New Kids on the Unblocking: Strategies to Overcome Blocking Networks

Caio Von Rondow Morais[1], Jeronimo Penha[1], José A. Nacif, Ricardo Ferreira[1]

[1]email: ricardo@ufv.br, Universidade Federal de Viçosa, Brazil

Abstract. *Full-crossbar interconnections offer high parallel bandwidth, simple routing, and performance. However, their cost is prohibitive. Multistage networks have emerged as a cost-effective alternative, reducing the cost to* $O(n\log(n))$*. This study addresses the challenges posed by large networks with 256 connections, where the configuration space expands exponentially. We present an approach that efficiently handles routing by reducing the extra stages. We reduce network cost by a factor of* $2\times$*. Our approach enables graph routing with up to 214 operators within the 256-connection multistage network, doubling performance compared to previous multistage CGRA frameworks.*

1. Introduction

Interconnections play a fundamental role in parallel computation. High-performance Alveo FPGAs have recently incorporated a hardwired built-in partial crossbar network to provide high throughput for distributing data from its 32 high bandwidth memory (HBM) channels. Despite the peak memory rate being 460 GB/s, even with dedicated interconnection hardware, achieving peak performance in real scenarios remains challenging. For instance, sort algorithms implemented using the partial crossbar achieve a maximum throughput of 10 GB/s [Choi et al. 2021] since it requires distributing multiple HBM and performing all-to-all communication. However, by incorporating a customized reconfigurable interconnection using dynamic 2x2 switches and ignoring the built-in crossbar, it is possible to improve the memory throughput up to 200 GB/s [Choi et al. 2021]. The customized network relies on the butterfly topology, a classical multistage network.

The hardware cost of a multistage network is $O(n\log n)$, which is more efficient than the $O(n^2)$ cost of full crossbar networks. As a result, multistage networks can scale to accommodate hundreds of channels, while crossbars are typically limited to 32 channels in most technology implementations. However, it is important to note that full crossbars do not have routing conflicts, while multistage networks can block.

When considering a network with n inputs/outputs, a network with $\log(n)$ stages will block since the number of possible permutations is smaller than the total number of $n!$ permutations. As the Benes network proves, one solution to this issue is to double the number of stages, using at least $2\cdot\log(n)-1$ stages. Benes is a rearrangeable network that allows all permutations by appropriately assigning connections to the network switches. Nevertheless, when dealing with multicast patterns, the complexity escalates considerably, and doubling the number of stages may not be sufficient to handle it. In the domain of parallel computing, the majority of graphs are non-tree structures, consequently exhibiting multicast patterns.

This study proposes strategies to address the blocking issues associated with multistage interconnections (MIN) in multicast scenarios. The main focus is accelerators by using coarse grained reconfigurable architectures (CGRA). A previous

work [Silva et al. 2019] shows that MINs provides flexible CGRAs that could be reconfigured at run-time. CGRAs simplify placement by offering reconfigurability at the word level, rather than the bit level seen in FPGAs. However, it relied on a greedy placement and routing approaches. We propose to investigate the bounds and the trade-offs of blocking and non-blocking as a function of the number of stages and placement algorithms.

This work is organized as follows. Section 2 describes the CGRA mapping with a multistage. Section 3 presents the challenges of performing a high number of multicast routing. We outline our mapping approaches in Section 4.4. Section 5 presents the main results. Finally, Sections 6 and 7 discuss the related work and draw the conclusions.

2. Problem

We evaluate the performance of multistage placement and routing strategies for parallel architectures with n Processing Elements (PEs). While mesh-based architectures help reduce interconnection costs to $O(c)$, the mapping process is NP-complete. Exact solutions using integer linear programming or SAT solvers do not scale well for graphs with more than 30 nodes [Walker 2019]. Despite the use of recent state-of-the-art mapping techniques that employ reinforcement learning and graph attention networks [Kong et al. 2023], the mapping process still requires approximately 10 seconds and is limited to graphs with fewer than 68 nodes. Furthermore, these architectures have yet to be validated on real devices. As a baseline, we choose a reconfigurable architecture [Silva et al. 2019] consisting of 128 PEs that has been successfully validated on an FPGA using multistage networks.

