Efficient Interconnect Design for a Data-centric Weather Prediction Accelerator

Ravi, Keerthana

Award date:
2020

Technische Universiteit
Eindhoven
University of Technology

Department of Electrical Engineering
Electronic Systems Research Group

Efficient Interconnect Design for a Data-centric Weather Prediction Accelerator

Master Thesis

Keerthana Ravi

Supervisors:
Prof. Dr. Henk Corporaal
Gagandeep Singh
Kanishkan Vadivel

External Supervisor:
Ronald P. Luijten

Eindhoven, October 2020

Abstract

Consortium for Small-Scale Modeling (COSMO) is a high-resolution weather prediction and climate control model that is being used by a dozen nations. The central part of COSMO model operates on a multi-dimensional grid performing various stencil computations. Unlike elementary stencils, weather stencils consist of compound stencil kernels that have complex data access patterns with limited locality, which do not perform well on general purpose processors. Two representative stencil kernels were mapped onto a data-centric CGRA accelerator designed with an array of processing elements and SRAM units to run these stencils. The mapped stencils required an interconnect with minimum delay and a routing algorithm to route the connections in the stencil kernels. The large number of stencil kernels in the COSMO application demanded a re-configurable architecture and the mapping of the kernels was done with the aim of producing new outputs every cycle. To meet these demands, an interconnect made of re-configurable switchboxes connecting horizontal and vertical wires was chosen. The proposed approach was to use longer wires to connect far off switchboxes directly in order to reduce the overall combinatorial path delay. A heuristic routing algorithm was also developed to route the stencil kernels by making use of the longer wires to reduce maximum delay to achieve better performance. A peephole placement optimization technique was used to further improve the results of the routing algorithm. It was seen that a switchbox topology with longer wires could achieve 50% lower delay, 12% lower power, and 12% lower area values compared to a baseline switchbox topology with wires connecting adjacent switchboxes, for the delay, power and area model used in the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Stencil computations involve sweeping over a large amount of data in a multi dimensional grid to perform computations for each data point with a subset of its neighbours as defined by the shape of the stencil. A wide range of applications make use of stencil computations. One such application is the Consortium for Small-Scale Modeling (COSMO)[1]. It is a weather prediction and regional climate model used by many countries[2]. It contains a dynamical core(Dycore) which performs all required calculations based on a set of governing equations. Physical space is discretized in a rotated latitude–longitude–height coordinate system and projected onto a regular, structured, three-dimensional grid. The spatial discretization applied to solve the governing equations in the Dycore generates different stencil compute patterns.

In stencil computations, the input data is usually larger than the available data caches in general purpose processors and the amount of data reuse is limited to the number of points in the stencil. The number of data accesses to memory is high for each operation. Due to limited memory bandwidth and high memory latency in general purpose processors, applications with stencil computations achieve only a small percentage of the processor's peak performance. A data-centric approach which reduces the movement of data through the different levels of memory hierarchy could alleviate this issue.

The COSMO model has more than 100 stencil kernels, an accelerator designed for it has to be flexible enough to adapt to the requirements of the kernel running on it. Re-configurable architectures provide such flexibility. Re-configurable architectures stand between general purpose processors and application specific integrated circuits in terms of flexibility and performance and have become a popular and economic choice for many applications[3]. FPGAs are a type of re-configurable architecture with fine grained re-configurability at the gate level. Another type of re-configurable architecture is the CGRA. They typically consist of an array of processing elements(PEs) whose functionality can be programmed. These PEs are connected together by a programmable interconnect. Reconfiguration of a CGRA takes place at a coarser granularity, which reduces the reconfiguration overhead.

A novel customized data-centric architecture[4] on a CGRA with an array of units was given to run the COSMO stencil kernels. The COSMO CGRA array consists of two types of units: SRAM units to store input and output fields and function units to perform operations required by the kernels. Two stencil kernels from the COSMO application, horizontal diffusion and vertical advection, have been chosen to represent all the stencil computations in the Dycore. The placement of the operations in the 4x4 vectorized version of the horizontal diffusion and vertical advection kernels on the CGRA array has been given in 2.4.1. The PEs work as a systolic array that generates new results every clock cycle. This requires an interconnect to be designed with single cycle paths. Since single cycle paths are combinatorial, the longest path could contribute to the critical path delay of the architecture which impacts the frequency at which it can operate. Therefore, to compute all weather prediction kernels using the given CGRA design, a flexible interconnect architecture needs to be developed.

1.1 Problem Statement

An interconnect network for the CGRA which has resources for the connection requirements of COSMO stencil kernels has to be designed along with a routing algorithm which can leverage the interconnect for the given placement of operations for the COSMO stencil kernels. The interconnect and routing algorithm design should minimize the overall delay but trade-offs with power and cost in terms of area have to be performed. Also, a certain amount of flexibility has to be maintained while designing the interconnect for other kernels mapped onto this design in the future to achieve 100% routability.

We aim to answer the following research question:

Given a resource mapping of weather prediction kernels on a CGRA-based architecture, can we design a low-latency and flexible interconnect network that can run weather prediction kernels in an energy-efficient way?

The delay of the longest combinatorial path and the power consumed and area occupied by the interconnect should be minimized.

1.2 Contributions

The major contributions of this work are as follows:

1. Design of a highly flexible interconnect topology for a real-world weather prediction application. We optimize the wire length and wire distribution of the interconnect, and further improve the switchbox design.
2. Design of a novel routing algorithm that utilizes the interconnect topology efficiently to minimize overall delay.
3. We demonstrate the use of peephole placement optimization technique for post routing to improve the routing results
4. Design space exploration of different interconnect topologies and switchbox design for delay, power and area efficient design. The optimal interconnect topology had 50% lower maximum delay and 12% lower power and area in the interconnect network compared to a baseline topology based on Blocks[5].

1.3 Outline

The organization of the thesis is as follows. Chapter 2 goes over the novel data-centric architecture for the COSMO weather stencils. Chapter 3 looks at related work in CGRAs, FPGA routing and global routing algorithms. Chapter 4 deals with the interconnect network that will be explored in this thesis. Chapter 5 is about the design of a custom routing algorithm to evaluate the options in the interconnect topology. Peephole placement optimization is discussed in Chapter 6. The different interconnect topologies are evaluated on the basis of power, delay and area and the results are discussed in Chapter 7. Conclusions and scope for future work are discussed in Chapter 8.

Chapter 2

Background

This chapter gives a background on the weather stencil kernels and COSMO CGRA architecture. Some preliminary terms required to understand the rest of the document are explained in Section 2.1. A brief description of the COSMO weather kernels is given in Section 2.2 and a system level overview of the CGRA accelerator is given in Section 2.3. The working of the given data-centric CGRA designed to run the weather stencil kernels is explained in detail in Section 2.4.

2.1 Preliminaries

The design of an interconnect and routing algorithm is similar to that found in FPGAs therefore terminology used in FPGA is borrowed and used interchangeably for the CGRA. Below we explain various terms used in our work:

- Programmable Element(PE): It can be programmed to perform operations (ALU/Multiplier/Division)
- Function Unit: Other name for PE
- Connection: A source and destination pair of PEs that have to be connected
- Net: A source PE and one or more destinations PEs that have to be connected. A net can be decomposed into one or more connections.
- Multi-terminal Net: A source PE and two or more destinations of PEs that have to be connected. A multi-terminal net can be decomposed into two or more connections.
- Netlist: A list of nets to be routed
- Wire: A bus used to form a connection
- Switchbox: A re-configurable switch used to connect wires
- Routing resource: Wires
- Routing Channel: The area between rows and columns of PEs

2.2 COSMO Weather Kernels

The DyCore of the COSMO model performs numerous stencils computations and it accounts for 70% of the runtime. Horizontal Diffusion and Vertical Advection are two kernels found in the dyCore that are representative of data access patterns found in the entire COSMO model.

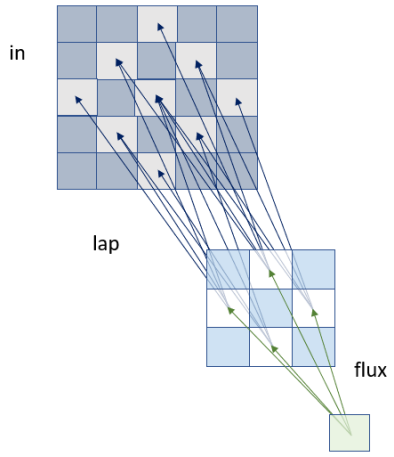


Figure 2.1: Data dependency for output field of horizontal diffusion

An example of the data dependency in the horizontal diffusion kernel is illustrated in Figure 2.1. Two elementary stencils, laplacian and flux are applied consecutively on the input fields[1] in order to obtain an output field. This large data dependency leads to complex memory access patterns because of which these kernels have a low compute intensity. Therefore, the weather stencils are not suitable for existing caching techniques that are used on our current CPU-based systems.

2.3 Accelerator Architecture

A data-centric CGRA-based accelerator[4] was developed to run the COSMO stencil kernels. The proposed architecture is depicted in Figure 2.2. All the fields required by the COSMO kernels are stored in the accelerator DRAM. The fields required for a stencil computation are pre-fetched onto an on-chip SRAM and after computation the output fields are written back to DRAM. Stencil computation overlaps input field pre-fetch/writeback to DRAM. The compute resources have been matched to meet the DRAM bandwidth.

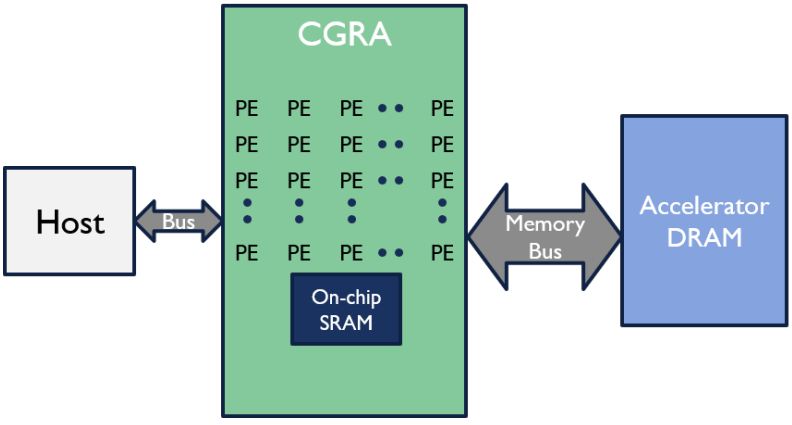


Figure 2.2: Accelerator Architecture

[1]The fields represent prognostic variables for temperature, wind etc and tracers for pollen, aerosols etc, stored in 3D grids.

2.4 COSMO CGRA

The data-centric CGRA designed to work as an accelerator to run the stencil kernels in the COSMO application has an array of 38x38 programmable elements(PEs) and SRAM units. Figure

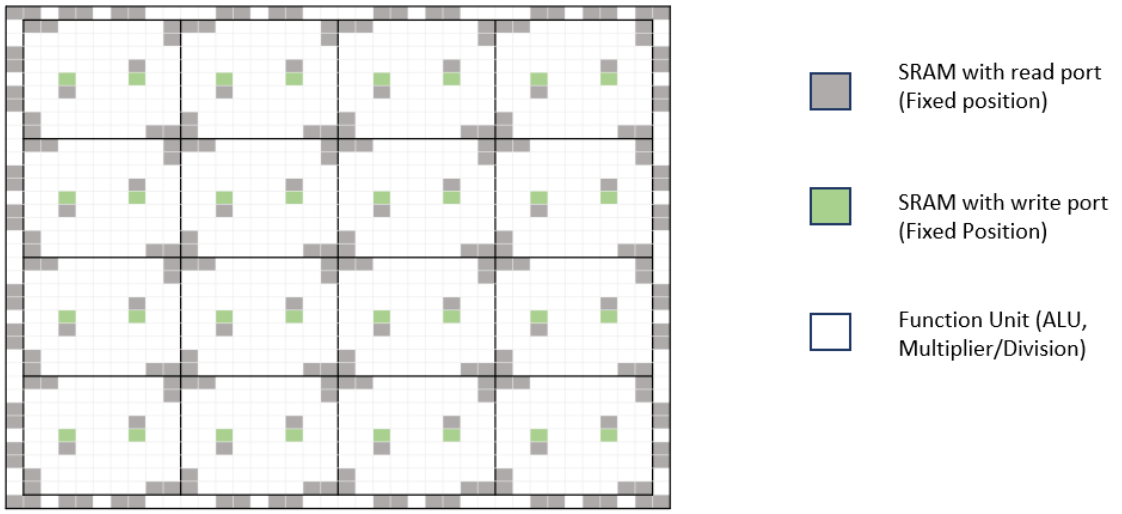


Figure 2.3: PEs in the 38x38 CGRA architecture

2.3 shows a pictorial depiction of the COSMO CGRA array. The white units in the figure can be programmed to work as function units as required by the operations in the stencil kernel. The grey and green units represent SRAMs with read and write ports to the DRAM respectively. A 9x9 block is formed with the function units and SRAM units. This 9x9 block is duplicated to form a 4x4 array of 9x9 blocks along with an outer layer of extra function units and SRAM units for the halo points, which are the boundary points used to compute an output field.

