FPGA initially had enough resources across both of its planes. For such a solution, the probability of an unsuccessful binding could even increase when an application has stringent constraints against even a single FPGA (logic / communication) plane.

A. Motivational Case Study

Let us motivate the need for a unified placement, mapping and allocation scheme by means of a simple case study. We have an application to be bound to two partial reconfigurable regions (PRRs), Figure 2A. The PRRs are connected to a NoC. Single or multiple IPs can coexist in a single PRR, provided these don't exceed a PRR area, i.e., 100 LUTs. On the other hand an NoC link, which provides a maximum of 2000 MB/s bandwidth, can be shared by multiple applications. In our case study, 40% of the R0-R1 link is utilized by other existing applications (not shown for the simplicity) of the system. Therefore, the R0-R1 link has a spare bandwidth of 1200 MB/s, for our *test* application.

As shown in Figure 2A, the test application comprises an IP *W*, which serves as source for the remaining three IPs, i.e., *X*, *Y*, and *Z*. As a starting point, we place *W* in PRR0. As shown in Figure 2 (B, C, and D), three different solutions are

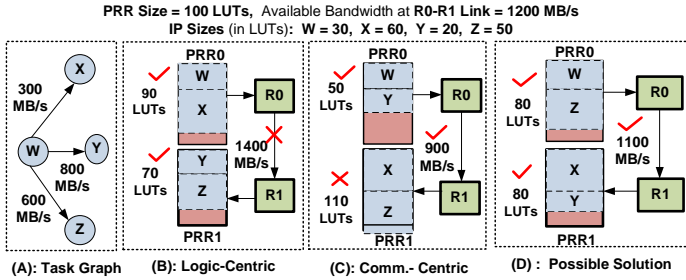


Figure 2. Motivational Case Study

possible to select the next IP. Figure 2B shows a logic-centric binding solution, which opts for W and X in the same PRR0 for producing least possible fragmentation in it. Y , and Z are placed afterwards in PRR1. However, it can cause an application to under perform, because W does not get the required throughput on R0-R1 link for communicating with Y and Z . Figure 2B shows a communication-centric binding solution, which opts for W and Y in the same PRR0 for restricting highest throughput connection from going to network, i.e., to the shared R0-R1 link. However, due to excessive fragmentation in PRR0, binding of the complete test application is not possible in this case.

Figure 2D shows a possible successful binding solution for the test application. Here, before selecting any IP that is placed next to W , the temporal and the spatial constraints (of both application and FPGA) are evaluated. However, doing so is critically complex, when noticed that the logic-centric binding alone is an NP-hard problem [12]. The problem becomes more complex, when an application exhibits: a) higher number of IPs with increased dependencies, b) and stringent QoS constraints.

B. Our Contributions

Our PUMA scheme performs application binding by accounting the required (application) to the available (FPGA) resources across both logic and communication planes, simultaneously. More precisely, PUMA scheme contributes towards solving an application to architecture binding problem by:

- 1) using a hardwired NoC as the communication plane,
- 2) minimizing an application dependencies by decomposing it into fully / partially independent clusters,
- 3) transforming placement, mapping and allocation into a single problem, while selecting an FPGA region,
- 4) including temporal (time-slots), and spatial (logic CLBs, communication links) constraints, as an FPGA region selection objective function.

In the remainder of this paper we position ourself in Section II, formulate the application to FPGA binding problem in Section III, illustrate PUMA scheme to solve the problem in Section IV, provide results and analysis in Section V, and conclude at the end in Section VI.

II. RELATED WORK

In the literature different application binding solutions can be found [2], [5], [7], [8], [9], [10], [12]. Table I shows that the researches on the basis of their techniques can be categorized

Table I
OUR WORK POSITIONING W.R.T. EXISTING APPROACHES

Approach	Scheme	QoS
Unified Place and Map	[1], [2], [12]	Y
Unified Map and Allocate	[5], [9], [13]	Y
Non-Unified	[7], [8], [10]	N
Unified Place, Map, and Allocate	Our PUMA Scheme	Y

as: i) unified place and map, ii) unified map and allocate, iii) and non unified,

In [1], a physical planner is used during topology design to reduce power consumption on wires. However, number and size of network partitions are manually fed. Authors in [2] treat each IP as a component which encapsulates a circuit implemented with the resources in a given area (routers logic and IP). Importantly, after an IP is placed, the IP coordinates are set to that of its router. The work in [12] takes into account the physical planning issues, while mapping an application IP on the communication plane, i.e., a network-on-chip. For this purpose a floor-planner is used during the mapping process to get area and wire-length estimates. In *unified place and map schemes* [1], [2], [12], the placement of IPs, due to physical planning, will induce lower logic fragmentation. However *towards limitations* the binding techniques of [1], [2], [12] could suffer through long routes with no analytical bounds over the QoS guarantees. This is mainly because of the allocation phase, which is disintegrated from place and map phases.

The works in [5], [9], [13] apply unified mapping and allocation for ensuring QoS guarantees. The authors in [5], [9] incorporate application binding into path selection, while aiming to minimize the over-allocation of the network. This is done on communication flow basis. The authors in [13] analyse the average of heterogeneous NoC network latency in terms of the queuing latency. They apply the branch-and-bound algorithm to find out the QoS aware mapping that automatically maps IPs onto NoC architecture. In these works [5], [9], [13], mapping and allocation are tightly integrated. Therefore, an IP ports are mapped only and only if, resources for its communication flows are guaranteed over the NoC. However *towards limitations* placement can only be performed once mapping and allocation are done. However, during mapping and allocation the available logic area against a communication node is not taken into account. These schemes can be applied to an FPGA, but after assuming that sufficient logic resources are available next to the communication node, which is not pragmatic.

Authors in [7] propose a branch and bound based scheme that binds the IPs onto the network with an objective of minimum communication energy. The work in [8] uses two heuristics, a fast successive relaxation and a genetic algorithm to find mapping over network with irregular topology. Authors in [10] map an application onto regular mesh NoC by using a genetic algorithm, with an objective of minimized execution time. In these *non unified schemes* [8], [10] real-time guarantees on an application QoS requirements are not ensured. However, in [7] QoS constraints are fulfilled, but after iteratively refining mapping and routing for an application IPs.