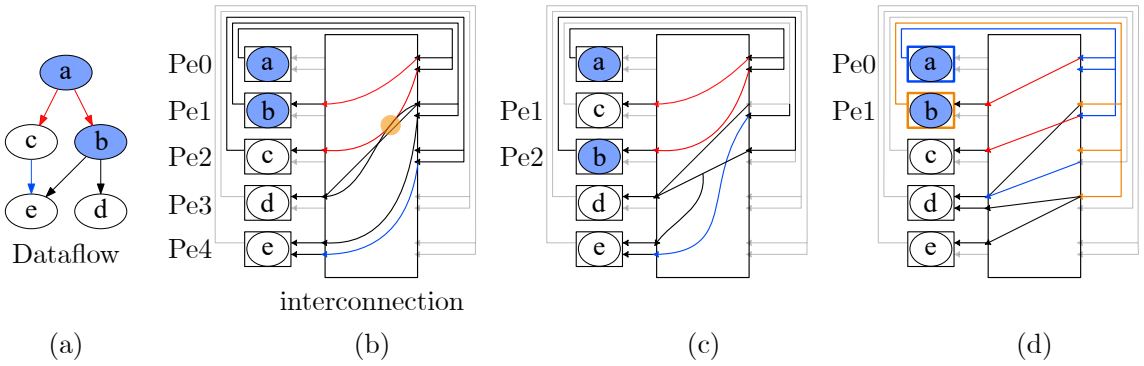

Figure 1. (a) Dataflow; (b) Routing Conflict; (c) Swapping; (d) Multicast.

The input for the mapping algorithm is an application dataflow graph (DFG) depicted in Figure 1(a). The goal of the mapping algorithm is to assign the DFG nodes to physical PEs and route the edges using the interconnection network. A routing conflict invalidates a placement solution as shown in Figure 1(b) for the edges $a \rightarrow c$ and $b \rightarrow d$. There are several strategies to overcome a routing conflict, such as leveraging symmetries present in the target architectures, as illustrated in Figure 1(c), where the nodes b and c are replaced by swapping them. Another strategy is considering the design features of the PEs and interconnection network, as depicted in Figure 1(d), where the PE_0 and PE_1 (highlighted in blue and orange, respectively) have three outputs as they require multicast patterns. Key design features include the number of interconnection stages, the types and quantity of PEs, and input/output permutation assignments.

3. Multistage Network: Why it is Blocking

We will present a simple 4x4 Multistage Interconnection Network (MIN) to illustrate the blocking properties. A MIN with n inputs and n outputs consists of a set of s stages, each with 2x2 programmable switches. The interconnection pattern between the stages determines the network type. In Figure 2(a), we depict a 4x4 MIN with two stages using the shuffle-exchange pattern, where the input/output permutation pattern routes successfully.

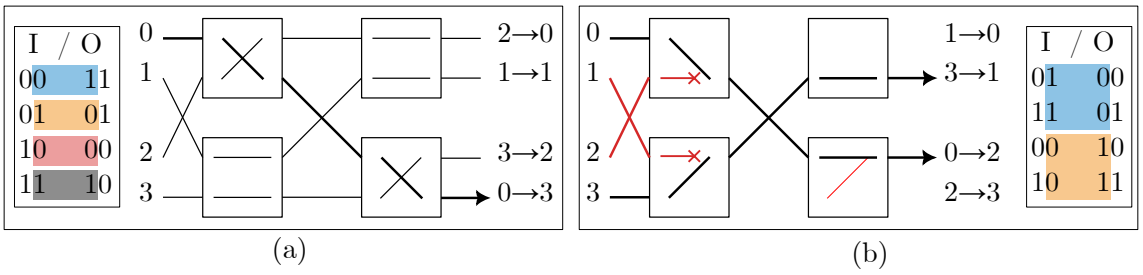

Figure 2. (a) No conflict $0 \to 3, 1 \to 1, 2 \to 0, 3 \to 2$; (b) Conflict permutation.

When $n = 4$, there are $n! = 24$ possible permutations. However, this MIN has only 4 switches, with two states each (direct and crossover). Consequently, it can accommodate only $2^{\text{number of switches}} = 2^4 = 16$ distinct permutations. As a result, it is not possible to route at least 8 patterns. Figure 2(a) and (b) provide examples of one possible permutation and one blocking permutation pattern, respectively. The connections $0 \to 2$ and $2 \to 3$ have a conflict in the first stage, as well as the connections $1 \to 0$ and $3 \to 1$ (see Figure 2(b)).

In the presence of a multicast connection, the routing problem becomes more complex. Each switch in the network exhibits four different states: direct, crossover, diffusion up-to-all, and down-to-all, as illustrated in Figure 3(a). In a 4x4 network, where each multicast switch has four states, there can be up to $4^4 = 256$ possible configurations to cover 256 multicast patterns.

However, specific patterns, such as the simple pattern $0, 2, 0, 2$, cannot be routed as depicted in Figure 3(b). This situation happens because inputs 0 and 2 should broadcast to both outputs. A third stage can be added to the network architecture to overcome this limitation, as demonstrated in Figure 3(c). This additional stage increases the number of possible patterns to $4^6 = 4096$, which is sixteen times larger than the maximum of 256 patterns in the initial configuration. Consequently, the routing problem exhibits a significantly larger solution space.