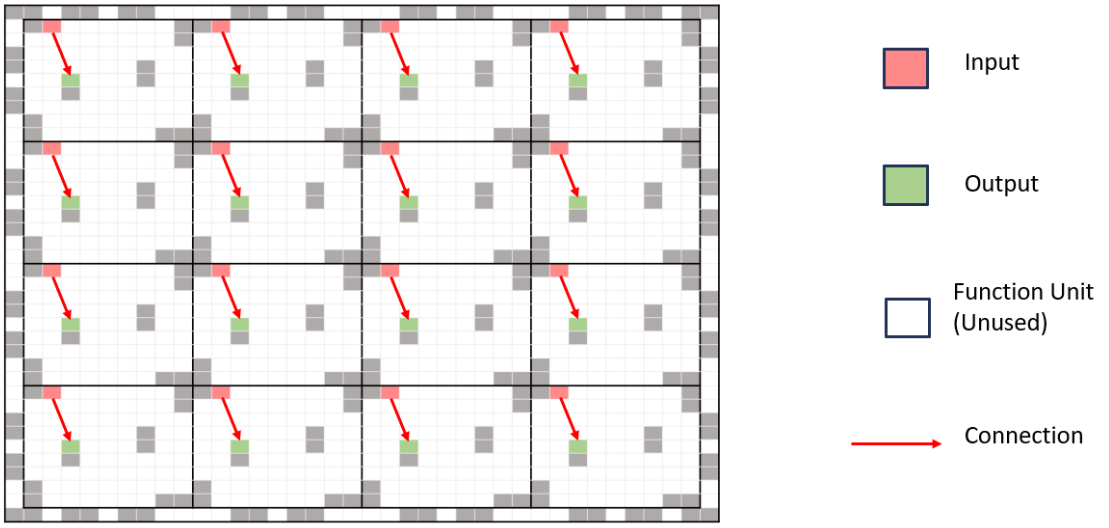


Figure 2.4: Copy stencil mapped onto 38x38 CGRA

In Figure 2.4 we show a 4x4 vectorized version of a simple copy stencil from the COSMO model. The input fields are loaded onto the red SRAM blocks and the output fields are written back to the DRAM from the green SRAM blocks. Each output field is assigned to each 9x9 block,

thus creating a 4x4 vectorization to produce 16 outputs at a time. Since the loading of inputs and the write back of output fields is pipelined, the architecture works like a systolic array which generates new results every cycle. In this case 16 outputs are computed every cycle. In a similar manner other stencils can be mapped to the architecture. No other operation is required between the input and output fields for the copy stencil. For other stencil kernels from the COSMO model, the required operations can be mapped to the PEs in the 9x9 blocks. The computation of each output field depends on reuse of input fields defined by the shape of the stencil. For each 4x4 set of output fields, the input fields are loaded once and connections between the PEs deliver the input fields to all the dependent operations to calculate the respective output fields.

2.4.1 COSMO kernels mapped on the CGRA

The mapping of the COSMO kernels is done in a similar manner, as explained for the copy stencil. Each output field is mapped onto a 9x9 block along with the corresponding input field and operations. The shape of the stencils are such that the data reuse is limited to the immediate neighbour of a field. Therefore, to compute an output field, the transfer of required data is limited to adjacent 9x9 blocks. It can also be seen that the SRAM blocks which would be loaded with the input fields are placed at the corners of the 9x9 blocks. As a result, the connection lengths are kept to a minimum and this prevents the need to add long buses running across the entire design.

Horizontal Diffusion

Figure 2.5 shows the placement of operations for a 4x4 vectorized horizontal diffusion kernel. A lot of SRAM units are unused because only one output field is computed by each block and the input fields required by it are few. In total, 16 output fields are computed in each cycle.

- Maximum Manhattan Distance: 14
- Maximum distance in x/y direction: 10
- Number of Nets: 901
- Number of Multi-terminal nets: 185
- Number of Connections: 1178

Vertical Advection

Figure 2.6 shows the placement of operations for a 4x4 vectorized vertical advection kernel. Each block computes two types of output fields. All the SRAM units are used to load the required input fields and write back the output fields computed. In total, 32 output fields are computed every cycle.

- Maximum Manhattan Distance: 10
- Maximum distance in x/y direction: 7
- Number of Nets: 952
- Number of Multi-terminal nets: 136
- Number of Connections: 1056

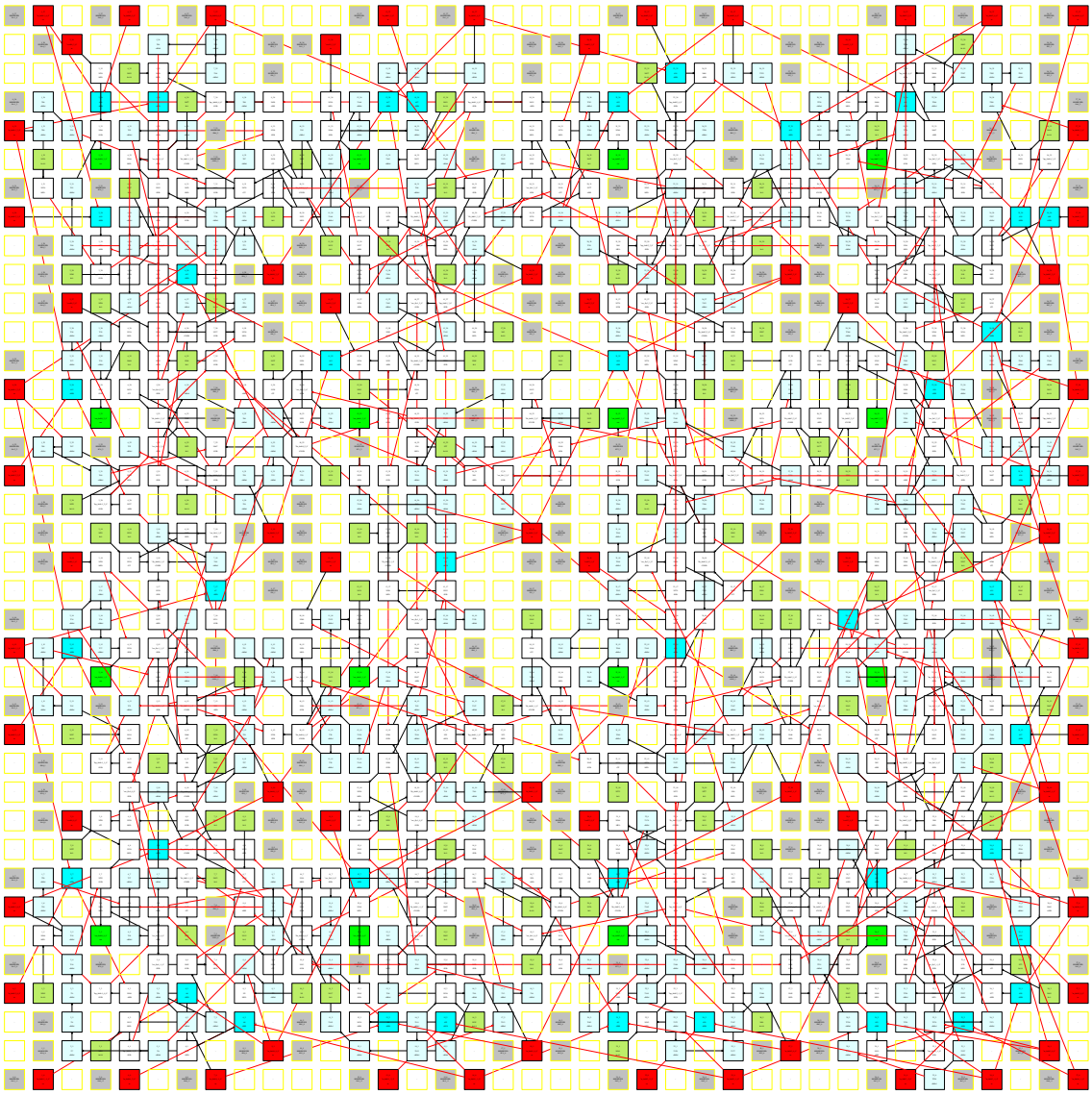


Figure 2.5: Horizontal Diffusion

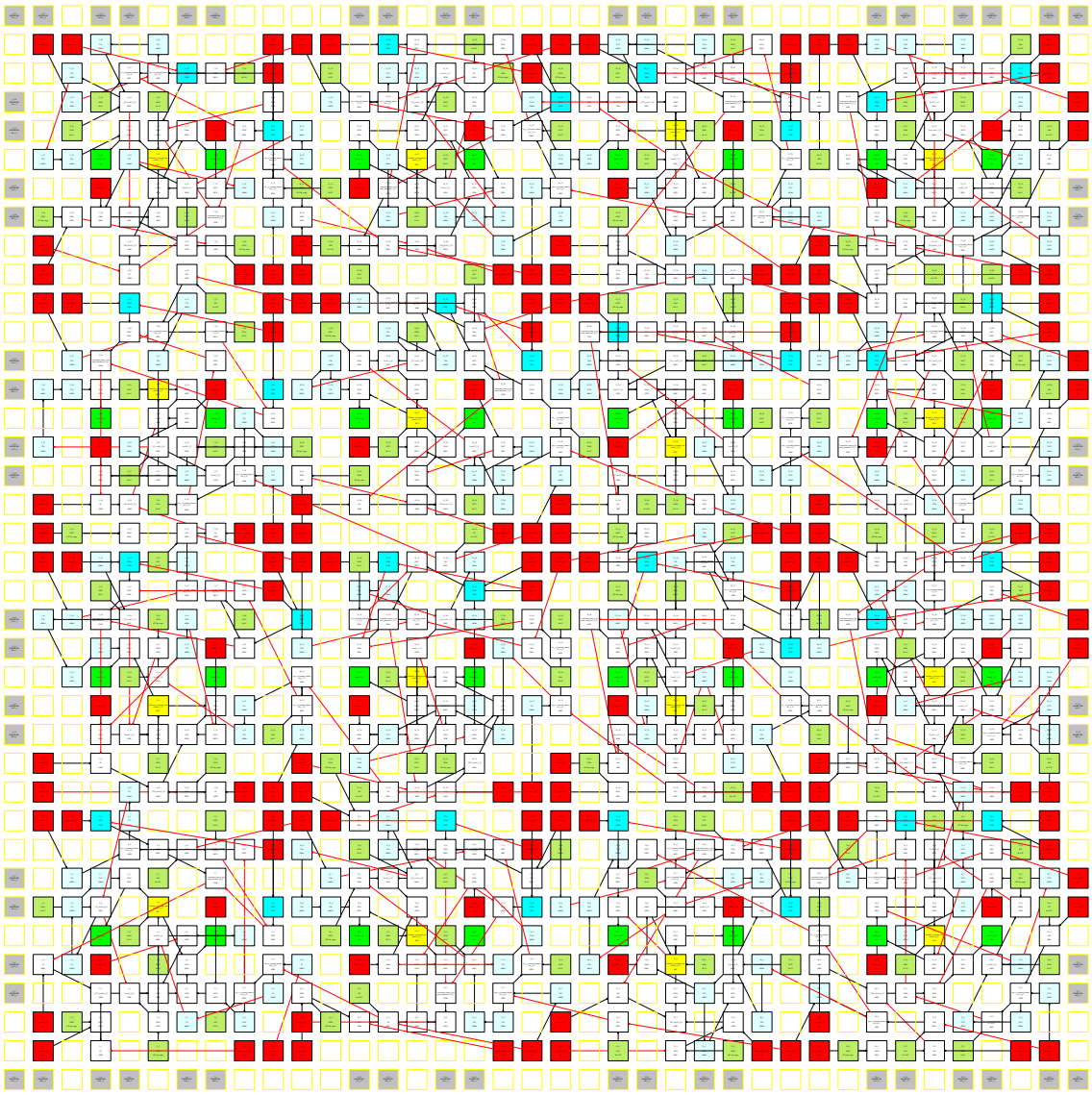


Figure 2.6: Vertical Advection Forward

Distribution of distance

The Figure 2.7 shows a distribution of the distance between source and destination terminals for all the connections in the netlist of Horizontal Diffusion and Vertical Advection. This distribution gives an insight into the connection lengths required by the mapped design. Most of the connections have a distance of 1 and 2, therefore the interconnect has to be able to connect adjacent PEs. The connections with larger distances are few, but they will increase the maximum delay significantly. It is necessary to have a low latency path for these.