In our comparison with above-mentioned works, our proposed PUMA scheme unifies all three placement, mapping, and allocation phases. The PUMA scheme, ensures that for a successful application to FPGA binding, an application QoS constraints are fulfilled. At compile time, our PUMA scheme performs application binding, after exploiting available resources across both the FPGA planes, simultaneously. This means the temporal and the spatial constraints of an application are considered simultaneously, while selecting an FPGA region. We use HWNoC as our FPGA communication plane, because in comparison with a soft NoC, the HWNoC: i) does not occupy an FPGA logic area, ii) exhibits higher scalability, iii) and posses 148 times better cost:performance ratio. However, PUMA is not hardwired NoC specific, and can be implemented by using alternative soft [9], [15] or hard [11] communication architectures. However, in the target FPGA architecture both the planes (communication and the logic/computation) should be logically/physically disjoint at the time of binding.

III. PROBLEM FORMULATION

We now formulate the problem of unified placement, mapping and allocation.

A. FPGA Specifications

Definition 1: An FPGA with a hardwired NoC [4], is shown in Figure 3A. We can formulate it as $FPGA = (N, L)$. Here N stands for FPGA nodes (Fnodes), and L stands for the physical links that are used by Fnodes to transport data in between them.

Definition 2: Each link $l \in L$ has a slot table $t(l)$ of size S . The residual bandwidth (in terms of discrete time-slots) of an $l \in L$ is denoted with $\gamma(l) \in \mathbb{N}$. The source and the destination ends of $l \in L$ are denoted as $s(l)$ and $d(l)$, respectively.

Definition 3: Each $n \in N$ is further decomposed into three nodes, i.e., n_{CFR} , n_{NI} , and n_R . Here, n_{CFR} represents the logic node in n . Its is a configuration functional region with 32 minimum reconfiguration regions (MCRs), where each MCR equals to 128 LUTs [20]. The residual n_{CFR} area is termed as $\bar{A}(n)$. The detailed architecture of a CFR is explained in [18].

On the other hand, n_R and n_{NI} represent the network nodes in n . The residual ports in n_{NI} can be defined as $\bar{P}t(n)$. Moreover, $\forall n \in N$, n_{NI} and n_R have one-to-one mapping. Hence, are connected through exactly one egress and one ingress links, which are defined as $l_E(n_{NI}) \in L$ and $l_I(n_{NI}) \in L$, respectively. Since, two Fnodes share data by using their routers. This means a link $l_{j,k} \in L$ between $n_j \in N$ and $n_k \in N$, represents a link in between $n_{Rj} \in n_j$ and $n_{Rk} \in n_k$.

We illustrate the above formulation by using an **example** architecture, Figure 3B. It shows an FPGA with 4 Fnodes. Here $Fnode_0$ and $Fnode_3$ are expanded for illustration. Each Fnode comprises a router, a CFR (32 MCRs), and an NI (2 port). Allocated slots (shown with dark color) across the links are also shown. Slot tables for NI0 egress and NI3 ingress links are also shown. This shows that each FPGA link has a slot table of size 5. For simplicity all the links in between the Fnodes are now shown, e.g., $Fnode_1$ to $Fnode_0$ link is missing in Figure 3B. The architectural formulation involves only those details, which could be required by our PUMA scheme.

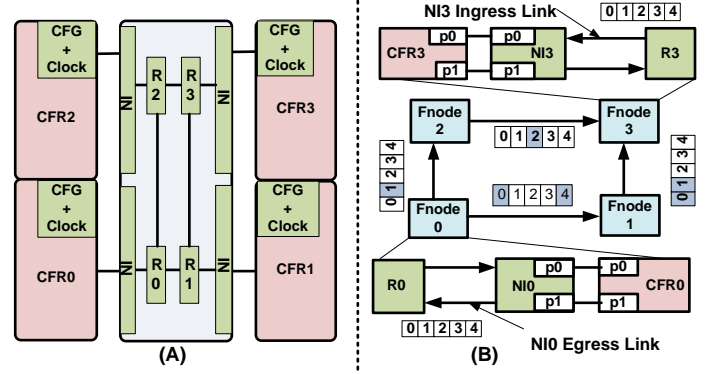


Figure 3. FPGA With HWNoC : A) Architecture, (B) Resource Formulation

B. Application Specifications

Definition 4: An application task graph is a directed graph, defined as $G = (P, C)$. Here P indicates computational IP cores, and C represent the communication connections among the IPs. The nature of our application is streaming, i.e., it is throughput sensitive and does not have strict latency constraints. Additionally, we assume a connected task graph for an application.

Definition 5: Application computation is carried out by a set of IP cores, where each $ip_i \in P$ is annotated by its area and ports, i.e., $ip_i = (a_i, pt_i)$. Here a_i stands for area, and pt_i for a port vector to indicate the ports of an ip_i .

Definition 6: Application communication $\forall ip \in P$, is performed over a set of connections C . Since, in our application model, two IPs don't have more than one connection in between them. Therefore, a connection in between ip_s and ip_d can be termed as $c_{s,d} \in C$, i.e., without specifying the ports that are used to exchange the data in between them.

Definition 7: Application connection $c \in C$ has a minimum throughput demand $b(c) \in \mathbb{R}$, converted into discretised slots, i.e., $\alpha(c) \in \mathbb{N}$. Additionally, each $c \in C$ requires a path, $\pi(c)$ to exchange data in between the IPs that belong to it.

Definition 8: Connection path $\pi = seq Links$, from source $ip_s \in P$ to destination $ip_d \in P$ across a $c_{s,d} \in C$, can be defined as the nonempty sequence of links (l_0, l_1, \dots, l_n) . Here $l_0 = l_E(n_{NI}) \in L$, $l_n = l_I(n_{NI}) \in L$, and $(l_1, \dots, l_{n-1}) \in L$ represent links between Fnode routers. Additionally, $s(l_0) = ip_s$ and $d(l_n) = ip_d$.