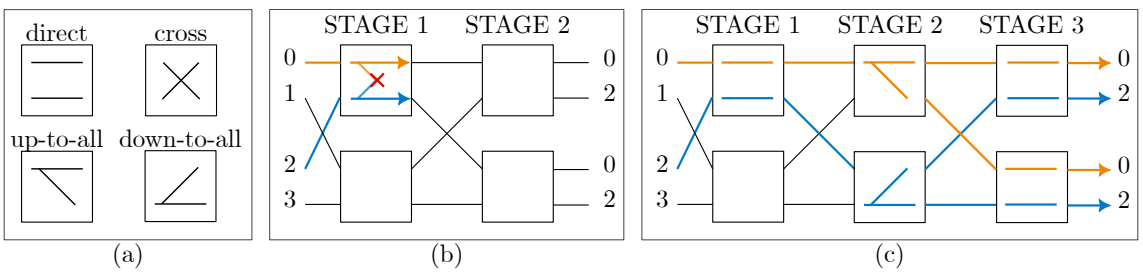

Figure 3. (a) 4 States; (b) Multicast conflict; (c) Extra Level Solution.

However, when we consider a more extensive network, such as an 8x8 configuration, the complexity of the routing problem escalates. In an 8x8 network with the minimum of $\log(n) = 3$ stages, there are $4^{12} = 16$ million possible configurations to cover 16 million possible patterns. However, the network is blocked since it is impossible to route all patterns.

Additional stages are required to address this limitation. After including one extra stage, the network would have 4 billion possible configurations for the 16 million patterns to verify. This number grows exponentially with $4^{n/2}$ for each additional level. Considering that an 8x8 network necessitates at least two extra levels, the routing complexity becomes an immense challenge in the presence of multicast. Although extra stages increase the latency, a multi-thread approach could be used to hide it [Silva et al. 2019].

4. Mapping Strategies

4.1. How blocking is a Network

Table 1 illustrates the routing capability of 4x4 and 8x8 networks with a minimal size and an additional level. A minimal 4x4 network experiences blocking for 112 out of 256 patterns, representing 43% of all patterns. Conversely, a minimal 8x8 network encounters blocking for nearly 97% of the patterns. By introducing an extra level, a 4x4 network becomes unblocking. However, finding a suitable configuration within a space of 4K configurations poses challenges. Specifically, 48 patterns have only 2 configurations (Cfgs) out of the 4K possibilities, making them difficult to identify compared to one-to-all patterns like 0 in all outputs, with 176 configurations within the 4K space.

Table 1. Minimal and one extra level blocking patterns in 4x4 and 8x8.

		Minimal $\log(n)$		One extra level			
net	number of patterns	block pattern	percentage of blocking	Cfg	blocking patterns	2 Cfgs	one-to all
4x4	256	112	43%	4K	0 (0%)	48	4 (176)
8x8	16M	16M	97%	4G	10M (60%)	256K	8 (3M)

For an 8x8 network, the routing problem intensifies. Despite adding an extra level, 60% of the patterns may remain unroutable. Moreover, 256K patterns possess only 2 valid configurations within a vast space of 4G possibilities. In contrast, broadcast patterns have 3 million valid configurations, rendering them relatively easy to find. Adding two extra levels, where all one-to-one patterns have a valid solution, the configuration space grows to 1 Tera possibilities, which becomes hard to explore.

We focus on 256x256 networks, where the configuration space is so vast that exploring even a small subset becomes unfeasible. We propose a set of strategies to handle large multistage networks. Our experiment considers multistage networks built using 4x4 local switches, radix 4 instead of radix 2, thus reducing the number of minimal stages to $\log_4(256) = 4$. Each switch has 8 configuration bits and implements 256 possible 4x4 patterns.

4.2. Random Patterns

We propose analyzing the routing capability of a 256x256 network using a random pattern. The total number of multicast patterns in this network is $256^{256} = 10^{671}$. We gener-

ate random patterns with a few multicasts to evaluate the routing capability. To measure routing success, we focus on pairwise connections. For example, if we can route 192 out of 256 pairwise connections, the routing rate is $\frac{192}{256} = 75\%$. We also evaluate partial patterns using only a subset of inputs. For a 50% partial pattern, we randomly select 128 inputs to generate the pairwise connections, including multicast. This analysis provides insights into routing challenges and serves as a baseline for exploring unblocking routing strategies within the network.

4.3. Input and Output Assignments

We generate a routing path in a shuffle-exchange multistage network by concatenating the input and output addresses. Figure 2(a) provides a few examples. For example, the routing word $00\mathbf{11}$ represents the pairwise connection $0 \rightarrow \mathbf{3}$, where the line for the first stage follows the pattern inside the box $0\boxed{01}1$. Routing conflicts only arise if the patterns within the "boxes" are identical. In such a scenario, as illustrated in Figure 2(a), there are no conflicts. However, as shown in Figure 2(b), the pairwise connections $0 \rightarrow 2$ and $2 \rightarrow 3$ encounter a routing conflict as they both require the same "box" code, specifically $0\boxed{01}0$ and $1\boxed{01}1$. One can conclude that conflicts can be amplified when input code suffixes and output code prefixes are identical.