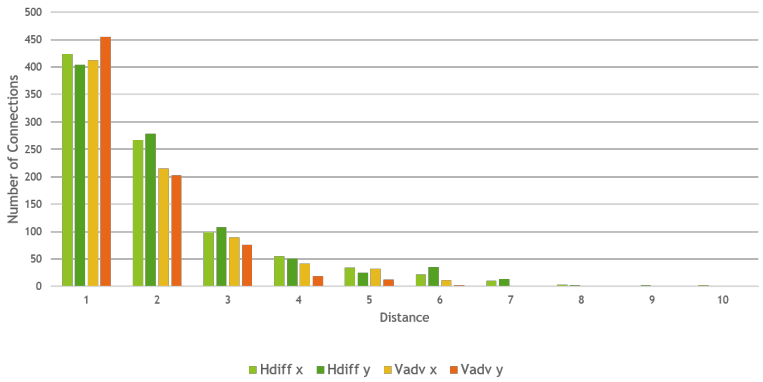


Figure 2.7: Distribution of distance

It can also be seen that the number of connections in Horizontal Diffusion is more than in Vertical Advection. The number of shared input fields for the computation of adjacent output fields is more in Horizontal Diffusion. For the same reason, the distance of connections in Horizontal Diffusion is greater than that of Vertical Advection. Therefore, designing an interconnect that satisfies the connection requirements of Horizontal Diffusion to achieve 100% routability and minimizes the delay of the longest connection would also achieve the same for Vertical Advection.

Chapter 3

Related Work

The given mapping of the opersations in the weather stencil kernels requires an interconnect. In order to evaluate if an interconnect topology is suitable for the COSMO CGRA, a routing algorithm is required. Therefore, work done in the past to solve a similar problem in CGRAs was looked into in Section 3.1. Routing is also a problem addressed in the physical design process of a custom ASIC or FPGAs, therefore routers developed for these were also explored and Section 3.2 presents the key takeaways from them.

3.1 Routing in CGRA

In CGRAs, a modulo routing resource graph (MRRG) is used to represent a hardware element in the CGRA in time and space and an edge represents a connection between hardware elements. An application is represented as a dataflow graph (DFG) where the vertices represent the operations to be performed and edges represent the dependency relations between operations. The routing problem is usually solved as part of mapping the DFG to the MRRG. In most of these works[6, 7, 8, 9, 10], it is assumed that the PEs in the CGRA have only nearby connections, a maximum of two hops away. In [6], with this assumption, an integer linear approach is used to map application DFGs to the MRRG. Since ILP does not scale well for larger architectures and benchmarks, a relaxed routing constraint was used for these where a constrained number of routing paths driven by different PEs to share a routing resource. Similarly, in [7] and [8], the CGRA used had a mesh architecture with connections only between adjacent PEs. In [7], the mapping of the DFG to the MRRG is optimized by reducing the NoC congestion. For this, a congestion index was introduced along with workload balance which determined how many operations were mapped to each PE. In [8], the mapping of DFG to CGRA is relegated to finding a rectilinear Steiner tree. This was done using a 2 step ant colony optimization approach. The CGRA modelled as a graph and the ants were used to find Steiner points to form a Steiner tree. Each ant calculates a probability for multiple Steiner tree candidates through local heuristic and global pheromone and makes a choice accordingly. Distance of a vertex from the ant was used as a local heuristic and the total length of the tree was used as the global pheromone.

HyCube[11] is one of the few architectures that implement single cycle multi-hop connectivity, which is required by the weather stencil kernels. A minimal cost path to route the dependencies from the mapped predecessors of each unmapped operation is determined using Dijkstra's algorithm and a cost function, which aids in mapping the operation to a PE. The cost function is meant to discourage the use of congested resources and some architecture-specific costs are also included to discourage non-memory operations from being mapped to PEs meant for memory operations. If a valid mapping is not found, the initiation interval[1] is increased and the mapping is executed again.

[1]The number of clock cycles between the start times of consecutive loop iterations.

The work done to solve the routing problem in CGRAs is not applicable for the given COSMO CGRA. Routing is solved as part of the mapping problem, where an operation is only mapped to a PE if it is possible to route to its predecessors or successors and all the heuristics used try to improve this mapping. In the COSMO CGRA, the mapping is given and cannot be modified until the post routing stage. Also, in the related work the initiation interval is increased if a mapping is not found and the process is repeated. The initiation interval for the weather kernels on the COSMO CGRA has been fixed to 1, hence increasing the initiation interval is not desirable. Most of work done in CGRAs also assume that connections exist only between adjacent PEs but for the COSMO CGRA connections between PEs far from each other are required to obtain single cycle multi-hop connectivity.

3.2 Global and FPGA Routing

Over the years, there have been many iterative routers that have proven to be successful in obtaining good results and are also scalable with the benchmark and architecture size. Pathfinder[12] is one such router where a negotiated congestion mechanism was introduced. The router negotiates between performance and routability based on the criticality of the net. VPR[13], which is one of the most used open source FPGA router is based on pathfinder, with some modifications to the cost function and some enhancements to speed up the algorithm and quality of results. In [14], a lazy approach is taken while rerouting nets. Each net's routing solution is stored as a routing tree and only subtrees which have overuse of routing resources are ripped apart instead of the whole routing tree. Connection router[15] also has a similar approach, where each connection of a net is ripped up instead of the whole net along with a cost function to promote sharing of resources in a multi-terminal net. Croute[16] takes this concept further by improving the cost function so that the order in which the connections of a net are routed doesn't impact the amount of sharing. A lot of the heuristics discussed in these routers are applicable to the routing problem in the 38x38 CGRA, which requires minimization of maximum delay path and minimization in router runtime.

Many different concepts were also introduced in global routing, especially by the routers developed for the ISPD Global Routing Contest[17] in 2007,2008. The best performing routers[18, 19, 20], are based on generating rectilinear Steiner minimum trees(RSMT) using heuristics to decompose a multi-terminal net into two terminal connections, which are then routed using a shortest path algorithm. The topology of the Steiner trees generated is then improved with congestion information using techniques like edge shifting. Global routers like [18, 21, 20] also discuss variations of congestion cost functions. They argue that the use of a logistic function with a slower increase in cost compared to a linear function would be more successful in finding routing solutions. This may not always hold true and has to be tested out as the application of the cost function and the parameters used in the cost function could also differ.

. In conclusion, FPGA routers solve a routing problem very similar to the one present in the COSMO CGRA where the number of routing resources are limited and the routing algorithm works on reducing the maximum delay in a combinatorial path. Global routers solve the routing problem with a different approach by generating RSMT which is useful for reducing the overall wire-length of the routing tree but in FPGAs and the CGRA discussed here, it may not lead to the shortest delay routing solution as the use of long wires is not taken into account. Among FPGA routers, the negotiation congestion mechanism in [12] to prioritize reducing delay for critical connections would work well in obtaining these objectives but it would require some modifications to be aware of the architecture and utilize it effectively. In [13], the timing driven router aims at reducing delay, but it requires delay values as an input for each routing resource. Since, the interconnect for the COSMO CGRA has not been designed yet, a generic routing algorithm is required for it, without architecture or technology specific delay values. The concepts in [15, 16] to improve sharing of resources are applicable to the COSMO CGRA, but they should not work against the primary objective of reducing maximum delay. Suitable modifications have to be made to ensure the same.

Chapter 4

Interconnect Network

The interconnect used to connect the array of PEs in a CGRA is usually a mesh connecting each PE to its adjacent neighbours or nearby PEs, as seen in architectures like ADRES, MorphoSys[3]. The given mappings of the COSMO weather kernels in Section 2.4.1 demand an interconnect with single cycle connectivity between PEs to produce results every cycle. Immediate neighbor connections between PEs will not suffice. The interconnect also has to be re-configurable to be able to run different stencil kernels on the CGRA.

In Blocks[5], an FPGA style network is used, where PEs are connected to re-configurable switchboxes, and directional data buses(wires) connect the entire network of switchboxes. The configuration of the switchboxes allows a PE to connect to any other PE in the 2D array. Therefore, this interconnect was chosen as the baseline for the COSMO CGRA. This chapter discusses the design of the interconnect network for the COSMO CGRA, based on Blocks. In Section 4.1, the topology of the interconnect network is discussed and in Section 4.2.1, the design of the switchboxes used to build such an interconnect is explained along with the optimization done for delay, power and area.

4.1 Topology

A switchbox network as seen in Blocks[5] is depicted in Figure 4.1 where SWB represents the switchbox and W_x and W_y represent the horizontal and vertical channel widths. A wire entering a switchbox can be connected to any wire exiting a switchbox using a programmable switch or a multiplexer. In Blocks, the horizontal and vertical channel widths can be defined at design time. Increasing the channel width improves the routability of the design but results in an increase in the number of wires in turn increasing size of switchboxes. This increase in the number of wires increases the delay, power, and area occupied by the switchbox. Therefore, it is necessary to distribute an adequate amount of wires in the topology for routability while trading off for power, delay and area. Also, defining the channel width would distribute the same number of wires throughout the horizontal and vertical channels of the CGRA. An uneven distribution of wires, where some switchboxes have more wires between them than others, might prove to be useful to reduce the size of some switchboxes without much impact on routability.

Another issue that arises in the Blocks[5] interconnect architecture is the long combinatorial paths required to connect far-off PEs which involve traversing through several switchboxes. The delay added by each switchbox is high, around 0.2ns based on previous research in 40nm technology. This could reduce the achievable clock frequency for this design significantly. It is, therefore, necessary to add connections with less delay between PEs which are far from each other. Henceforth, we will refer to the number of switchboxes traversed from the source switchbox of a connection to the destination switchbox as the hop count and focus on reducing this value. Use of longer wire of different lengths in the switchbox network, as seen in Figure 4.2, is common in FPGAs[22]. FPGAs typically employ the staggered non-uniform segmentation model[23] where

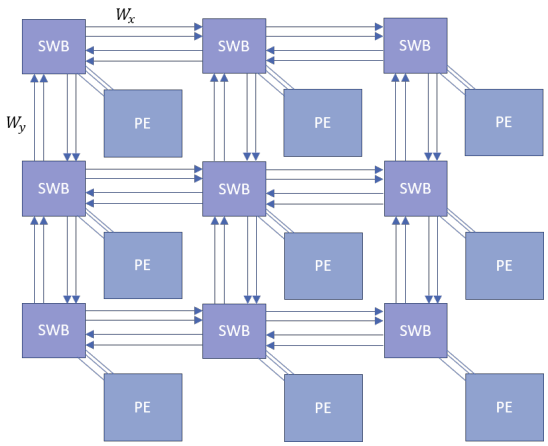


Figure 4.1: Switchbox Network

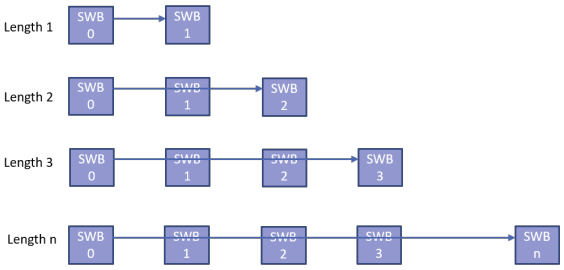


Figure 4.2: Different Length Wires

for an n-length wire, n tracks are added in a staggered fashion to connect switchboxes in a row or column. In the COSMO CGRA, the PEs which require long connections are sparse, and need not be located adjacent to each other, therefore a different approach was used to explore the design space.

4.1.1 Wire Lengths

It was necessary to determine the lengths of the wires required to reduce the overall hopcount while also keeping the utilization of long wires in mind. A look at the distribution of distances from the placement in Section 2.7 gives an idea of the required wire lengths. The maximum number of connections are at a distance of one and the number of connections with a distance greater than 6 are very low. Based on this, two assumptions were made. Length 1 wires would be used in all switchboxes and wire lengths greater than 6 would be ignored. The next step was to determine the combination of wires required to reduce the hop count of all the connections. For the Manhattan distance of each connection from the placement of the weather kernels, maximum hop count obtained using each combination of wires considered was determined for all connections. The results of this are seen in Figure 4.3. The [6,2,1] combination was chosen as it was possible to reduce the maximum hop count from 14 to 4 with a combination of only three wire types.