C. Required Objectives

A successful application binding on an FPGA is defined as:

$$bind : P \mapsto N, \text{ s.t. } bind(ip_i = n_j), \forall ip_i \in N, \exists n_j \in N \quad (1)$$

The $bind$ function accounts for the placement, mapping and allocation of all application IPs. This means an ip_i is said to be placed in $n_{jCFR} \in n_j$, when ip_i area constraint are met, i.e.,

$$place : a_i \leq \bar{A}(n_j). \quad (2)$$

The ip_i is mapped on $n_{jNI} \in n_j$, when n_{jNI} has enough ports for connecting ip_i ports. In addition, $n_{jNI} \in n_j$ should have enough bandwidth at its ingress and egress links for the incident c_{ink} , and outgoing c_{outk} connections of the ip_i , respectively.

$$mapPorts : sizeof(pt_i) \leq sizeof(\bar{P}t(n_j)) \quad (3)$$

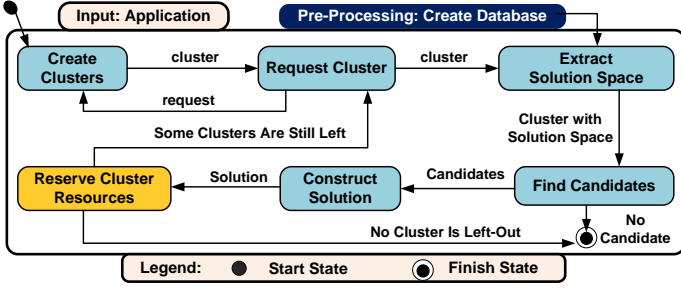


Figure 4. High Level Flow Of Our PUMA Scheme

$$\text{mapInConns} : \gamma l_I(n_{NI}) \geq \sum_{k=0}^{k=n} \alpha(c_{ink}) \quad (4)$$

$$\text{mapOutConns} : \gamma l_E(n_{NI}) \geq \sum_{k=0}^{k=m} \alpha(c_{outk}) \quad (5)$$

An ip_i is said to be allocated, when by binding it to n_j , the ip_i connections are allocated. For **example**, if the connection between ip_i and ip_k is denoted as $c_{i,k} \in C$, which requires αc_{ik} slots and follows a $\pi(c_{ik})$, *definition 7*. The path $\pi(c_{ik})$ comprises (l_0, \dots, l_n) links, *definition 8*. Here, each link can have $\gamma(l)$ residual slots, *definition 2*. Then, to perform a successful allocation for a $c_{i,k} \in C$ following equation should be satisfied:

$$\text{allocate} : (\gamma(l_0), \dots, \gamma(l_n)) \geq \alpha(c_{ik}) \quad (6)$$

However, it is important that the slots across the path-links should be assigned in a pipelined fashion i.e.

$$\text{IF } slot_s \in t_{l_0} \text{ THEN } (slot_{s+1} \% |S|) \in t_{l_1} \quad (7)$$

The next section expounds our PUMA scheme for application to FPGA binding in accordance with the above formulation.

IV. PUMA: ROAD TO UNIFIED PLACEMENT, MAPPING AND ALLOCATION

Figure 4, illustrates the working principle of our PUMA scheme. Initially, a database is created that reflects the availability of FPGA residual resources across both of its planes, Section IV-A. The database is contacted/updated, during an application binding process. Afterwards, PUMA creates clusters for the input application, Section IV-B, where each cluster represents inter-communication dependencies across a group of IPs. In Figure 6C, CL1 represent one such cluster. Then, based on the area and the communication demands of a cluster, its solution space is obtained as explained in Section IV-C. The solution space is obtained to find out the perspective candidate solutions, Section IV-D. The construction of the best solution comes out next, which is evaluated in accordance with a given cost matrix, Section IV-E. In case of a successfully constructed solution, the cluster binding is performed by reserving the (constructed solution) resources across both the FPGA planes, Section IV-F. Our PUMA flows then, depending upon the remaining clusters, either requests the next cluster or terminates the flow for the input application. We now explain the different steps of our PUMA scheme, individually.

INPUTS: t_{agg}	0	1	2	3	4	Path0: NIO-R0, R0-R1, R1-R3, R3-NI3					
1) t_{NIO-R0} (Shift Left by 0) :	0	1	2	3	4	5) t_{R1-R3} (Shift Left by 2) :	0	1	2	3	4
2) $t_{agg} = t_{agg} + t_{NIO-R0}$	0	1	2	3	4	6) $t_{agg} = t_{agg} + t_{R0-R1}$	0	1	2	3	4
3) t_{R0-R1} (Shift Left by 1) :	0	1	2	3	4	7) t_{R3-NI3} (Shift Left by 3) :	0	1	2	3	4
4) $t_{agg} = t_{agg} + t_{R0-R1}$	0	1	2	3	4	8) $t_{agg} = t_{agg} + t_{R3-NI3}$	0	1	2	3	4
OUTPUT: Aggregated Slot Table For Path0 (t_{agg}) :	0	1	2	3	4						

Figure 5. Finding Affective Throughput On Path0

A. Preprocessing: Database Creation

In our PUMA scheme, initially a database for the available FPGA resources, is constructed. For explaining the database creation process, Figure 3B is used as the reference. First, the available resources across all the FPGA nodes (Fnodes) are extracted. For each Fnode, this accounts for the available: i) MCRs in its CFR, ii) ports in its NI, iii) and bandwidth across the NI ingress and egress links. Afterwards, paths are constructed in between all the Fnodes. Currently, PUMA constructs two possible shortest paths, after applying XY and YX routing in between the two Fnodes. In Figure 5, $Path_0$ represent one such path in between $Fnode_0$ and $Fnode_3$. PUMA then evaluates the affective throughput (in terms of time-slots) across each path, i.e., which can ensure the availability of time-slots in a pipelined fashion (equation 7). The affective throughput is then used to determine the allocation of a cluster in Section IV-D.

As an **example**, Figure 5 illustrates the process for obtaining the affective time-slots for $Path_0$. These are obtained by constructing a path table by using the approach of [5]. This accounts for obtaining an aggregated slot table (t_{agg}) of all the path-links through shift and merge processes, Figure 5. For instance, merging of two empty slots tables, i.e., (t_{agg}) and t_{NIO-R0} results in an empty slot table, Figure 5 (Step 2). Then, the slot table of the next link is selected, which as shown in Figure 3B has two occupied slots (i.e., at slot 0 and at slot 4). Shift left of t_{R0-R1} by 1 moves slot 4 to slot 3, and slot 0 to slot 4, Figure 5 (Step 3). As a next step (t_{agg}) (Step 2) and t_{NIO-R0} (Step 3) are merged (Step 4) to obtain an updated (t_{agg}). The next path-link t_{R1-R3} is then shift left by 2, and so on. As its a shift and merge operation. Therefore in the aggregated slot-table an empty slot on a particular position is found, if and only if, it is empty across all the path links. After creating the database, the clusters of an input application are created, as explained below.