We propose to compare the sequential and random assignment of input and output codes for Processing Elements (PEs). In the sequential approach, each PE, denoted as PE_i, is assigned the codes $2i$ and $2i + 1$ for its two inputs and two outputs. For example, for PE_0, the input/output connections are assigned codes 0 and 1. On the other hand, in the random approach, the input codes for PE_0 can be 45 and 103, while the output codes can be 3 and 191, for instance.

4.4. Placements: Random, Greedy, Local Search, and Simulated Annealing

We evaluate four placement strategies for mapping graph nodes to Processing Elements (PEs). Let's consider nodes a and b, where there is a directed edge from a to b, and we assume a sequential input/output code. The first strategy is just random placement and routing. While a random approach may appear less favorable than an ordered approach, such as sequential assignment, it actually performs better. This is because randomness helps prevent conflicts, given that input suffixes and output prefixes do not exhibit similarities. The second strategy is a greedy approach. This approach places node a in the first available PE, for example, PE_0. Then, it places node b in the next available PE, PE_1. Subsequently, the algorithm checks the routing feasibility from PE_0 to PE_1. If the routing is possible, it adds the routing path to the current mapping. However, if the routing is not possible, the greedy approach assigns node b to the next available PE, such as PE_2. If there is no available and routable PEs to connect to node a (placed in PE_0), the routing adds this edge to the set of unrouted edges and evaluates the next edge. Ultimately, the greedy strategy produces a valid, complete routing or a partial routing with a set of unrouted edges.

The third strategy is the Local Search (LS) approach. It begins by executing the greedy placement strategy. If the greedy approach fails to find a feasible routing for all edges, the Local Search attempts to improve the routing by swapping the positions of two nodes, denoted as a and b, if such a swap reduces the number of unrouted edges. For example, let's assume that the algorithm initially places node a in PE_3 and node b

in PE_8. The LS evaluates whether swapping the positions of a and b (placing b in PE_3 and a in PE_8) reduces the number of unrouted edges. The algorithm terminates when swapping nodes without increasing the number of unrouted edges is impossible.

The last strategy is the meta-heuristic Simulated Annealing (SA). SA prevents getting trapped in local minima within the solution space by temporarily increasing the current routing cost. SA conducts node swapping, even if it results in an increase in the number of unrouted edges. SA starts from a random placement and performs the node swaps to optimize the solution. Additionally, we execute LS as a post-optimization step after the SA process.

4.5. Architecture

The final optimization axis investigated in this study is the heterogeneity of the target architecture. In a homogeneous scenario, all Processing Elements (PEs) in the network have two inputs and two outputs. However, we propose evaluating a non-uniform distribution where certain types of PEs have different configurations. For example, one type of PE may have one input and three outputs, while another type may have two inputs and one output. Introducing this non-uniform distribution adds flexibility. Additionally, if some PEs have only one input, we could accommodate more than 128 PEs for a 256x256 network, expanding the capacity and capabilities of the architecture.

5. Experimental Results

This study proposes evaluating placement and routing strategies as a function of the Connection Workload (CW). In this context, we define CW as the quotient of the number of need-to-be-routed edges and the total size of available interconnections in the network. To illustrate, if we consider a graph containing 128 edges to be placed and routed within a network comprising 256 connections, the corresponding CW would be 50% ($\frac{128}{256} = 50\%$). Since the blocking probability is high, even with a low value of CW could be difficult to route all edges successfully.

5.1. Random Samples

First, we propose to evaluate the routing capabilities of a 256x256 Omega multistage network when handling a set of random patterns. Figure 4 illustrates the success rate of routing edges as a function of the number of extra stages for multicast patterns. We use a box-plot chart that shows the median (middle value) and quartiles (25th and 75th percentiles). When there are no extra stages, there is only one option for each edge, leading to numerous routing conflicts. However, it is possible to reduce the number of conflicts when using extra stages.

In our experiment, we tested one million random patterns with different CW (connection workload) values, namely 50%, 78.125%, and 100%. These correspond to 128, 200, and 256 out of 256 connections, respectively. A routing rate of 95% indicates that we routed 243 out of 256 edges without conflicts. When using 256 connections without extra stages, we could route only 47% of the edges. This rate implies that out of one million random patterns attempting to connect 256 inputs to 256 outputs without additional stages, we should route only 120 edges.