4.1.2 Distribution of wires

The next step was to distribute the wires such that the number of wires was adequate for routing while also keeping the maximum hop count to a minimum. The Algorithm in 1 was used to generate a bunch of topologies with different distribution of wire segments. Length 1 wires were distributed throughout the design. Longer wires of length i were added to every n_i^{th} switchbox in

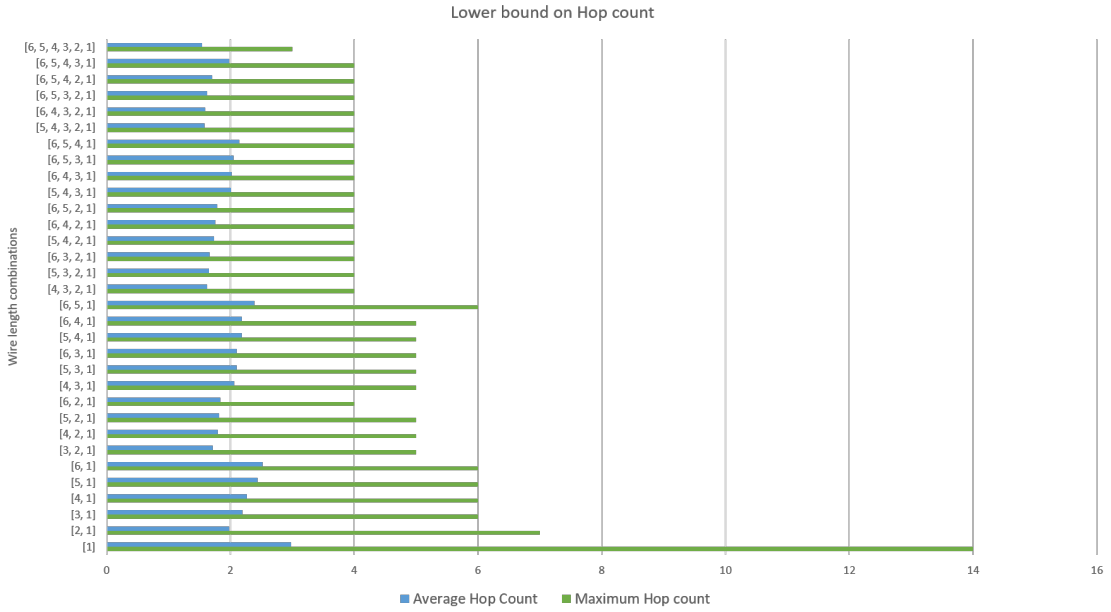


Figure 4.3: Lower bound on Maximum hop count for all wire length combinations

a 9x9 block. The value of n_i was bounded by n_{max} which was set to 9 so that in each 9x9, either in every row or in every column there is at least one switchbox with longer wires. Each 9x9 block was replicated to form 4x4 blocks of 81 switchboxes each. In this manner, the design is scalable to form bigger arrays required for higher orders of vectorization. Longer wires were added to the outer layer of switchboxes which do not belong to any 9x9 block if their position corresponded to n_i.

A check was also added to check if the bi-section bandwidth[1] of the topology was adequate for the bandwidth demanded by the placement of operations of each kernel at that bi-section. This was checked for all bi-sections. If this check is not satisfied for a kernel, it is not possible to get a valid routing solution for that kernel on that topology. Based on the distribution of distances of the connections in the kernels in Figure 2.7, the number of length 2 wires required is always greater than the number of length 6 wires required. Therefore, the search space was reduced by only exploring topologies where $n_2 \leq n_6$.

Algorithm 1 Algorithm to generate topologies

```
L: Nets to be routed
T: Empty List of useful topologies
for n_6 = 1 to n_max do
    for n_2 = 1 to n_{n_6} do
        block= initializeBlock(length=1)
        for swb_position in block do
            if swb_position % n_6==0 then
                addWireToSwb(block,swb_position,length=6)
            if swb_position % n_2==0 then
                addWireToSwb(block,swb_position,length=2)
        topology= generateTopology(block)
        if checkBi-SectionBW(topology, L) then
            T.append(topology)
```

[1]Bandwidth available between two partitions of a bi-section through the entire interconnect network

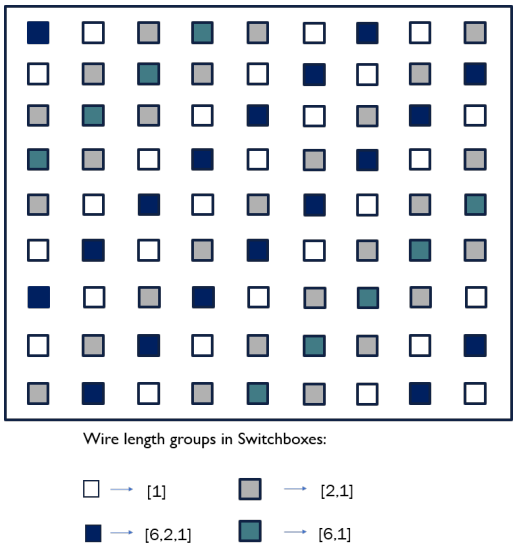


Figure 4.4: An example topology for $n_6 = 3$ and $n_2 = 2$

A total of 44 topologies were generated. In order to accurately gauge if a topology is suitable, the results of the routing algorithm are required. The routing algorithm is explained in Chapter 5. Choosing a suitable topology and designing a routing algorithm to route the connections requires hardware-software co-design. Therefore, a subset of these topologies were used to test the routing algorithm, the reults of which are in Chapter 7. The final routing algorithm was used for the design space exploration of all topologies in Chapter 7.

4.2 Switchbox Design

Different types of switchboxes with different number input and output wires were considered in Section 4.1. Each function unit requires a maximum of one output and three inputs based on the operations in the weather stencil kernels. Therefore, each switchbox was assigned one FUOutput as its input and three FUInputs as its outputs along with input and output wires in the TOP, BOTTOM, LEFT and RIGHT directions. An output wire can be connected to any of the input wires using a multiplexer. The control bits given to the multiplexer determines the connectivity. These wires are depicted in Figure 4.5 for a switchbox with wires of lengths 6, 2 and 1.

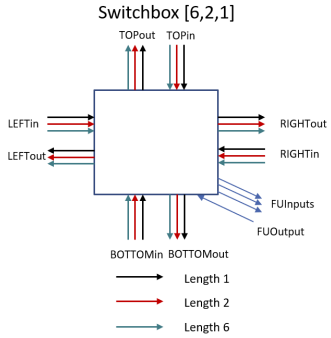


Figure 4.5: Example of a Switchbox

(a) Connectivity of a wire in RightOut

(b) Connectivity of a wire in FUInput

Figure 4.6: Switchboxes with full connectivity

4.2.1 Connectivity in the Switchbox

Switchboxes can be optimized for delay, power and area by reducing the connectivity between the wires. While this results in a reduction in delay, power and area the reduced connectivity might result in detours in the routing result or no possible routing. In [24] , it is shown that even by limiting the connectivity of each wire exiting the switchbox to three wires entering the switchbox, the percentage of routing completion obtained was as high as 99%. For this type of reduced connectivity to have minimum impact on routability, the number of routing tracks in each channel has to be set to a high value. In the switchboxes considered for this CGRA, the maximum number of routing tracks is 3 and the wires used in each of the tracks are of different types. The selection of wire lengths was done on the basis of using them in any combination. Therefore, reducing the connectivity based on the work done in FPGAs is not suitable but some optimizations can be done based on the nature of the given placement. The different types of connectivity explored are described below.

Full connectivity

Each switchbox output is connected to all switchbox inputs as seen in Figure 4.6. Each FUInput output wire is connected to 13 inputs including FUOutput. Each output wire in the four directions is connected to 10 input wires.

Reduced Length 6 Connectivity 1

Since the sharing of data happens only among adjacent 9x9 blocks and the input fields are stored in memory blocks which are located at the corner of a 9x9 block, the length of connections remains below a distance of 9 units in any direction for most of the nets. Based on this, the connectivity of length 6 wires in the same direction was restricted. With this, the connectivity of a length 6 output wire was decreased by to 12. On the whole, 4 connections were removed, one for each of the four length 6 wires.

(a) Reduced Length 6 Connectivity 1

(b) Reduced Length 6 Connectivity 2

Figure 4.7: Connectivity of Length 6 wires

Reduced Length 6 Connectivity 2

While the connection lengths in a particular x or y direction could remain under 9, the Manhattan distance is slightly greater. Therefore, restricting the connectivity of orthogonal length 6 wires might have an adverse effect on routability and maximum hop count. This was implemented to examine the outcome. With this, the connectivity of a length 6 output wire was decreased to 10. On the whole 12 connections were removed, 3 for each of the four length 6 wires compared to the fully connected switchbox.

The impact of the decreased connectivity in switchboxes on routability and delay, area and power are discussed in Section 7.3.1 and Section 7.3.2 respectively, in Chapter 7.

Chapter 5

Routing

In the routing problem for the CGRA, the number of routing resources are fixed with re-configurable switchboxes to form connections between them. In order for the the solution to the routing problem to be legal, each routing resource can be used by only one net, although resources can be shared among the the connections in a single net.

In the CGRA, the objective is to find a legal routing solution while also minimizing the maximum hop count in order to be able to run the COSMO application at a higher frequency, assuming that the critical path is determined by the maximum hop count path. The routing problem is usually divided into two phases:

- Global Routing: In global routing, a coarse route is found through the layout for each net. The coarse route specifies the channels used to connect all the terminals of each net.
- Detailed Routing: In detailed routing, specific wire segments are assigned to each net along channels allotted to the net during global routing.

Routing resource graphs(RRG) are used to represent the layout of an architecture. Coarse routing resource graphs can be used to perform global routing, whereas fine routing resource graphs can be used to perform combined global and detailed routing. For the CGRA, the vertices of the graph were used to represent the switchboxes connected to PEs and edges were used to represent connections between switchboxes. It is different from other coarse RRGs because different edges represent different length wires but a capacity is used to represent a group of same length wires. The capacity of an edge determines the number of nets that can use it. Due to this, the RRG is not as coarse as in global routing where the edges are used to represent a channel. With the CGRA RRG, it is possible to find a combined legal routing solution where there is no edge where the capacity is exceeded by the number of nets using the edge. It is also coarser than the graphs used for combined routing in FPGAs, hence the routing algorithm would run faster on this CGRA RRG.

In the following sections, the routing problem for the COSMO CGRA and the solution is explained in detail. In Section 5.1, a formal description of the routing problem and its objective is given. Section 5.2 first gives a background on existing routing algorithms and then explains the modifications made to it in order to develop a suitable routing algorithm for the COSMO CGRA.

5.1 Formal Description of the Routing Problem

A formal description of the routing resource graph and the routing problem is given in this section. Let $G(V, E)$ be a directed graph where V is the set of switchboxes and E is the set of wires connecting each pair of switchboxes. The following attributes are assigned to each edge:

- Edge capacity: $c_e \forall e \in E$
- Edge cost: $w_e \forall e \in E$

The netlist is represented by a set N given by, $N = \{T_1, T_2....T_n\}$ where $T_i \subseteq V$. $R(T_i)$ is a set of all possible directed graphs that connect the terminal points in T_i. If $t(V', E') \in R(T_i)$, then $t(V', E')$ is a subgraph of $G(V, E)$ where $V' \subseteq V$ and $E' \subseteq E$. The objective function for the routing problem can be formulated as,

$$Minimize\ max(\{f(i) : i = 1, ...n\}) \tag{5.1}$$

where,

$$f(i) = \sum_{t \in R(T_i)} \alpha_{ti} x_{ti} \tag{5.2}$$

where x_{it} is a binary decision variable that is equal to 1 if a net T_i is routed with directed graph t, α_{ti} is the maximum hop count in routing tree t.

The constraints for the objective function are,

$$\sum_{i=1}^{n} \sum_{t \in R(T_i)} a_{te} x_{it} \leq c_e \forall e \in E \tag{5.3}$$

where a_{te} is 1 if routing graph t uses edge e.

$$\sum_{t \in R(T_i)} x_{it} = 1 \forall i = 1, ..., n \tag{5.4}$$

$$x_{it} \in \{0, 1\}\ \forall R(T_i)\ \forall i = 1, ..., n \tag{5.5}$$

5.2 Routing Algorithm

In order to find a routing solution for the given nets, a routing algorithm is required which achieves the given goal of minimizing the maximum delay path by minimizing maximum hop count. The main issue here is the complexity of the routing problem. The decision problem of whether it is possible to find a legal routing for a given netlist on the CGRA is NP-Complete[25]. There is no known polynomial time algorithm that can solve this problem. Integer Linear programming or boolean satifiability can be used to obtain the optimal solution for the problem in a reasonable amount of time, provided the routing resource graph and netlist are small. That is not the case with the given application. Runtime is of prime importance as the stencil kernels have to compiled periodically and the number of kernels is also large. Hence, iterative algorithms with heuristics were looked at, to devise a suitable routing algorithm.