B. Application Traversal Order

As applications are becoming increasingly complex [16]. Therefore, picking an application as a whole and try every possible binding possibility becomes highly time consuming. Alternatively, there are different schemes [14] that can allow cluster- or level-based traversal of an application task graph. Our PUMA scheme performs an application binding on cluster-wise basis. The clusters are created by exploiting inter-IP communication dependencies, because inputs of an IP can be dependent on the output of some other IP(s). For explaining the cluster creation process, we use an application in Figure 6A that exhibits all types of inter-IP communication, i.e., single to single, single to many, many to single, and many to many. For instance in Figure 6A, ip_a communication with (ip_b , ip_c ,

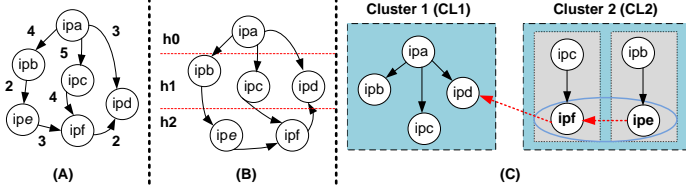


Figure 6. Showing: (A) An Application Task Graph With Communication Requirements In Terms Of Time-Slots, (B) Hierarchies, (C) Clusters

ip_d) stands for single to many case. The numbers in Figure 6A, indicate inter-IP communication demands in terms of time-slots. Starting with the first IP (ip_a), we start creating clusters based on the following rules:

Rule 1: We first identify the communication levels (also termed as hierarchies) among an application IPs. For this purpose, we place (ip_a) in the zeroth hierarchy (h_0), Figure 6B. As the inputs of ip_b, ip_c , and ip_d are driven by ip_a outputs. Therefore, these are placed next to ip_a hierarchy, i.e., h_1 . In this situation, we can term ip_b, ip_c , and ip_d as the destinations for the source ip_a . Similarly, IPs in h_2 are the ones that serve as destinations for IPs in h_1 .

Rule 2: A cluster indicates communication among the IPs that belong to different hierarchies. Moreover, a cluster can comprise of multiple sub-clusters. A sub-cluster represents communication between a *source IP* and single/multiple destination IP(s). For **example**, Figure 6C shows a cluster (CL_2) that indicates communication between h_1 and h_2 IPs. Additionally, in CL_2 the ip_c - ip_f relation stands for one of its two sub-clusters. **Rule 3:** An IP inputs can be driven by multiple IPs, which exist in different hierarchies. In this case, the source IP in the lower hierarchy determines the hierarchy of the destination IP. For **example** in Figure 6B, ip_d is accessed by ip_a and ip_f that belong to h_0 and h_2 , respectively. However, ip_d comes next to ip_a hierarchy.

After the clusters are created, their binding is performed one by one, i.e., CL_2 binding follows CL_1 . However, three types of dependencies (based on inter-IP communication) can exist at inter- and intra- cluster levels. a) Two IPs communicate and both in the current input cluster, but belong to different sub-clusters of it. For *instance*, ip_e - ip_f connection in CL_2 , Figure 6C,. This inter-dependency is detailed in Section IV-D. b) Two IPs communicate and one of them belong to an already bound cluster. For *instance*, ip_f placement would be fixed after it ensures allocation for ip_f - ip_d connection. This inter-dependency is detailed in Section IV-D. c) Two IPs communicate and one of them belongs to a cluster that is yet to be bound. For *instance*, at the time of CL_1 binding, the resultant ip_d placement should not block the allocation of a future ip_f - ip_d connection. This inter-dependency is detailed in Section IV-E.

We assume that the nature of our application is streaming, i.e., it is throughput sensitive and does not have strict latency constraints. Additionally, we assume a connected task graph for an input application.

Algorithm 1 Solution Space Creation

Require: $minlatency, maxslots, maxarea, SrcFnode$

Ensure: $SolSpace$ for $SrcFnode$

1. $NetFnodes = getNetFnodes(SrcFnode)$
2. **for** ($i = 0; i < NetFnodes; ++ i$) **do**
3. $FArea = getArea(NetFnodes_i)$
4. $PSlots = getPathSlots(SrcFnode, NetFnodes_i)$
5. $Hops = getHops(SrcFnode, NetFnodes_i)$
6. $Hdelay = Hops * 0.006$
7. **if** ($Hdelay < minlatency$) & ($PSlots > maxslots$) & ($Farea > maxarea$) **then**
8. $SolSpace.push(NetFnodes_i)$
9. **end if**
10. **end for**

C. Solution Space Extraction

The solution space (Sspace) for a cluster is extracted before performing its binding. A *cluster solution space* is defined as the set of FPGA nodes that can fulfill its combined logic and communication demands. A cluster can be comprised of multiple sub-clusters. In this situation, the Sspace of each sub-cluster is extracted, individually. As our input applications have connected task graph. Therefore, each cluster (except the first one), at the time of its binding, would have its source IP(s) placed due to its presence in an earlier bound cluster. Currently, we assume that the source IP of first cluster can be placed in the first available Fnode. This can be $Fnode_0$, or the Fnode where the previously selected application was successfully bound.

To extract the solution space for the selected (sub)cluster, the source Fnode (where the source IP is placed) is obtained. Then, the (sub)cluster task graph is traversed to find out: a) its IPs with minimum area requirement, b) connections with maximum throughput, c) and connections with minimum latency requirements. For each of the network Fnodes, the available logic area is obtained. Then, available path-slots are also obtained in between the source Fnode and the selected network Fnode. Initially, remaining network Fnodes are obtained, algorithm 1 (line 1). The available logic area is obtained for a selected network Fnode. Hop delay and path-slots are also obtained in between the source Fnode and the selected network Fnode, algorithm 1 (line 3-6). In algorithm 1, 0.006 indicates a per hop delay in μs . This information is then, evaluated against the minimum permitted latency, maximum required throughput, and maximum area demands, algorithm 1 (line 7). In case of success, the selected network Fnode is put in the solution space of the source Fnode.