With 128 connections, the best-case scenario resulted in 84 routed edges (or 66%), as depicted in Figure 4. Therefore, finding a configuration with no routing conflicts is

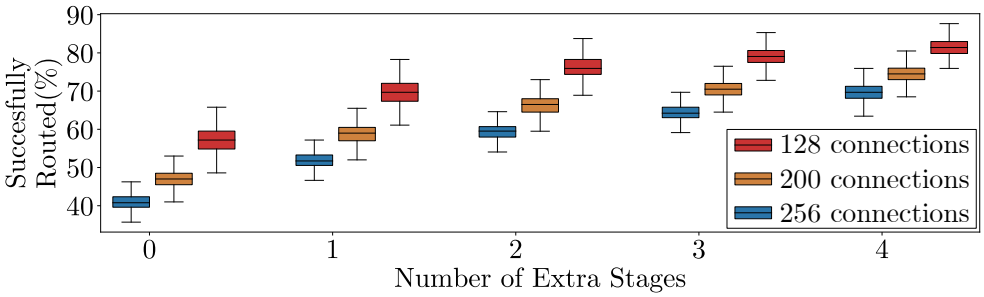

Figure 4. Random Multicast Patterns and Success Rate.

challenging even when the connection workload is low without extra stages. Even with four extra stages and a low CW of 50%, the best-case scenario achieved 114 out of 128 edges successfully routed (89% success rate). Thus, none of the evaluated configurations achieved a conflict-free routing configuration in this experiment.

5.2. Connection Workload and Benchmarks

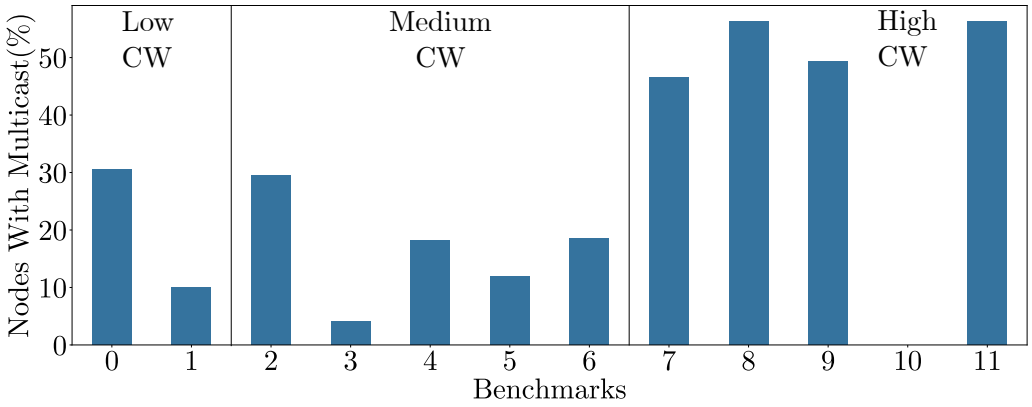

Figure 5. Percentage of nodes with multicast for the 12 evaluated benchmarks.

Generally, benchmarks for dataflow kernel sizes span from 10 to 100 nodes for image processing, communications and DSP applications [Lee 1997]. The benchmark dataset used in this study is publicly available on two website [UCSB 2020, UFV 2023], ensuring the transparency and reproducibility of our findings. To effectively evaluate the network routing capability concerning the Connection Workload (CW), we merge two or more dataflows to generate graphs exhibiting high CW values ranging from 49.21% to 100%. Such an approach emulates real-world scenarios where the concurrent mapping of multiple graphs is equivalent to executing various applications in parallel.

Figure 5 shows 12 publicly available benchmarks in [UFV 2023]. This set was created with the objective of generating a diverse range of connection workloads through the utilization of one or more instances of dataflow graphs from [UCSB 2020, UFV 2023]. It illustrates that out of the 12 selected benchmarks, 3 have approximately 50% multicast nodes, 4 exhibit multicast in the range of 20-30%, while 2 have around 10% multicast nodes. Furthermore, 4 benchmarks have less than 10% of nodes with multicast. The following sections evaluate the mapping strategies in relation to the multicast percentage, additional network levels, and the placement algorithm.

5.3. Heterogeneous Architectures

In the previous section, we discussed the target architecture, which consisted of a homogeneous CGRA with 128 Processing Elements (PE). The network has 256 inputs and outputs; each PE has two inputs and two outputs. However, the application graphs exhibited an irregular degree distribution, meaning that some nodes had only one input while others had two or three inputs. Similarly, specific nodes had multiple outputs, which could be mapped onto a single-output PE due to the network's multicast capability. When dealing with external inputs and outputs, it's important to consider that mapping them on a PE would lead to resource wastage. This inefficiency arises because external inputs do not consume any network output.

Table 2. Processing Elements for Evaluated Architectures.