In FPGAs, iterative algorithms based on the pathfinder negotiation congestion mechanism described in [12] have been used extensively and have proven to be successful in obtaining good routing solutions. The pathfinder algorithm was adapted to solve the CGRA routing problem along with modifications to achieve the primary objective function while also improving other aspects of the algorithm.

5.2.1 Background

In iterative algorithms, each net is routed one after the other. The routing of each n-terminal net involves breaking the net into at least n-1 connection pairs of source and destination and routing each of those using a shortest path algorithm. A cost is associated with each routing resource, and its value is dependent on the usage of the resource. This cost helps in guiding the shortest path algorithm to avoid congested resources, if possible. Nets using a congested resource can be ripped up and rerouted in subsequent iterations until no resource is congested.

Shortest Path Algorithm

The routing of a source and destination connection can be done by using breadth first search to find the shortest path in the routing resource graph. In one of the earliest routers[26], this was done by expanding a wavefront from the source terminal in all directions until the destination terminal was reached. When costs are associated with the routing resources, the routing resources graph is a weighted graph and Dijkstra's shortest path algorithm can be used to find the routing path between source and destination. Cost of using a resource $e(u, v) \in E$:

$$f(e) = f_{prev} + w_e \tag{5.6}$$

where f_{prev} is the minimum cost of reaching u from source.

Dijkstra's algorithm performs an exhaustive search in all directions until the destination vertex is found through the lowest cost path. The search can be sped up by adding a heuristic to expand the search towards the direction of the destination terminal by adding an estimated cost to the destination. This is called the A-star heuristic and usually the distance from the destination is used as the estimated cost.

Cost of using a resource $e(u, v) \in E$:

$$f(e) = f_{prev} + d_v + k \times w_e \tag{5.7}$$

where d_v is estimated cost to reach the destination from v and k is a user defined value.

Pathfinder Negotiation Congestion

In an iterative rip-up and reroute approach, the order in which the nets are routed and the decision made on which nets to re-route affect the outcome of the routing algorithm. This was overcome in an approach introduced by [27] where in each iteration all the nets are ripped up and re-routed one by one. The cost of congested resources are increased permanently so that nets may avoid congested regions in subsequent iterations of routing.

Pathfinder[12] takes this concept further with the addition of a negotiation between routability, which involves avoiding congested routing resources and performance, which involves finding the shortest path from source to destination. Critical connections have a higher priority while negotiating for resources by placing a lesser weight on the congestion penalty. For the CGRA router, criticality is set to one for the first iteration and for all subsequent iterations, it is determined by the hop count of the routing path for each connection.

For a connection pair (s, d) in a net T, the criticality $\alpha_{(s,d)}$ is given as:

$$\alpha_{(s,d)} = \frac{D_{(s,d)}}{D_{max}} \tag{5.8}$$

where $D_{(s,d)}$ is the hop count of the route from s to d and D_{max} is the maximum hop count for all connections. While routing the connection (s, d), the cost of using a resource $e(u, v) \in E$ is now given as:

$$f(e) = f_{prev} + \alpha_{(i,j)} \times d_v + (1 - \alpha_{(i,j)}) \times w_e \tag{5.9}$$

The congestion cost w_e is given below, and is based on a present congestion penalty p_e, historical congestion penalty h_e and b_e is the base cost associated with e.

$$w_e = (b_e + h_e) \times p_e \tag{5.10}$$

The congestion penalties are determined by the usage, u_e and the capacity c_e of a resource. The present congestion penalty is update each time a net is routed, for the resources used to route that net.

$$p_e = 1 + p_{fac} \times \max(0, 1 + u_e - c_e) \tag{5.11}$$

The historical congestion penalty is updated after each iteration and is dependent on the value of the previous iteration.

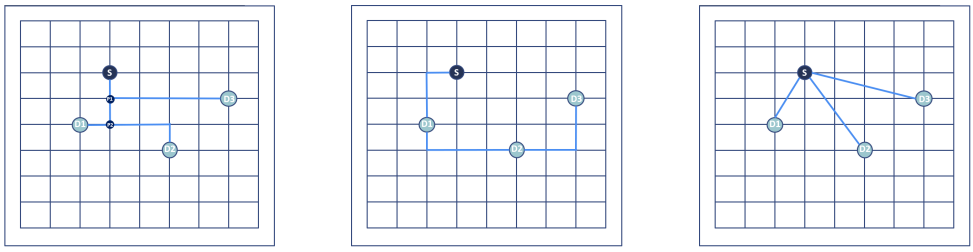
$$h_e^i = h_e^{i-1} + h_{fac} \times \max(0, u_e - c_e) \tag{5.12}$$

p_{fac} and h_{fac} are the congestion factors and the change in their values determine the routing schedule. With the historical congestion penalty, it is possible to avoid historically congested routes and with the present congestion cost, it is possible to avoid newly congested path.

.

5.2.2 Multi-terminal Nets

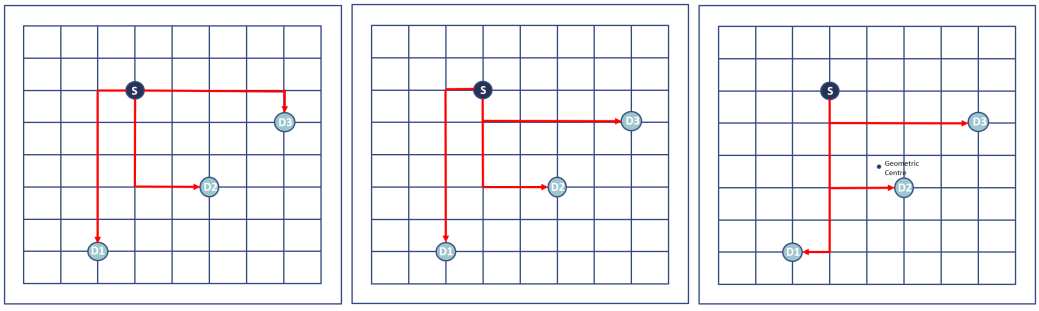
A multi-terminal net has to be decomposed into source and destination pairs in order to run the shortest path algorithm. There are different ways of doing so. Global routers aim to minimize



(a) RSMT (b) RMST (c) Connections

Figure 5.1: Decomposition of a Multi-terminal net

wirelength and therefore rely on finding rectilinear Steiner minimum tree. This includes introducing Steiner points between the terminals of the net. The connections are then broken into pairs of terminals and Steiner points. Finding the RSMT to minimize wirelength is an NP-Hard problem, but global routers[18, 19], rely on heuristic techniques to generate them. As an alternative, the rectilinear minimum spanning tree is used, which can be generated with polynomial time algorithms. Reducing the total wirelength, may not necessarily reduce the maximum delay in FPGAs or the given CGRA due to the presence of longer wires. Therefore, FPGA routers[13, 14] decompose nets into the connections between the source and each of the destination terminals. While this approach allows the algorithm to find minimum delay paths, there is no emphasis on sharing of resources. By sharing resources if possible, the amount of congestion that has to be resolved might reduce as a net would use less resources. In turn, it might be easier for the router to arrive at a routing solution. There is also a possibility of a decrease in the dynamic energy spent when resources are shared by the connections in a net although this part is not explored in the thesis.



(a) No Sharing of resources (b) Partial Sharing of Resources (c) Full Sharing of Resources

Figure 5.2: Routing of a multi-terminal net

Introducing an incentive to share resources among the connections in a net with a share cost and bias cost like in [15] and [16] would help in obtaining routing solutions like in Figure 5.2b and

5.2c wherever possible. The cost to use a resource in equation 5.9 is modified as:

$$f(e) = f_{prev} + \alpha_{(i,j)} \times d_v + (1 - \alpha_{(i,j)}) \times w_e + \frac{1}{1 + share(e)} \tag{5.13}$$

The share cost, $share(e)$ is initialized to zero for all resources $e \in E$. The source and destination connections of a multi-terminal net are routed one after the other. Each time a connection is routed, the share cost is incremented for the resources used. When the next connection of the net is routed, it will find that a resource used by a previous connection has lower cost and prefer that resource if all other costs are the same. After all the connections of a net are routed, the share cost is reset for all the resources.

For this kind of share cost to work along with the other costs in the cost function, the order in which the connections of a net are routed is important. In the example given in Figure 5.2, if connection (S1,D1) is routed first, the share cost of all resources would be zero. Therefore, the router might end up choosing a path as shown in Figure 5.2b. The connections (S1,D2) and (S1,D3) will not be able to use the resources used by (S1,D1) because the cost to their destination, d_v would overpower the share cost.

In order to solve this problem, a bias cost is introduced towards the geometric centre of the net. The bias cost is calculated as the distance from the geometric centre of a net. This would allow the connections routed earlier to favor routing paths towards the centre. When routing (S1,D1) first, the router would choose the path shown in Figure 5.2c, provided all other costs are the same.

$$f(e) = f_{prev} + \alpha_{(i,j)} \times d_v + (1 - \alpha_{(i,j)}) \times w_e + s_{fac} \times (\frac{1}{1 + share(e)} + bias) \tag{5.14}$$

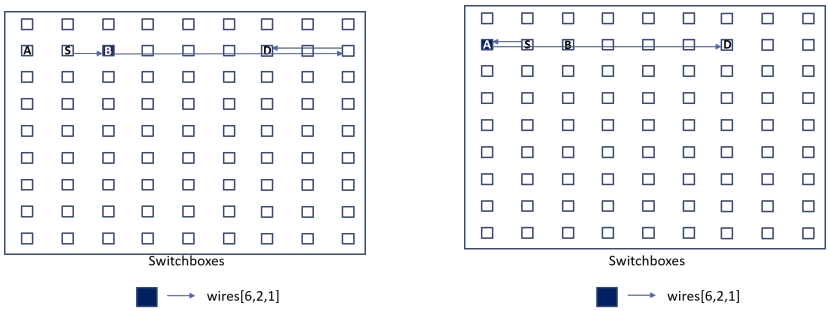
Since the primary objective of the router is to resolve congestion and route through minimum delay paths, the share and bias cost should not overpower these goals. Therefore, a constant factor s_{fac} was added to the share and bias cost in 5.14 and the value of s_{fac} was set such that $s_{fac} << 1$.

5.2.3 Router Lookahead

The addition of longer wires to the switchbox network was discussed in Chapter 4. The routing algorithm should be able to leverage the longer wires wherever possible. The heuristic discussed in Section 5.2.1 was to use the distance from the destination vertex while considering each vertex in the RRG to form a path from source to destination. With this heuristic it is possible to improve the use of longer wires in the direction of the destination as shown in Figure 5.3a. The distance from destination heuristic causes the source S to find a path to B from which it gets access to a longer length 6 wire. Since the length 6 wire reaches closer to the destination D than a length 1 wire from B, the routing algorithm would make use of the length 6 wire.

This heuristic works well in making use of longer wires between the source and destination but there is also the possibility where a longer wire is accessible at a small distance away from the destination as seen in Figure 5.3b. If the distance from destination heuristic is used, the route of S to D would consist of 5 hops using length 1 wires. If the length 6 wire at A is used, the hop count can be reduced to 2. Therefore, it is necessary for the router to be aware of the network topology and the presence of long wires.

In Pathfinder, the possibility of using a lookup table for the A-star heuristic is mentioned. In Independence[28], such a lookup table is used for an FPGA with a hierarchical interconnect. The lokup table is created by running Dijkstra's algorithm from each wiring resource to each input logic pin in the FPGA, and the delay and cost values are stored. In order to reduce the memory footprint, wiring resources were clubbed together using a k-means clustering algorithm and assigned to a logic unit. Dijkstra's algorithm is run from each cluster and values are only stored for each cluster instead of each wiring resource. A similar lookup table would make the routing algorithm aware of the network topology.



(a) Towards Destination (b) Away from Destination

Figure 5.3: Use of longer wires

CGRA Lookup Table

For the CGRA router, a simplified lookup table was constructed. The delay in terms of hop count was obtained for each switchbox(vertex in the RRG) for traversing a range of distances by running Dijkstra's algorithm for each resource. These values were stored in a map and can be looked by referencing the distance to travel in the x and y direction. In the cost function given in Equation 5.14, the value of d_v is replaced by the value from the lookup table.
$Map[swb][dx][dy] = Hopcount$

For example, in Figure 5.3b, to route S and D, in the first step A and B are considered. The distance of A and B from the D is used to lookup the map. The map contains the amount of hops required to travel that distance. It is seen that using A would result in a lower hop count and therefore, the routing algorithm chooses to route through A.

$Map[A][6][0] = 1$
$Map[B][4][0] = 4$

The distribution of wires discussed in Chapter 4 is done for each 9x9 block, so the lookup table is only created for switchboxes in a 9x9 block. When a switchbox is considered while routing a source and destination pair, the value corresponding to the position of the switchbox in its 9x9 block is used to look up the table.