D. Candidate Solution Finding

The extracted solution space is now used to obtain *candidate binding solutions* for the input cluster, Figure 8. In case a cluster is comprised of multiple sub-clusters. Then, the process is performed for all the sub-clusters, simultaneously. The reason is that a cluster IP can have connections in multiple sub-clusters. Thus raising inter sub-cluster dependencies. For *instance*, in

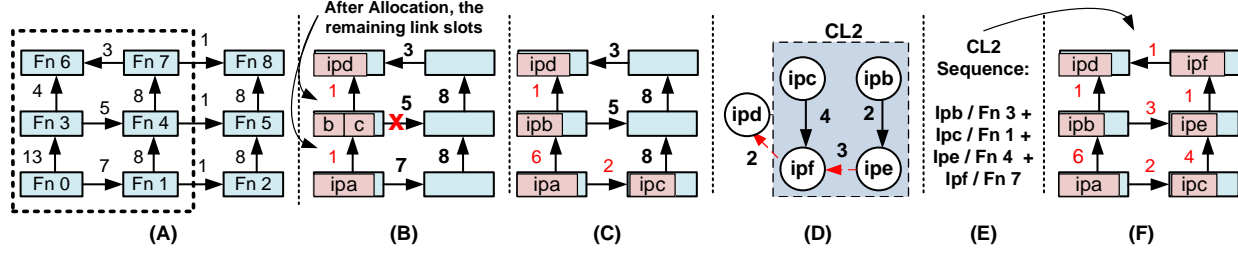


Figure 7. Showing: (A) FPGA architecture with available time-slots. Rectangle indicates CL1 solution space, (B) Failed binding for CL1 IPs, (C) Successful binding for CL1 IPs, (D) CL2 IPs with time-slot requirements, (E) A sequence for CL2, (F) Binding of CL2 IPs in accordance with the sequence.

Figure 6C, ip_f communicates with ip_c and ip_e that exist in different sub-clusters of (CL_2).

PUMA generates a number of sequences, to find out the solution candidates of a cluster, Figure 8. A sequence consists of all the cluster IPs that are paired with some/all Fnodes of the solution space. More precisely, a term in a generated sequence represents an IP/Fnode pair, and a single Fnode can be paired with multiple IPs. Figure 7E shows one such generated sequence.

After a sequence is created, PUMA examines its validity/failure by using a 3-tier pruning process, Figure 8. It is performed for all the IP/Fnode pairs in a sequence, and by comparing an Fnode available resources against the required resources of its paired IP(s). The 3-tier pruning process is successful, when all of its IP/Fnode pairs pass the placement, mapping, and allocation pruning process. On finding a valid sequence, the candidate solution database is updated. However, in case of an invalid solution, (a possible) failed IP/Fnode pair is sent back. This helps in branching out those sequences, which could be invalid due to the same IP/Fnode pair. Once, the validity/failure of a sequence is concluded. The resources that were marked reserved during the sequence evaluation, are restored back. This way the remaining sequences are examined, to create a set of perspective candidate solutions. We next explain the 3-tier pruning process that decides the validity/failure of a sequence.

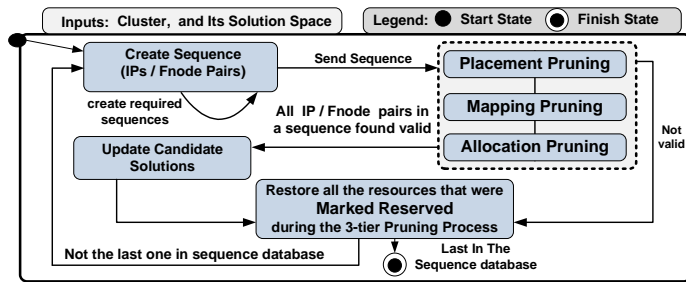


Figure 8. Finding The Candidates Solutions

1) *Placement Pruning*: Placement pruning for an IP/Fnode pair ensures that the respective CFR meets the area constraints of the IP that is going to be placed on it. A successful placement pruning requires that equation 2 is satisfied for all the IP/Fnode pairs of the input sequence. For **example**, in Figure 7E a generated sequence pairs ip_f to $Fnode_7$. In this case $Fnode_7$ passes placement pruning for ip_f , only and only if, the available area of $Fnode_7$ CFR is less than the required area of ip_f .

Similarly, the test is conducted for the remaining IP/Fnode pairs of the sequence in Figure 7E. Currently, we don't take into account the xy dimensions of an IP. After a successful placement pruning, sequence mapping is evaluated.

2) *Mapping Pruning*: A successful IP to NI mapping ensures that the respective NI has enough ports for connecting the IP ports. In addition, the NI should have enough bandwidth at its ingress and egress links for the incoming and outgoing connection of the mapped IP, respectively. In other words, a successful mapping pruning requires that equations 3, 4, and 5 are satisfied for all the IP/Fnode pairs. For **example**, for the sequence in Figure 7E, the $ip_f / Fnode_7$ passes mapping pruning, if and only if, $Fnode_7$ associated NI has: i) ports available to connect ip_f ports (in this case 3 ports), ii) more time-slots available on the NI *ingress* link than are required by the connections that are incident to ip_f (from Figure 7D, this accounts for ip_c-ip_f and ip_e-ip_f), iii) more time-slots available on the NI *egress* link than are required by the connection that are emerging from ip_f (from Figure 7D, this accounts for ip_f-ip_d).

Similarly, the test is conducted for the remaining IP/Fnode pairs of sequence in Figure 7E. PUMA, while evaluating the mapping of a sequence, takes into account the amount of time-slots but not their positions. The actual positioning of time-slots across the network paths is evaluated during the allocation pruning, as explained below.

3) *Allocation Pruning*: During an allocation pruning it is ensured that the paths between the sequence Fnodes have sufficient time slots to allocate connections that exist among the sequence IPs. A (cluster) sequence passes allocation pruning, only and only if equations 6 and 7 are satisfied for the input connections. Additionally, when an IP (in the current cluster) has dependencies with IP(s) of previously bound clusters. Then, the dependencies are accounted during an allocation phase.