	Homogeneous		Heterogeneous							
Arch.	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
PEs	128	256	136	140	145	185	187	172	176	191
1 in/out	0	256	76	24	34	114	118	88	96	126
2 in/out	128	0	60	116	111	71	69	84	80	65

We evaluate the adaptability of our mapping approaches in ten architectures, as displayed in Table 2. Among these, two architectures are homogeneous, while eight are heterogeneous. Furthermore, "1 in/out" denotes the count of PEs that possess both 1 input and output connections.

5.4. Low Workload Mapping

Table 3 presents the successful routing percentages for benchmarks 0 and 1, using 30% and 10% of multicast nodes and 49.21% and 53.90% CW, respectively. We evaluated four strategies: Random, Greedy, Local Search (LS), and Simulated Annealing. The results consist of two numbers: the minimum number of extra stages required for successful routing of all edges and the routing rate represented within parentheses. A routing rate of 100 indicates successfully routing all edges. For example, in benchmark 0, comprising 2 copies of the Fir16 dataflow, both Random and Greedy mapping strategies required 2 extra stages to achieve 100% routing, denoted as $2(100)$. However, it is important to note that the possibility of achieving this rate depends on the mapping strategy used.

Table 3. Successful Routing for Low Workload Benchmarks.

			Extra (% Routed)			
Benchmark (DFG)	Arch.	CW (%)	Random	Greedy	LS	10×SA
0 (2 copies of Fir16)	A_0	49.21	**2 (100)**	**2 (100)**	**0 (100)**	**0 (100)**
1 (6 copies of Mults1)	A_5	53.90	4 (88.40)	4 (83.33)	**0 (100)**	**0 (100)**

The initial mapping experiment reveals that the Local Search and Simulated Annealing strategies successfully find the optimal solution for both benchmarks without requiring extra stages. However, the Random and Greedy approaches can only solve benchmark 0, and both fail for benchmark 1 even with adding 4 extra stages. The Random approach outperforms the Greedy approach, achieving an 88.40% success rate in routed edges compared to the best Greedy approach's 83.33%.

In Table 3, we evaluate four different greedy approaches but only depict the best result. These approaches vary in how they traverse the graph and assign mappings to nodes, each with its unique ID. The sequential approach maps nodes in sequential order, starting from node 1, followed by node 2, and so on. In contrast, the random approach randomly selects a node i from the graph for mapping. Additionally, the DFS (Depth-First Search) and BFS (Breadth-First Search) methods traverse the graph in depth-first and breadth-first order, respectively. Regarding the Local Search, we apply it based on the best greedy approach, as described in section 4.4. For the Simulated Annealing, we execute the process 10 times with random initial placements to explore various solutions.

5.5. Medium Workload

Table 4 presents five benchmarks with Connection Workloads ranging from 73.43% to 83.20%. The Random and Greedy approaches fail to achieve successful routing for all benchmarks. The Random solution performs better, as it mitigates issues related to similar input and output codes, as explained in section 4.3. However, even with four extra stages, at best, only around 82% of the edges achieve successful routing using the Random and Greedy approaches, leaving an average of 35 edges unrouted.

Table 4. Successful Routing for Medium Workload Benchmarks.

			Extra (% Routed)			
Benchmark (DFG)	Arch.	CW(%)	Random	Greedy	LS	10×SA
2 (4 Ewf)	A_2	73.43	4 (80.31)	4 (82.44)	4 (99.46)	**2 (100)**
3 (7 Conv3)	A_7	73.82	4 (74.60)	4 (74.60)	4 (99.47)	**3 (100)**
4 (16 Mac)	A_8	81.25	4 (79.32)	4 (78.84)	4 (99.51)	**3 (100)**
5 (2 Ewf 2 Conv3 4 Horner)	A_6	82.81	4 (82.07)	4 (75.00)	4 (99.52)	**2 (100)**
6 (1 Fir16 5 Arf)	A_9	83.20	4 (80.75)	4 (77.93)	4 (98.59)	**3 (100)**

Conversely, the LS (Local Search) approach significantly improves the Greedy method, successfully routing nearly 100% of the edges for a network with 4 extra levels. However, the LS approach must still include routing 1, 2, or 3 edges for the five benchmarks. In comparison, the SA (Simulated Annealing) method effectively solves the problem with extra stages when compared to LS. For 2 out of 5 benchmarks, it requires 2 extra stages. The 3 others needed 3 extra stages.

5.6. High Workload

Table 5 presents five examples with workloads exceeding 87%. In addition to previous benchmarks made by using the application dataflow graph from [UCSB 2020], we add two new graphs to this set. The first graph is a simple pipeline with 256 nodes as a chain to evaluate the mapping behavior in a simple regular graph, the benchmark 10. The second graph, Synthetic, was built based on patterns found in real graphs plus a few cycles. The graph has 16 nodes and 32 edges with an average degree of 2, as shown in Figure 6.