$Map[swb\%9][dx][dy] = Hopcount$

The range of distances stored in the lookup table can be limited to reduce the memory footprint. If the dx and dy distances to be referenced are not available in the lookup table, the Manhattan distance(dx+dy) is used instead.

5.2.4 Bounding Box

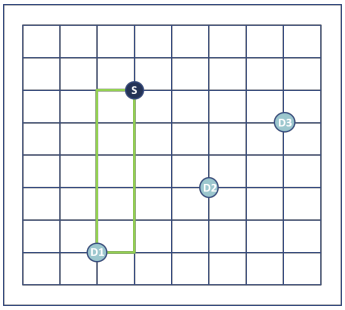


Figure 5.4: Bounding Box

While the A-star heuristic tries to reduce the search space of the shortest path algorithm, there is no hard bound on the search space. In the CGRA router, a bounding box was used to restrict

the search space of routing a connection as shown in Figure 5.4. Since the router utilizes longer wires by moving away from the destination, the bounding box was increased by l to allow it to do so. l was set equal to the length of the second longest wire, so that longest wire can always be used to move back towards the destination.

In order to avoid congestion, non critical nets should be allowed to make detours. Therefore, for a connection (S,D1) a slack value was calculated as:

$Slack = 1 - \frac{D_{(S,D1)}}{D_{max}}$

where $D_{(S,D1)}$ is the hop count from S to D1 and D_{max} is the maximum hop count. This slack value is used to expand the bounding box from all four sides.

The total increase in bounding box is given by Δ and depicted in Figure 5.4, where

$\Delta = l + \frac{Slack}{4}$

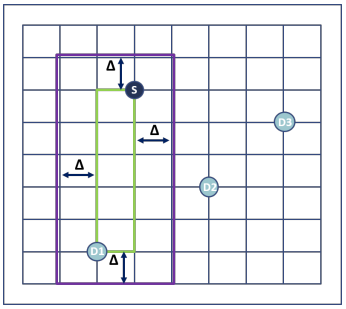


Figure 5.5: Increased Bounding Box

Using a strict bounding box for a number of iterations forces the router to negotiate among the resources present in the bounding box and find a legal routing path within it. If after a number of iterations, the router does not come up with a legal routing solution, the bounding box is slowly expanded further, to search through a larger space.

5.2.5 Reduced Connectivity

In Section 4.2.1, reduced length 6 connectivity was introduced for the switchboxes. In order to capture the decreased connectivity, a simple modification was made in the router to not use consecutive length 6 wires in the same direction and to not use consecutive length 6 wires in any direction while routing.

5.2.6 Improved Routing Result

While the router might find a legal routing solution in one of the early iterations, it might be beneficial to run the routing algorithm for a fixed number of iterations to store the best routing results out of all the legal routing solutions. The primary objective function is to reduce the maximum hop count. Therefore, the routing solution with the least maximum hop count was stored after running the routing algorithm for a user defined number of iterations.

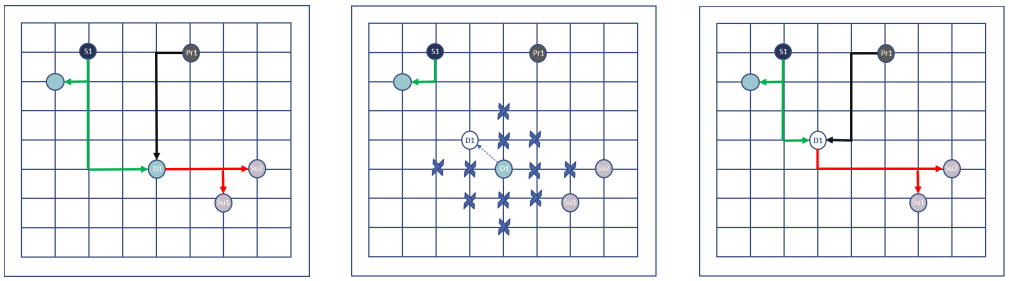
A peephole placement optimization technique is introduced in Chapter 6 to further improve the results by optimizing the placement of terminals with maximum hop count. Keeping this in mind, the number of nets with maximum hop count had to be reduced. The router was made to store the routing results that not only have the least maximum hop count but also have the least number of nets with maximum hop count.

The routing algorithm is evaluated with a subset of the topologies explored in Section 4.1 and the results are given in Chapter 7.

Chapter 6

Placement Optimization

Placement and routing in the physical design process are usually done independently although they are dependent on each other due to the complexity of these steps. The same approach was followed for the weather stencil kernels, the placement of the operations was given and a routing algorithm was designed to route the connections. Based on the results of the routing algorithm, it is possible to design a feedback to the placement algorithm. Then placement and routing will have to be run all over again and this will increase the amount of time taken to finish the process significantly. Instead, a peephole placement optimization approach was used, where a polynomial time algorithm was run to improve the placement of operations post routing in order to reduce the maximum hop count if possible. Since from the given placement in Section 2.4.1, it is seen that there are many unoccupied units, it is possible to conduct a limited search to find a more suitable position for the mapped operations.



(a) Original Routing Subgraphs (b) Breadth First Search (c) Re-routing

Figure 6.1: Post routing placement optimization

A set M was considered where each $m \in M$ is a 2-tuple of the source and destination terminals of connections with maximum hop count. The hop count of these connections may be reduced by modifying the position of either the source or the destination. In order to reduce the time complexity of the algorithm the placement optimization is run only if the number of elements in M is less than a user defined threshold.

Algorithm 2 finds a new position and is used on each $m \in M$. A new position is found for a terminal in m by doing a breadth first search in the routing resource graph from the terminal until an unoccupied unit which is of the same type(SRAM or function unit) as itself is found. The existing routing subgraphs to and from that terminal are removed. The predecessors and successors of the terminal are re-routed. If valid routes are found and they have a hop count less than the previous maximum, the new position and routing subgraphs for the terminal are accepted and the function returns. If the routes are not valid or the hop count does not decrease, the search for a new position continues until a user defined depth is reached. If the algorithm is still unsuccessful, the process is repeated for the other terminal in m . If all the elements in M are

successful in finding a new position, the overall maximum hop count reduces. Then the placement optimization is run again for a new set M.

An example is given in Figure 6.1. A net with a connection (S1,D1) has the maximum hop count of 6 in Figure 6.1a. The existing routing paths from the S1 to D1, Pr1 to D1 and from D1 to Su1 and Su2 are ripped up. The position of D1 is modified as shown, using breadth first search for an empty location which is of the same type in Figure 6.1b. The routing for the ripped connections is done using a shortest path algorithm with obstacle avoidance with the new location of D1 in Figure 6.1c. This placement optimization is only carried out if the number of elements in M is less than a user designed threshold.

Algorithm 2 Algorithm to find a new position for a terminal

```
RRG : Routing Resource Graph
(s, d) : Source and destination of connection with maximum hop count
Maximum depth: Depth of search for new location
function FindNewPosition(RRG,(s,d), Maximum Depth)
    for t in (s,d) do
        while  depth != Maximum depth do
            search status,(x,y) = BreadthFirstSearch(RRG,(t.x,t.y), t.type)
            depth+= 1
            if search status is unsuccessful then
                continue
            for pr in t.predecessors do
                RemoveRoutingSubgraph(p,t)
                route[p] = ShortestPathRouting(RRG,(p.x,p.y),(x,y))
                if len(route)>= max then
                    routing is unsuccessful
                    RestoreRoutingSubgraph(p,t)
                    break
            for su in t.successors do
                RemoveRoutingSubgraph(t,su)
                route[su] = ShortestPathRouting(RRG,(x,y),(su.x,su.y))
                if len(route)>= max then
                    routing is unsuccessful
                    RestoreRoutingSubgraph(p,t)
                    break
            if routing is successful then
                (t.x,t.y) = (x,y)
                UpdateRoutingSubgraph(route) return New Position Found
    return New Position Not Found
```

Chapter 7

Results and Evaluation

In this chapter, the results of the evaluation of the routing algorithm, the peephole placement optimization technique and interconnect topologies are discussed. The function unit and switchboxes required were synthesized and these values were used to compare the different topologies for delay, power and area.

7.1 Evaluation of Routing Algorithm

Algorithm 1 in Chapter 4 generated topologies with long wires of length 6 at every n_6^{th} switchbox and long wires of length 2 at every n_2^{th} switchbox. The capacity of all the wires in the generated topologies was set to one. In Chapter 5, an algorithm based on Pathfinder[12] was designed to efficiently use the switchbox network topology. In order to evaluate the effect of the modifications made to the original pathfinder algorithm, a subset of the topologies generated, where $n_6 = n_2$, was used along with a baseline topology with only length 1 wires. A label is used to represent each of the topologies: $t : n_6\_n_2$ and the baseline topology is represented as $t : 0$. The capacity of the wires in the baseline topology was set to two.

Routing Iterations

The naive pathfinder algorithm and the CGRA router were run until a legal routing solution was found. The purpose of this was to see how long it takes for the router to resolve all congestion and arrive at a solution. In the results in Table 7.1 it is seen that the CGRA router takes fewer iterations to arrive at a solution and the time taken by the CGRA router was also less for all the solutions. While usually arriving at solutions can be made faster by using a greedy heuristic, this could result in a compromise in the quality of results, as the solution found may be far from optimal. That is not the case here, as the maximum hop count results are much better with the CGRA router in spite of its speed. This can be attributed to the use of the lookahead, which results in better use of the topology and share cost which reduces the number of resources used per net by improving sharing among the connections in a net.

Topology	Horizontal Diffusion						Vertical Advection					
	Naïve Pathfinder			CGRA Router			Naïve Pathfinder			CGRA Router		
	Iterations	Time(s)	Maximum Hop Count	Iterations	Time(s)	Maximum Hop Count	Iterations	Time(s)	Maximum Hop Count	Iterations	Time(s)	Maximum Hop Count
t:0	18	10.63	14	6	2.73	14	7	3.06	12	3	1.22	11
t:1_1	8	3.75	4	6	1.31	4	5	3.44	4	4	1.14	4
t:2_2	8	4.31	7	6	2.08	5	5	4.20	5	5	1.53	5
t:3_3	23	14.97	8	8	2.98	7	12	10.00	7	8	2.27	6
t:4_4	29	20.36	9	9	3.02	7	11	10.20	9	7	1.94	6
t:5_5	29	37.31	10	13	4.98	7	19	12.33	8	10	2.80	6
t:6_6	20	16.92	9	19	8.13	8	18	13.73	9	10	3.03	9
t:7_7	45	40.98	11	30	15.94	10	15	13.38	9	12	3.45	7

Table 7.1: Comparison of Routing Iterations

Sharing of Resources

In the CGRA routing algorithm, a share and bias cost was added to the cost function in order to improve the sharing of resources between multi-terminal nets. In Figure 7.1, the improvement in sharing of resources by multi-terminal nets in the results of the CGRA router compared to a naive pathfinder algorithm is shown.

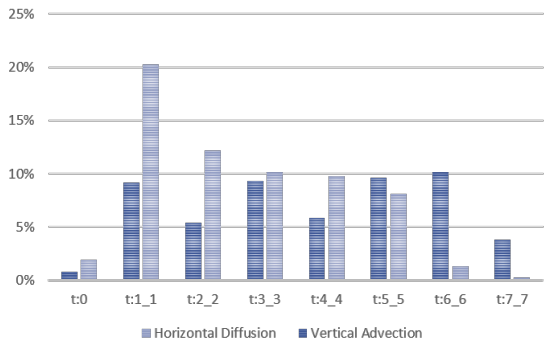
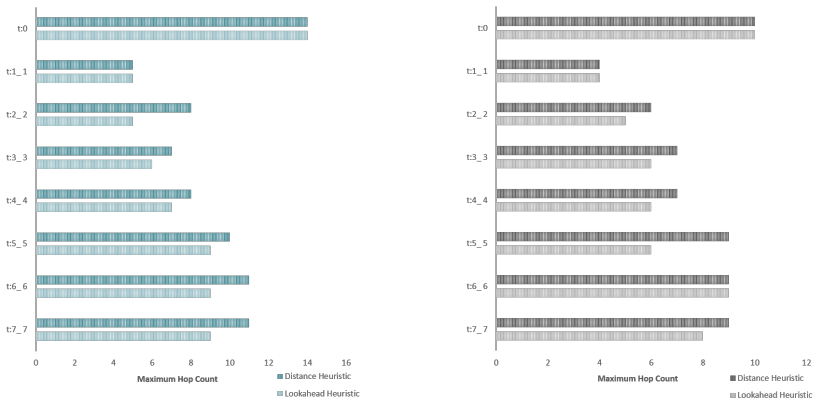


Figure 7.1: Increase in shared resources for multi-terminal nets

It is seen that there is always an increase in sharing. The weight given to sharing is very less compared to reducing hop count and resolving congestion. Therefore, in topologies with a lot of resources like $t : 1_1$ and $t : 2_2$ the increase in sharing for Horizontal Diffusion is high because of less congestion and abundant longer wires to reduce hop count. As the resources decrease, the amount of sharing decreases as resolving congestion takes higher priority. For Vertical Advection, the number of nets to be routed is fewer than Horizontal Diffusion. Resolving congestion is not as much of an issue, so the increase in sharing of resources is consistent for all topologies with long wires. In the baseline topology $t : 0$, even though there a lot of wires, due to the absence of longer wires the cost of routing through the minimum distance path takes over and the sharing of resources is less.