We explain allocation pruning by using our **example** application in Figure 6. We assume that at the time of CL_2 allocation pruning, CL_1 IPs ip_a , ip_b , ip_c and ip_d are bound to $Fnode_0$, $Fnode_3$, $Fnode_1$, and $Fnode_6$, respectively (Figure 7C). Now, if the generated sequence in Figure 7E, binds CL_2 destination IPs, i.e., ip_e and ip_f to $Fnode_4$ and $Fnode_7$, respectively. Then, the allocation pruning ensures that this sequence is valid, only and only if sufficient resources (time-slots) available across: i) $Fnode_3-Fnode_4$ path for ip_b-ip_e connection, ii) $Fnode_1-Fnode_7$ path for ip_c-ip_f connection, iii) $Fnode_4-Fnode_7$ path for ip_e-ip_f connection, iv) and $Fnode_7-Fnode_6$ path for ip_f-ip_d connection.

Our PUMA scheme after a successful 3-tier pruning, marks

the current sequence as a possible candidate solution. Then, depending upon the sequence generation logic, either apply 3-tier pruning on the next generated sequence or start the construction of the best possible solution as explained below.

E. Solution Construction

After obtaining candidates solutions (SI), where each $S_i \in (SI)$ binds cluster with guarantees on placement, mapping and allocation. PUMA starts constructing the best possible solution, for which it takes into account two important factors.

1) *Inter-Cluster Dependencies*: First, in a constructed solution an IP (now placed in a Fnode), which has dependencies with IPs in yet to bind cluster, should not be blocked at its Fnode router.

For **example**, Figure 6C shows that CL_1 IPs ip_b and ip_c have dependencies with yet to bind CL_2 cluster. Now, if ip_b and ip_c are paired to the same $Fnode_3$, Figure 7B. Then, the throughput requirements of ip_a-ip_b (4 time-slots) and ip_a-ip_c (5 time-slots) connections are fulfilled. Meanwhile, the produced logic fragmentation is at minimum. So, from the current cluster perspective, $ip_b / Fnode_3$ and $ip_c / Fnode_3$ pairs should be valid.

However, in this case, ip_b or ip_c gets blocked at $Fnode_3$ router, Figure 7B. Because $Fnode_3$ router has lesser number of time-slots (5) than are required by ip_b and ip_c , i.e., 6 time-slots. This in turn can block either ip_b or ip_c from sending data into the network. Alternatively, Figure 7C shows one such solution. Here, $ip_b / Fnode_3$ and $ip_c / Fnode_1$ pairs not only fulfill CL1 allocation, but also don't block ip_b and ip_c from sending data into the network.

2) *The Optimisation Criteria*: Next, an optimisation criteria is used that decides the overall cost of the constructed solution. It consists of three factors as described below:

$$cost(S_i) = \lambda * LogicI(S_i) + \beta * CommI(S_i) + \theta * ContI(S_i) \quad (8)$$

Here $LogicI(S_i)$ indicates the fragmentation produced by (S_i) in the solution Fnodes. It is designed such that the best solution opts for an S_i with minimum fragmentation that is produced in least number of Fnodes. This not only reduces the number of fragmented Fnodes, but also increases the (untouched) Fnodes for the remaining clusters/apps.

The second part of *equation 8* accounts for the network over-allocation, which is caused by an S_i . It is designed such that the best solution opt for an an S_i with minimum over-allocation. Now, if there are n connections in S_i . Then, the original communication volume required by it is defined as: $\sum \alpha(c_i), \forall i = \{0, \dots, n\}$. With each $c \in C$ going through a network path $\pi(c)$ for data transfer, the actual volume over the network then becomes: $\sum \pi(c_i) * \alpha(c_i), \forall i = \{0, \dots, n\}$. Hence the over-allocation of S_i over the network can be calculated by using the following equation.

$$\sum_{i=0}^{i=n} \pi(c_i) * \alpha(c_i) \div \sum_{i=0}^{i=n} \alpha(c_i) \quad (9)$$

The third part of *equation 8* accounts for the mean of contention over the network links. It is calculated by averaging the residual slots across the links of selected paths.

In equation 8 λ, β and θ are constant that vary from 0 to 1.0, and indicate the importance of the decision w.r.t. logic and/or communication resource optimization. Its because the $S_i \in SI$ can belong to an application, which falls in any of the four categories of Figure 1. Therefore, the optimisation criteria is not the same every time. For example, an application with low area and high communication demands can require a solution, which comprises shorter paths and uniform contention. On the contrary, an application with high area and low communication demands would prefer a solution, which causes lower logic fragmentation. After constructing the best solution, resources are reserved for it, as explained below.

Algorithm 2 Resource Reservation Process

Require: $Solution, CurClstrCon, PrvClstCon$

Ensure: $\forall ClusterIPs : place, map, allocate$

1. $IPandFnode = getIPandFnodes(Solution)$
2. **for** ($i = 0; i < IPandFnode; ++ i$) **do**
3. $ip_i = getIP(IPandFnode, i);$
4. $cfr_i = getCFR(IPandFnode, i);$
5. $cfrNI_i = getCFRNI(cfr_i);$
6. $link_E = getEgLink(cfrNI_i);$
7. $link_I = getInLink(cfrNI_i);$
8. $Con_{ip_i} = getCons(ip_i, CurClstrCon, PrvClstCon);$
9. $Cons2Allocate.insert(Con_{ip_i})$
10. $place : ip_i \rightarrow cfr_i;$
11. **for all** $ports_i \in ip_i$ **do**
12. $mapPorts : ip_i \rightarrow cfrNI_i;$
13. **end for**
14. **for all** $conn_{ip_i}$ **do**
15. $mapInConns : Iconn_{ip_i} \rightarrow link_I;$
16. $mapEgConns : Econn_{ip_i} \rightarrow link_E;$
17. **end for**
18. **end for**
19. **for** ($j = 0; j < Cons2Allocate; ++ j$) **do**
20. $sCFR_j = getCFR(getsIP(Cons2Allocate_j));$
21. $dCFR_j = getCFR(getdIP(Cons2Allocate_j));$
22. $FnodePath_j = getFnodePath(sCFR, dCFR_j);$
23. $allocate : ClusterCon_j \rightarrow FnodePath_j;$
24. **end for**

F. Cluster Resource Reservation

The resource reservation is performed for the best solution after extracting all of its IP/Fnode pairs, algorithm 2(line 1). Then, necessary information for each IP/Fnode pair is obtained that include: a) the IP and its ports, b) CFR and NI of the Fnode, c) ingress and egress NI links, d) and IP connections that provide communication with other IPs in the current cluster and in the already bound clusters, algorithm 2(line 3 – 8). After placing an IP in the CFR, the IP is mapped to the respective Fnode NI. This is achieved by connecting IP ports to NI ports. Then reserving time-slots for the IP in and out connections over the NI links, algorithm 2(line 10 – 17).