The Random approach proves ineffective for all benchmarks. In contrast, the Greedy approach finds a route with only three additional stages for the 7 and 8 copies of the Synthetic graph. An exception appears in the Pipeline graph, where the Greedy approach reaches the solution without extra stages. However, the SA approach requires 2 additional stages in most cases. Despite this, the Local Search method manages to route

Table 5. Successful Routing for High Workload Benchmarks.

Benchmark (DFG)	Arch.	CW(%)	Extra (% Routed) Random	Greedy	LS	10×SA
7 (5 Synthetic 1 Fir16)	A_4	87.10	4 (87.44)	4 (85.65)	**2 (100)**	**1 (100)**
8 (7 copies of Synthetic)	A_0	87.50	4 (99.55)	**3 (100)**	**0 (100)**	**0 (100)**
9 (7 Synthetic 1 Mults1)	A_3	92.96	4 (90.28)	4 (90.28)	**4 (100)**	4 (99.19)
10 (1 Pipeline)	A_1	99.6	4 (85.09)	**0 (100)**	**0 (100)**	**1 (100)**
11 (8 copies of Synthetic)	A_0	100.0	4 (99.60)	**3 (100)**	**2 (100)**	**1 (100)**

all benchmarks successfully. Only in the case of benchmark 9, Simulated Annealing encounters difficulty routing a single edge out of 238 edges.

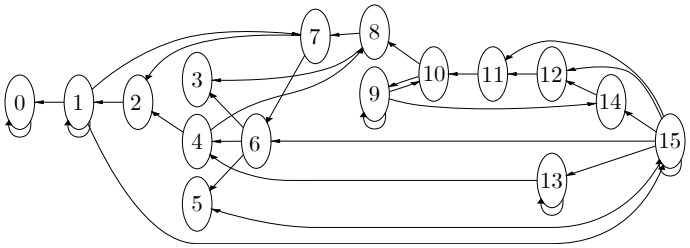

Figure 6. A Synthetic graph to evaluate the mapping strategies.

5.7. Execution Time for the Graph Mapping

In the case of all graphs, assuming there are no routing conflicts, the execution time is a single clock cycle, accounting for both temporal pipeline and spatial parallelism. However, when routing conflicts are present, the execution requires at least two clock cycles. In this section, we measure the execution time for mapping these graphs.

For each benchmark, we executed all algorithms only once, except for SA. In the case of SA, we generated 10 random initial solutions to introduce diversity into the search process. For each of these initial solutions, we ran the SA algorithm and selected the best result out of the 10 runs. Consequently, Table 6 displays the simple average execution time, in seconds, of the 12 benchmarks for each mapping strategy on an i7-7700 3.6GHz. The Random and Greedy approaches execute in just a few milliseconds, while the LS requires a few seconds. Although SA is the most time-consuming strategy, with an average time of around 1.5 minutes, it proves beneficial by saving area by reducing extra stages and improving the CGRA performance, enabling it to route graphs with high workloads efficiently. Considering the vast solution space of 10^{671} possibilities, SA's exploration, utilizing the PE symmetry properties, yields interesting results.

Table 6. Average execution time of mapping strategies, in seconds.

Random	Greedy	LS	10×SA
0.001	0.010	2.437	93.655

6. Related Work

The Omega or shuffle-exchange network [Lawrie 1975] is a multistage network with $\log N$ stages and $\frac{n}{2}$ switches. Initially designed to connect a set of processors to a set

of memories, this network offers conflict-free access to common address patterns, such as rows, columns, diagonals, and more [Lawrie 1975]. Moreover, it has been demonstrated that any one-to-one permutation pattern can be routed using k extra stages [Shen 1995]. However, for more complex patterns, including multicast [Yang and Wang 1998], if the single-pass routing fails, they need to be divided into a minimum number of groups (passes) to establish conflict-free paths for all pairs in each group simultaneously [Hu et al. 1996]. Therefore, this approach slows down the parallel communication by serializing the groups.

Moreover, the complexity of the problem limits the availability of analytical approaches for solving it. Only initial models have been proposed [Gazit and Malek 1989]. In the context of reconfigurable architecture mapping, such as CGRAs [Silva et al. 2019], the challenge extends to include both the placement problem and the routing problem. To address these issues, we introduce two novel placement strategies: Local Search and Simulated Annealing techniques. These strategies effectively unblock the network and help reduce the number of extra stages required.

Regarding the application of SA in multistage networks, a previous study suggested its use to mitigate cross-talk in dynamic routing for small-sized multistage systems [Katangur et al. 2002]. Additionally, within the context of CGRA architecture, SA has been applied to mesh-based architectures [Mei et al. 2005]. However, for multistage architecture, existing approaches have primarily relied on greedy algorithms [Ferreira et al. 2011, Silva et al. 2019].