Maximum Hop Count



(a) Horizontal Diffusion (b) Vertical Advection

Figure 7.2: Effect of Router Lookahead on maximum hop count

The purpose of adding long wires to the network was to utilize them and reduce the maximum hop count, resulting in a reduction in overall delay. To enable the use of long wires where necessary,

a CGRA lookup table was created. The pathfinder router with an A star heuristic using distance from destination as discussed in 5.2.1 and the CGRA router with the lookahead were run for 100 iterations and the best routing result with the least minimum hop count was taken. Figure 7.2a and Figure 7.2b, show the difference in maximum hop count obtained.

The $t : 0$ and $t : 1\_1$ topologies have a large number of resources, therefore even with the distance heuristic they are able to achieve the lower bound on maximum hop count. There is a decrease in maximum hop count for both the kernels on most of the other topologies, where longer wire resources may not be available in the direction of the destination to route connections.

7.2 Peephole Placement Optimization

In Chapter 6, a peephole placement optimization was discussed where if the number of nets with a maximum hop count was less than a threshold, the connections of the nets with maximum hop were re-routed by modifying the position of the source or destination terminal. The threshold for the number of connections whose placement would be optimized was set to 15 and the depth of the search for a new position was set to 5. The results for this are given in Table 7.2.

Topology	Horizontal Diffusion				Vertical Advection			
	Before Placement Optimization		After Placement Optimization		Before Placement Optimization		After Placement Optimization	
	Maximum Hop Count	Number of nets with Maximum Hop Count	Maximum Hop Count	Number of nets with Maximum Hop Count	Maximum Hop Count	Number of nets with Maximum Hop Count	Maximum Hop Count	Number of nets with Maximum Hop Count
t:0	14	1	11	15	10	2	10	1
t:1_1	4	44	4	44	4	16	4	16
t:2_2	5	11	4	99	5	5	4	63
t:3_3	7	1	5	44	5	17	5	17
t:4_4	6	13	6	3	6	3	5	24
t:5_5	7	6	7	3	6	9	5	48
t:6_6	8	2	8	1	6	14	5	55
t:7_7	9	1	8	20	7	2	6	20

Table 7.2: Results of Peephole Placement Optimization

The placement optimization is run only when the number of nets with maximum hop count is less than 15. In the $t : 1\_1$ topology, the lower bound on the maximum hop count was obtained without placement optimization. In most cases where placement optimization is run, the maximum hop count reduces. The exceptions are for topologies $t : 4\_4$, $t : 5\_5$ and $t : 6\_6$ for horizontal diffusion, where it was not successful in reducing the hop count. In the case of $t : 3\_3$, placement optimization is called twice and the hop count reduces by two for horizontal diffusion.

7.3 Design space exploration of interconnect topologies

In Chapter 4, Algorithm 1 was given to generate possible interconnect topologies to explore. The routing algorithm was run for the weather stencil kernels on each of these topologies for a number of iterations and the best routing result in terms of maximum hop count and number of nets with maximum hop count was obtained. Peephole placement optimization was run to reduce the hop count if possible. Out of the 44 topologies generated, routing was successful on 39 of them for Horizontal Diffusion and Vertical Advection. Routing failed on five of the topologies with the least number of resources for Horizontal Diffusion and on two of the topologies with the least resources for Vertical Advection.

The maximum hop count of both the weather stencils for a topology was taken to plot a pareto curve against the number of routing resources in that topology in Figure 7.3.

The trend is that with decreasing number of resources, the maximum hop count increases. This is as expected as the number of detours in routing a net increases as the number of available routing resources decreases. There are a few exceptions to this. In a few cases, like $t : 8\_1$, $t : 7\_1$ and $t : 5\_1$ the maximum hop count is higher than a topology with fewer resources like $t : 2\_2$.

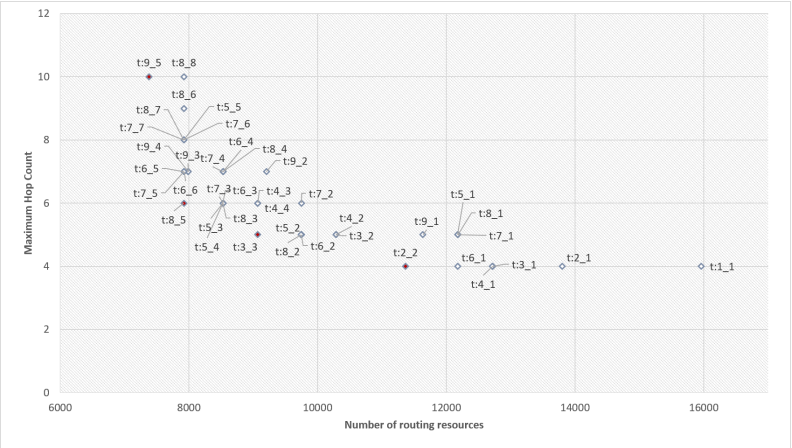


Figure 7.3: Maximum Hop count vs number of routing resources for all topologies

This is because of the sparse distribution of the long length 6 wires in the former topologies. The probability of a critical net not having access to a longer wire is higher in such cases. The same is observed for $t : 7\_2$ and $t : 9\_3$ compared to $t : 3\_3$ and $t : 8\_5$. This is also observed if the length 2 wires are too far apart. The topologies $t : 8\_5$, $t : 8\_5$, $t : 8\_5$ and $t : 8\_5$ have the same number of resources but the hop count for $t : 8\_5$ is less because of smaller distance between length 2 wires.

7.3.1 Results of Routing with Reduced Switchbox connectivity

Topology	Connectivity	Label	Maximum Hop Count			
			Horizontal Diffusion		Vertical Advection	
			Before placement optimization	After placement optimization	Before placement optimization	After placement optimization
t:0	Full connectivity	t:0_fc	14	11	10	10
t:1_1	Full connectivity	t:1_1_fc	4	4	4	4
	Reduced length 6 connectivity 1	t:1_1_rc1	4	4	4	4
	Reduced length 6 connectivity 2	t:1_1_rc2	5	4	4	4
t:2_2	Full connectivity	t:2_2_fc	5	4	5	4
	Reduced length 6 connectivity 1	t:2_2_rc1	5	4	5	4
	Reduced length 6 connectivity 2	t:2_2_rc2	5	5	5	4
t:3_3	Full conncectivity	t:3_3_fc	6	5	5	5
	Reduced length 6 connectivity 1	t:3_3_rc1	6	5	5	5
	Reduced length 6 connectivity 2	t:3_3_rc2	6	5	6	5
t:8_5	Full connectivity	t:8_5_fc	7	6	6	6
	Reduced length 6 connectivity 1	t:8_5_rc1	7	6	6	6
	Reduced length 6 connectivity 2	t:8_5_rc2	7	6	6	6
t:9_5	Full connectivity	t:9_5_fc	10	6	8	7
	Reduced length 6 connectivity 1	t:9_5_rc1	10	10	8	7
	Reduced length 6 connectivity 2	t:9_5_rc2	10	10	8	7

Table 7.3: Maximum Hop count with reduced connectivity

The reduced connectivity in the switchbox discussed in Section 4.2.1 was implemented by making the changes discussed in Section 5.2.5 in the routing algorithm and running the algorithm for the kernels on the topologies on the pareto frontier in Figure 7.3. The reduced connectivity for length 6 wires in the same direction does not affect maximum hop count in any of the cases. It is the same as the full connectivity results. This was expected because of the number of wires longer than 9 is few. Reducing connectivity in all directions did increase the maximum hopcount in some cases but it was reduced by placement optimization.

7.3.2 Synthesis of Switchboxes

Based on the topologies generated, the different types of switchboxes required was consolidated to the five types given in Figure 7.4. These switchboxes were synthesized for the different types connectivity discussed in Section 4.2.1 using a 28nm technology for a typical corner with 0.9V and 25°C at a frequency of 100MHz. A load capacitance of 0.6pF, 0.2pF and 0.1pF was used for length 6, 2 and 1 wires respectively. Along with the switchbox, a naive implementation of the function unit(ALU/Multiplier) was synthesized. Since the synthesis was done without simulation, the dynamic power values generated by the tool may be inflated but the relative values of the different switchboxes and the function unit can be used to compare them.

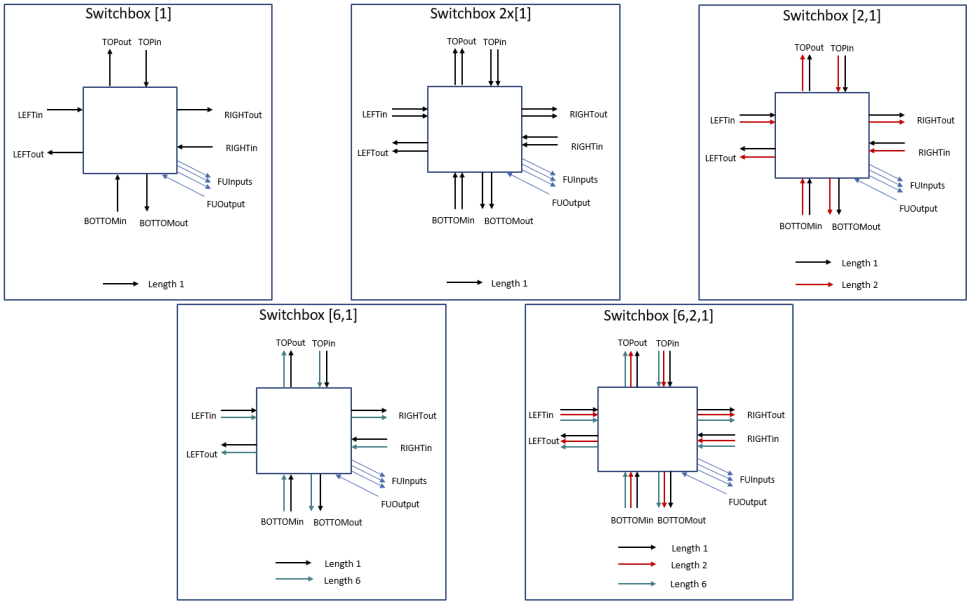


Figure 7.4: Types of Switchboxes

The synthesis results are given in Table 7.4. When the number of wires in a switchbox reduces like in Switchbox [6,2,1] and Switchbox [6,1], there is a significant drop in leakage power, more than 49%. The delay values also reduce by 10-20ps. The decrease in wire lengths in the switchboxes reduces the delay by a few picoseconds, as the load capacitance value used decreases. With the reduction in connectivity there is not much change in delay. For switchbox [6,1] the delay increases by a 1ps with reduced connectivity. This has to do with components used by the synthesis tool. The decrease in power and area is also quite less, around 5%. With a custom design of switchbox, it would be possible to see more significant improvement in the power, delay and area values for the switchboxes with reduced connectivity.

	Connectivity	Delay(ps)	Leakage Power(µW)	Dynamic Power(µW)	Gates	Area(µm2)
Function Unit	-	1330	1.52	917.46	2320	5367
Switchbox [6,2,1]	Full Connectivity	185	2.17	119.68	4956	7719
	Reduced Length 6 Connectivity 1	182	1.98	115.39	4453	7148
	Reduced Length 6 Connectivity 2	182	1.81	106.77	4210	6696
Switchbox [6,1]	Full Connectivity	177	0.77	70.44	2052	3464
	Reduced Length 6 Connectivity 1	178	0.68	69.78	2094	3238
	Reduced Length 6 Connectivity 2	179	0.61	63.30	1532	2742
Switchbox [2,1]	Full Connectivity	173	0.76	70.09	2044	3455
Switchbox 2x [1]	Full connectivity	172	0.76	69.85	2034	3451
Switchbox [1]	Full Connectivity	152	0.25	37.84	539	1182

Table 7.4: Switchbox Synthesis Results

7.4 Evaluation of power, delay and area values

The delay, power and energy values were modeled using the synthesis results from Section 7.3.2.