Allocation comes after placing and mapping the cluster IPs. The connections account for cluster IP(s) communication in the

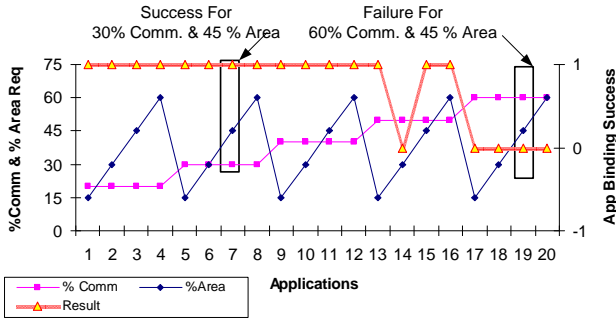


Figure 9. PUMA Success With Variable Communication And Area Demands

existing and previously bound clusters. This is executed by first extracting the required paths in between the Fnodes. Then reserving the resources across the paths in accordance with equation 6 and 7, algorithm 2(line 19 – 24). Meanwhile, the database is updated to reflect new residual Fnode resources and new slot tables for their paths.

V. RESULTS AND ANALYSIS

We implemented PUMA scheme in SystemC using the design flow of [3]. We evaluated performance, cost and scalability of our PUMA scheme, as described in following sections.

A. Performance: Success-Rate

PUMA success rate, i.e., *application binding with QoS guarantees*, was evaluated by generating a number of synthetic application using SDF3 [17]. In following Figures, we indicate successful binding with 1, and failed binding event with 0.

Figure 9, illustrates PUMA binding results for 20 apps., each of which comprised 15 IP cores. However, possessed different area / communication requirements against the the same FPGA architecture. Meanwhile, there is 0% standard deviation (SD) among the throughput demands of apps. connections, i.e., all the connections are at the same throughput value. In Figure 9, rectangles are used to highlight two instances of app. binding.

Next for the same set of 20 apps. of Figure 9, we introduced randomness in their throughput demands by changing SD values, Figure 10A. For instance, Figure 10B shows that an app. requiring 30% of FPGA communication resources, has a mean bandwidth of 250 MB/s for its connections. And, a 30% SD for this app. induces a difference of 100 MB/s from the mean of 250 MB/s. This means the connection demands can exist in

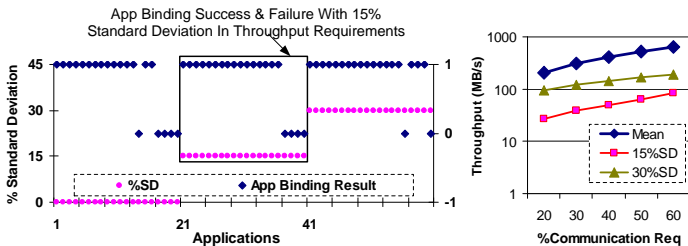


Figure 10. (A) Binding Results with varying % SD, (B) Relation between % SD and % Communication

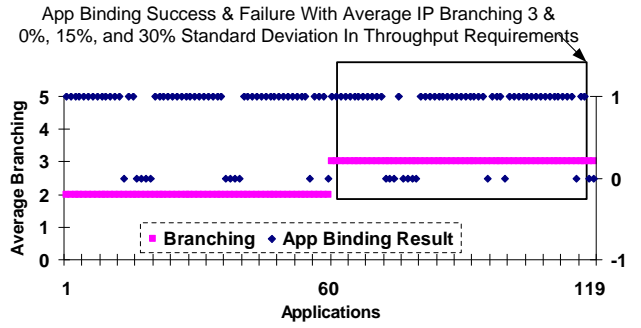


Figure 11. Impact on App. Binding Success by Varying inter-IP dependencies

between 350 MB/s and 150 MB/s. This increased randomness makes it difficult to find time-slots as per required positions.

As the throughput requirements reach 60% of FPGA resources, the failed binding instances are more with 0% SD than are with 15% SD or 30% SD. Because with 0% SD and 60% communication, all the app. connections have the same mean of 900 MB/s. Ideally, our HWNoC links support a max. of 2000 MB/s. Therefore, with even two connections that share the same link, the probability to induce contention becomes higher.

Next for the same set of 60 applications of Figure 10, we changed IP *branching*, i.e., average number of connections/IP. This means an IP can now communicate with more IP. Therefore, interdependencies among the application IPs increases. Figure 11 shows application success rate with average branching of 2 and 3. Interestingly, the results for each branch hold a similar pattern, mainly due to the assurance of enough resources over the Fnodes NIs at the time of IP mapping. This enables the future connections for that IP to get allocated.

Next we averaged the success rate for a particular area / communication combinations that exist in 120 applications of Figure 11. In Figure 12 a value, e.g., A30-C30 shows that an application requires 30% resources from both of the FPGA planes to fulfill its QoS demands. Figure 12 reflects high success rate with high area and low communication, and low area and low communication demands. However, as the communication req. increase, success rate decreases. In Figure 12A, this is mainly due to the highly random logic and communication demands. On the other hand in Figure 12B, the decreased success rate is due to the area saving objective that concentrates the resources in smaller number of Fnodes. The objective, here is to reduce the cost paid over QoS guarantees, i.e., to avoid producing high fragmentation over the logic plane, which can prohibit the binding of future applications.