Furthermore, our mapping application has demonstrated successful graph mapping for ranges spanning from 100 up to 200 nodes. In contrast, exact integer linear programming and SAT-solvers are restricted to handling only 40 nodes [Walker 2019]. For mesh architecture, SA has been able to find exact solutions for graphs of up to 80 nodes [Carvalho et al. 2020]. More recently, graph convolutional networks and reinforcement learning techniques have been utilized to address graphs with 200 nodes [Kong et al. 2023]. Nevertheless, it is important to note that the performance results for these methods are not presented in this work, making a fair comparison challenging.

7. Conclusion

Efficient parallel communication stands as one of the significant challenges. Multistage networks mitigate the cost and offer a network with $O(n \log(n))$ switches, but they introduce challenges in routing. They may lead to blocking, thereby requiring serialization of communication. In the case of large multistage networks with 256 connections, the configuration space explodes to 10^{671} possibilities. Despite that, we have demonstrated that a greedy approach, enhanced by Local Search or stochastic Simulated Annealing techniques, can efficiently perform routing without extra stages. Our placement explores the high symmetry properties available in the CGRA architecture. In addition, our proposed mapping strategy reduces the network cost by a factor of $2\times$ when no extra stages are required or at least 25% when 2 extra stages are required. For 256 connections, our approach enables the routing of graphs with up to 200 operators, effectively doubling the performance compared to previous multistage CGRA frameworks [Silva et al. 2019]. In future work, we plan to integrate our mapping approach in a CGRA framework in high-performance FPGA with HBM memories [Choi et al. 2021].

Acknowledgments

Financial support from FAPEMIG APQ-01577-22, CNPq, and UFV. This work was also carried out with the support of the Coordernação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Financing Code 001.

References

Carvalho, W., Canesche, M., , Silva, L., Nacif, J., and Ferreira, R. (2020). A design exploration of scalable mesh-based fully pipelined accelerators. In *IEEE ICFPT*.

Choi, Y.-k., Chi, Y., Qiao, W., Samardzic, N., and Cong, J. (2021). Hbm connect: High-performance hls interconnect for fpga hbm. In *ACM FPGA*.

Ferreira, R., Vendramini, J., Pereira, M. M., and Carro, L. (2011). An fpga-based heterogeneous coarse-grained dynamically reconfigurable architecture. In *Int conference on Compilers, architectures and synthesis for embedded systems - CASES*.

Gazit, I. and Malek, M. (1989). On the number of permutations performable by extra-stage multistage interconnection networks. *IEEE trans on computers*, 38(2).

Hu, Q., Shen, X., and Liang, W. (1996). Optimally routing lc permutations on k-extra-stage cube-type networks. *IEEE transactions on computers*, 45(1):97–103.

Katangur, A. K., Pan, Y., and Fraser, M. D. (2002). Message routing and scheduling in optical multistage networks using simulated annealing. In *Parallel and Distributed Processing Symposium, International*, volume 2, pages 8–pp. IEEE Computer Society.

Kong, X., Huang, Y., Zhu, J., Man, X., Liu, Y., Feng, C., Gou, P., Tang, M., Wei, S., and Liu, L. (2023). Mapzero: Mapping for coarse-grained reconfigurable architectures with reinforcement learning and monte-carlo tree search. In *ISCA*.

Lawrie, D. H. (1975). Access and alignment of data in an array processor. *IEEE Transactions on computers*, 100(12):1145–1155.

Lee, C. (1997). Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *IEEE MICRO*.

Mei, B., Lambrechts, A., Mignolet, J.-Y., Verkest, D., and Lauwereins, R. (2005). Architecture exploration for a reconfigurable architecture template. *IEEE Design & Test of Computers*, 22(2):90–101.

Shen, X. (1995). Optimal realization of any bpc permutation on k-extra-stage omega networks. *IEEE transactions on computers*, 44(5):714–719.

Silva, L., Ferreira, R., Canesche, M., Penha, J., , and Nacif, J. (2019). Ready: A fine-grained multithreading overlay framework for modern cpu-fpga dataflow applications. *ACM Transactions on Embedded Computing Systems (TECS)*, 18.

UCSB (2020). Bench:. `https://web.ece.ucsb.edu/EXPRESS/benchmark/`.

UFV (2023). Multistage. `https://github.com/lesc-ufv/graphs`.

Walker, M. J. (2019). Generic connectivity-based cgra mapping via integer linear programming. In *Symp on Field-Programmable Custom Computing Machines (FCCM)*.

Yang, Y. and Wang, J. (1998). On blocking probability of multicast networks. *IEEE Transactions on Communications*, 46(7):957–968.