Topology/ Connectivity	Maximum Hop Count	Maximum Delay(ns)	Total Power (mW)	Total Area (mm2)
t:0_fc	11	2.580	101.96	4.98
t: 1_1_fc	4	0.925	175.95	11.15
t: 1_1_rc1	4	0.910	169.49	10.32
t: 1_1_rc2	4	0.910	156.79	9.67
t:2_2_fc	4	0.925	122.26	6.96
t:2_2_rc1	**4**	**0.910**	**118.67**	**6.50**
t:2_2_rc2	5	1.092	111.60	6.13
t:3_3_fc	5	0.978	95.37	4.86
t:3_3_rc1	5	0.972	93.22	4.58
t:3_3_rc2	**5**	**0.972**	**88.98**	**4.36**
t:8_5_fc	6	1.122	79.06	3.49
t:8_5_rc1	6	1.120	75.27	3.29
t:8_5_rc2	**6**	**1.121**	**72.79**	**3.14**
t:9_5_fc	6	1.911	73.73	3.12
t:9_5_rc1	10	1.905	70.05	2.96
t:9_5_rc2	**10**	**1.905**	**68.64**	**2.89**

Table 7.5: Total Power vs Maximum Hop Count

From Figure 7.5 and Table 7.5, it was seen that compared to the baseline topology $t:0$ with only length 1 wires, topologies with longer wires like $t:3\_3$, $t:8\_5$ and $t:9\_5$ have not only lower delay, but also lower power and area values. $t:1\_1\_rc1$ and $t:1\_1\_rc2$ have the least amount of delay, as they have longer wires in every switchbox. But this leads to a 40% increase in total power and 48% increase in area compared to the baseline $t:0_0$. The topology $t:9\_5\_rc1$ has the least power and area, due to almost 80% its switchboxes being of type [1] in 7.4, but the delay is high due to a high maximum hop count. $t:2\_2\_rc2$ has the lowest delay of 0.910ns but has 14% higher power and 25% more area than the baseline model. $t:3\_3\_rc2$ has 50% lower delay, 12% lower power and 12% decrease in area. Therefore, it is an optimal design in terms of delay, power and area.

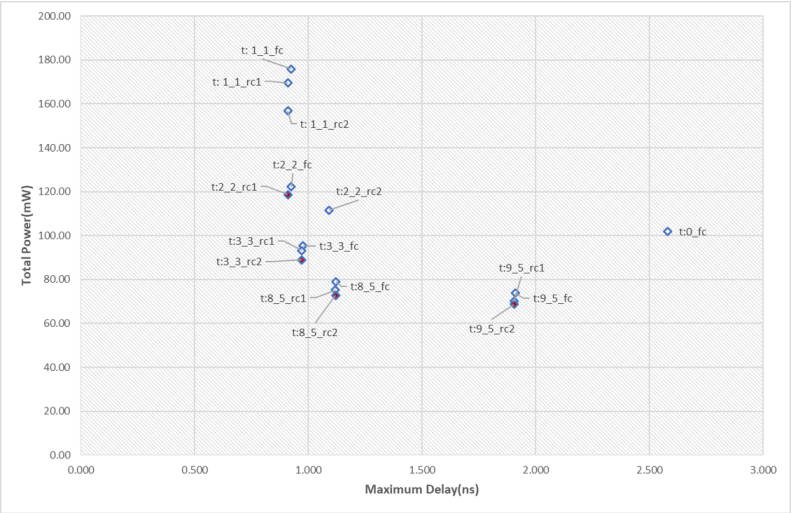


Figure 7.5: Total Power vs Maximum Hop Count

7.5 Delay, Power and Area of the COSMO CGRA

The delay of the entire COSMO CGRA was modeled using the synthesized values of the function unit and the switchboxes. The delay, power and area values for the SRAM units were obtained from a commercial 28nm SRAM. The results were compared for a COSMO CGRA with the baseline topology $t : 0$ and a COSMO CGRA with the optimal modified topology $t : 3\_3$ and is given in Table 7.6. The combinatorial path formed by the switchboxes starts at the output of

Architecture	Delay(ns)	Total Power(mW)	Total Area(mm2)
Baseline Interconnect	3.91	1263.76	12.09
Modified Interconnect	2.30	1250.78	11.47

Table 7.6: Comparison of delay, power, and area of the COSMO CGRA

the source function unit and ends at the output of the destination function unit. Therfore, the maximum delay is computed as the sum of the delay in a function and the maximum delay in the interconnect. The maximum delay reduces by 40% in the modified topology compared to the baseline. The decrease in power and area is around 1% and 4% respectively but this is because the power of the function unit and the area of the SRAM units dominate over the interconnect.

Chapter 8

Conclusions

We designed a novel re-configurable interconnect topology for a CGRA-based architecture for a real-world weather prediction application. An improvement to the existing Blocks switchbox topology was made by adding long wires, inspired by the interconnect in FPGAs. The lengths of wires required and the distribution of wires in the design in order to route weather stencil kernels mapped to the CGRA, was explored. The combination of wires of length 6, 2 and 1 was chosen as these wire in a combinations of maximum 4 are able to cover all the distances required by the weather stencil kernels.

A routing algorithm with the objective of reducing the delay of the longest connection was developed. In order to do so, the router was made architecture aware by using a custom lookup table. It was shown that with the use of the custom lookup table there was an improvement in the maximum hop count for the weather stencil kernels. Other techniques were used to improve other aspects of the routing algorithm, like use of a bounding box with slack and storing the best results in the context of the next step, that is placement optimization. Sharing of resources for the connections in a multi-terminal nets was improved through a modification in the cost function used by the router.

A peephole placement optimization was used to further improve the results of the router by reducing maximum hop count. The run time of the placement optimization was bound by restricting it to cases where the number of nets requiring placement optimization was less than a threshold value. Also, the search space for a new position was restricted to a depth. Although the threshold and depth were set to very low values, the placement optimization succeeded in reducing the hop count by 1 in many of the cases.

Switchbox connectivity was optimized to reduce the delay, power and area of the switchbox. The optimizations were made based on the nature of the placement of operations in the CGRA so as to not affect routability or maximum hop count. These optimizations were taken into account in the router and it was seen that routability was not affected and there was minimum impact on hop count. .

The routing algorithm was used to judge switchbox topologies with different distributions of wires. Since the number of topologies to explore was high, the pareto optimal topologies were chosen to evaluate with delay, power and area values of switchboxes. While individual switchboxes were synthesized, the delay, power and area values were modeled for the 38x38 topology using the synthesized values. The gain in delay, power and area values was not significant for switchboxes with reduced connectivity, but this can be improved with a custom design. It was seen that compared to the baseline switchbox topology with only length 1 wires, a topology with longer wires was able to obtain 50% lower delay, 12% lower power and 12% lower area. While comparing the entire COSMO CGRA with the interconnect topologies a similar decrease in delay is observed but the decrease in power and area is less than 5%. This is due to the the power consumed by the function units and the area occupied by SRAM units which have to be optimized in the future.

8.1 Future Work

The future work that can be explored is discussed below.

- Registers in Switchboxes
  The long combinatorial path to connect switchboxes was reduced as much as possible in this thesis. By adding registers in the switchboxes and a few more stages of pipelining, it is possible to break up the combinatorial paths, while maintaining an initiation interval of one and thereby reducing the overall delay. This might require more stages to be added to the data flow graphs of the weather stencil kernels and the mapping of these data flow graphs to the CGRA will have to redone for an initiation interval of one.

- Function Unit
  A naive implementation of the function unit was used to determine its delay. This was needed to compute the maximum path delay which starts at the output of the source function unit and ends at the output of the destination function unit. This function unit has to be implemented along with all the required operations and the total delay has to be calculated again in order to estimate the frequency at which the application can be run. The power consumed by the function unit is also very high about 9× more than that of a switchbox. Therefore an energy efficient implement of the function unit is necessary.

- Custom Design of switchboxes
  For this thesis, there was not enough time to design a custom switchbox. The components available in a 28nm library were used with the synthesis tools. The use of transmission gates to implement the multiplexers used in the switchboxes could lead to less delay and area but the signal level degradation that might occur has to be taken care of while designing them. Also, with custom designs the reduced connectivity in switchboxes will have more of an impact on improving performance.

Bibliography

[1] G. Doms and M. Baldauf. A description of the nonhydrostatic regional cosmo-model. 2018. 1

[2] Tobias Gysi, Tobias Grosser, and Torsten Hoefler. Modesto: Data-centric analytic optimization of complex stencil programs on heterogeneous architectures. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, page 177–186, New York, NY, USA, 2015. Association for Computing Machinery. 1

[3] M. Wijtvliet, L. Waeijen, and H. Corporaal. Coarse grained reconfigurable architectures in the past 25 years: Overview and classification. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 235–244, 2016. 1, 13

[4] Ronald P Luijten. Cgra accelerator for weather/climate dynamics simulation. P201909001US01. 1, 4

[5] M. Wijtvliet, J. Huisken, L. Waeijen, and H. Corporaal. Blocks: Redesigning coarse grained reconfigurable architectures for energy efficiency. In *29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 17–23, 2019. 2, 13

[6] M. J. P. Walker and J. H. Anderson. Generic connectivity-based cgra mapping via integer linear programming. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 65–73, 2019. 11

[7] S. An, M. Zhang, X. Ye, D. Wang, H. Zhang, D. Fan, and Z. Tang. C-map: Improving the effectiveness of mapping method for cgra by reducing noc congestion. In *2019 IEEE 21st International Conference on High Performance Computing and Communications*, pages 321–328, 2019. 11

[8] L. Zhou, J. Zhang, and H. Liu. Ant colony algorithm for steiner tree problem in cgra mapping. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 198–202, 2017. 11

[9] L. Chen and T. Mitra. Graph minor approach for application mapping on cgras. In *2012 International Conference on Field-Programmable Technology*, pages 285–292, 2012. 11

[10] M. Hamzeh, A. Shrivastava, and S. Vrudhula. Epimap: Using epimorphism to map applications on cgras. In *DAC Design Automation Conference 2012*, pages 1280–1287, 2012. 11

[11] M. Karunaratne, A. K. Mohite, T. Mitra, and L. Peh. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017. 11

[12] L. McMurchie and C. Ebeling. Pathfinder: A negotiation-based performance-driven router for fpgas. In *Third International ACM Symposium on Field-Programmable Gate Arrays*, pages 111–117, 1995. 12, 20, 21, 29

[13] Vaughn Betz and Jonathan Rose. Vpr: A new packing, placement and routing tool for fpga research. In *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, FPL '97, page 213–222, Berlin, Heidelberg, 1997. Springer-Verlag. 12, 22

[14] K. E. Murray, S. Zhong, and V. Betz. Air: A fast but lazy timing-driven fpga router. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 338–344, 2020. 12, 22

[15] E. Vansteenkiste, K. Bruneel, and D. Stroobandt. A connection-based router for fpgas. In *2013 International Conference on Field-Programmable Technology (FPT)*, pages 326–329, 2013. 12, 22

[16] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt. Croute: A fast high-quality timing-driven connection-based fpga router. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 53–60, 2019. 12, 22

[17] ISPD routing contest. `http://www.ispd.cc/contests/07/contest.html`. 12

[18] M. Pan and C. Chu. Fastroute: A step to integrate global routing into placement. In *2006 IEEE/ACM International Conference on Computer Aided Design*, pages 464–471, 2006. 12, 22

[19] M. D. Moffitt. Maizerouter: Engineering an effective global router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(11):2017–2026, 2008. 12, 22

[20] K. Dai, W. Liu, and Y. Li. Nctu-gr: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-d global routing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(3):459–472, 2012. 12

[21] Muhammet Mustafa Ozdal and M. D. F. Wong. Archer: a history-driven global routing algorithm. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 488–495, 2007. 12

[22] Jonathan Greene, Vwani Roychowdhury, Sinan Kaptanoglu, and Abbas El Gamal. Segmented channel routing. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, page 567–572, New York, NY, USA, 1991. Association for Computing Machinery. 13

[23] M. Pedram, B. S. Nobandegani, and B. T. Preas. Design and analysis of segmented routing channels for row-based fpga's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(12):1470–1479, 1994. 13

[24] J. Rose and S. Brown. Flexibility of interconnection structures for field-programmable gate arrays. In *IEEE Journal of Solid-State Circuits*, volume 26, pages 277–282, 1991. 17

[25] T. G. Szymanski. Dogleg channel routing is np-complete. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(1):31–41, 1985. 20

[26] C. Y. Lee. An algorithm for path connections and its applications. In *IRE Transactions on Electronic Computers*, volume EC-10, pages 346–365, 1961. 21

[27] R. Nair. A simple yet effective technique for global wiring. 6(2):165–172, November 2006. 21

[28] A. Sharma and S. Hauck. Accelerating fpga routing using architecture-adaptive a* techniques. In *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, pages 225–232, 2005. 23