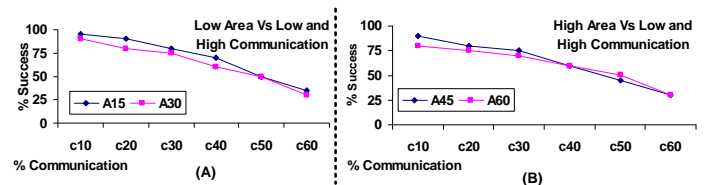


Figure 12. PUMA Success Rate For Different Area to Communication Ratios

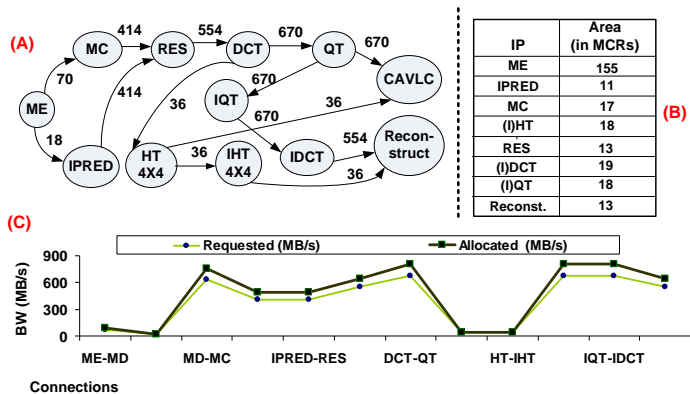


Figure 13. Showing H264: (A) Task Graph With Inter-IP Communication in MB/s, (B) And Area Demands, (C) Required Vs Allocated Bandwidth

B. Performance: Cost Over QoS Guarantees

We picked up a real-world H.264 encoder application, Figure 13(A), to evaluate PUMA cost that is paid over the QoS guarantees. The reason is that ensuring QoS guarantees is important, but it should not be at the cost of too-high resource reservation. In Figure 13B, the area numbers of H.264 encoder (excluding CAVLC) are obtained from synthesis, and from [16]. We synthesized RES, (I)DCT, (I)QT, and Reconstruction blocks, whereas the remaining ones were obtained from [16]. The throughputs were calculated for high definition format which requires 3600 Macro Block processing for each video frame.

Figure 13C illustrates that PUMA fulfills QoS demands of each H.264 connection. Additionally, the allocated throughput is kept around 1.2 times of the required ones. This accounts for a reasonable 20% extra of the required one, which is mainly due to the communication-aware binding.

C. PUMA Scalability

To find out the the scalability of our PUMA scheme, we use a varied number of application / architecture combinations as shown in Table II. For instance, the first row of Table II shows an FPGA with 9 CFRs and hops that are arranged 3×3 dimensions. During the binding process we: a) pick up an application task graph and an FPGA architecture, b) change the application area demands from 15% to 60% of the FPGA architecture, c) for each area percentage change the throughput demands of the application from 10% to 60% of the available ones, d) record the result for each combination, e) and average out the results to find out the success-rate of the application binding over the architecture. As could be seen in Table II, PUMA scheme holds a success-rate in between a good 70% and a reasonable 50%.

VI. CONCLUSION

In this paper, we presented a scheme for application to FPGA binding. The PUMA scheme unifies placement, mapping, and allocation. Moreover, the PUMA scheme ensures QoS guarantees whenever an application binding is successful. We presented the mechanism to perform the unification, evaluated the success-rate and scalability over multiple synthetic applications. PUMA success-rate was found to be in between a good

Table II
SUCCESS RATE OVER MULTIPLE APPLICATION AND FPGA DIMENSIONS

FPGA Dimensions	App IPs	App Connections	%Success-Rate
3x3	5	6	68
	7	8	60
	9	14	56
4x3	11	18	65
	13	21	53

70% and a reasonable 50%. We also examined the scalability of our PUMA scheme by binding multiple applications over different architectures.

REFERENCES

- [1] T. Ahonen, et al., "Topology optimization for application-specific networkson-chip," in *SLIP*, 2004.
- [2] C. Bobda, et al., "A dynamic NOC approach for communication in reconfigurable devices," in *FPL*, 2004.
- [3] K. Goossens, et al., "A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification," in *DATE*, Mar. 2005.
- [4] K. Goossens, M. Bennebroek, J.Y. Hur and M.A. Wahlah, "Hard-wired networks on chip in FPGAs to unify data and configuration interconnects," in *NoCS*. Apr. 2008.
- [5] A. Hansson, et al., "A unified approach to mapping and routing on a network on chip for both best-effort and guaranteed service traffic," *VLSI Design*, May. 2007.
- [6] R. Hecht, et al., "Dynamic Reconfiguration with hardwired Networks-on-Chip on future FPGAs," *FPL*, 2005.
- [7] J. Hu, et al., "Energy-aware mapping for tile based NoC architectures under performance constraints," *ASPDAC*, 2003.
- [8] W. Jang, et al., "A3MAP: Architecture-Aware Analytic Mapping for Networks-on-Chip," *ASPDAC*, 2010.
- [9] A. Kumar, et al., "An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip," *DATE*, April. 2007.
- [10] S. Lukovic, et al., "An Automated Design Flow for NoC-based MPSoCs on FPGA," *RSP*, June. 2008.
- [11] T. Marescaux, et al., "Networks on chip as hardware components of an OS for reconfigurable systems," *FPL*, Sep. 2003.
- [12] S. Murali, et al., "Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees," *ASPDAC*, 2005.
- [13] H. Nguyen, et al., "An QoS Aware Mapping of Cores Onto NoC Architectures," *ISPA*, 2007.
- [14] K. Purna, et al., "Temporal Partitioning and Scheduling Data Flow Graphs for Reconfigurable Computers," *IEEE Transactions On Computers*, Jun. 1999.
- [15] P. Sedcole, et al., "Modular dynamic reconfiguration in Virtex FPGAs," in *Proceedings Computers and Digital Techniques*, May. 2006.
- [16] M. Shafique, et al., "Optimizing the H.264/AVC Video Encoder Application Structure for Reconfigurable and Application-Specific Platforms," in *JSPS*, Vol 60, 2010.
- [17] S. Stuijk, et al., "Multiprocessor Resource Allocation for Throughput-Constrained Synchronous Dataflow Graphs," in *DAC*, 2007.
- [18] M.A. Wahlah, et al., "Modeling reconfiguration in a FPGA with a hardwired network on chip," in *RAW*, May. 2009.
- [19] Xilinx Inc., "Virtex-6 Data Sheets," 2009.
- [20] Xilinx Inc., "Virtex-4 Configuration Guide."