1995

Combinatorial Design and Analysis of Optimal Multiple Bus Systems for Parallel Algorithms.

Priyalal D. Kulasinghe
Louisiana State University and Agricultural & Mechanical College

COMBINATORIAL DESIGN AND ANALYSIS OF OPTIMAL MULTIPLE BUS SYSTEMS FOR PARALLEL ALGORITHMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by

Priyalal D. Kulasinghe
B.Sc., University of Peradeniya, Sri Lanka 1981
M.S., Louisiana State University, 1990
August 1995

ACKNOWLEDGEMENTS

I would like to express my thanks and gratitude to my advisor Dr. Ahmed El-Amawy for his constant support, guidance, and encouragement during this research work.

I also want to extend my thanks to Dr. Said Bettayeb who has helped me in many ways in developing my research activities. Helpful suggestions and many other supports given by the other committee members Dr. Subhash Kak, Dr. R. Vaidyanathan, Dr. J. Ramanujam, Dr. P. Bhattacharya, Dr. Gil S. Lee, and Dr. O. Hidalgo-Salvatierra are also appreciated.

Special thanks to my wife Eshani who was always there whenever I needed help throughout the course of my studies at *LSU*. Without her encouragement and sacrifice, this work would not have been completed.

TABLE OF CONTENTS

ABSTRACT

This dissertation develops a formal and systematic methodology for designing optimal, synchronous multiple bus systems (*MBS*s) realizing given (classes of) parallel algorithms. Our approach utilizes graph and group theoretic concepts to develop the necessary model and procedural tools. By partitioning the vertex set of the graphical representation *CFG* of the algorithm, we extract a set of interconnection functions that represents the interprocessor communication requirement of the algorithm. We prove that the optimal partitioning problem is *NP*-Hard. However, we show how to obtain polynomial time solutions by exploiting certain regularities present in many well-behaved parallel algorithms.

The extracted set of interconnection functions is represented by an edge colored, directed graph called interconnection function graph (*IFG*). We show that the problem of constructing an optimal *MBS* to realize an *IFG* is *NP*-Hard. We show important special cases where polynomial time solutions exist. In particular, we prove that polynomial time solutions exist when the *IFG* is vertex symmetric. This is the case of interest for the vast majority of important interconnection function sets, whether extracted from algorithms or correspond to existing interconnection networks. We show that an *IFG* is vertex symmetric if and only if it is the Cayley color graph of a finite group Γ and its generating set Δ. Using this property, we present a particular scheme to construct a symmetric *MBS* $M(\Gamma, \Delta)$ with minimum number of buses as well as minimum number of interfaces realizing a vertex symmetric *IFG*.

We demonstrate several advantages of the optimal *MBS* $M(\Gamma, \Delta)$ in terms of its symmetry, number of ports per processor, number of neighbors per processor, and the

diameter. We also investigate the fault tolerant capabilities and performance degradation of $M(\Gamma, \Delta)$ in the case of a single bus failure, single driver failure, single receiver failure, and single processor failure. Further, we address the problem of designing an optimal *MBS* realizing a class of algorithms when the number of buses and/or processors in the target *MBS* are specified. The optimality criteria are maximizing the speed and minimizing the number of interfaces.

CHAPTER 1

INTRODUCTION

Many of today's scientific and industrial applications such as image processing, weather forecasting, fluid dynamics, plasma physics simulations, robot vision, molecular biology, neural computing, and seismology require enormous amounts of computing power. For example, in molecular biology, simulation of a protein molecule with only a few thousand atoms for 1 μs time span would require 100 years on Cray 2. Due to technological limitations, these higher computational demands cannot be effectively met by a single processor system. A logical solution would be to use parallel processing. In a parallel processing system, several processors collectively solve a given problem by having each processor work on a different part of the problem simultaneously and exchanging messages over a network of communication links. Today, parallel processors are commercially available and they range from systems with few processors to systems with thousands of processors. An example from the low end is the Encore Multimax [118], in which up to 20 processors are connected by a single bus. An example from the high end is the Connection Machine [142], in which a maximum of 64 thousand processors are connected by a boolean hypercube.

A parallel processing system, or a parallel architecture, mainly consists of a set of processors, a set of memory modules, and an interconnection network that provides communication paths among processors and memory modules. Each processor simultaneously executes a subtask of the problem at hand. Whenever one processor needs information produced by another processor, they communicate via the interconnection network. Therefore, a parallel processing system executing a given problem spends time

not only on computation but also on communication. The major concern in a parallel processing system, which is not present in a single processor system, is the cost of (time spent on) the communication among processors. In many systems, the larger the number of processors in the system, the higher the communication cost would be. Running an algorithm on a massively parallel computer, more time would be spent on communication than on computation [37]. The communication among processors and memory modules is governed by the interconnection network. In fact, a major distinguishing feature of one multiprocessor system from another is its interconnection network. The focus of this dissertation is on the interconnection network of parallel processing systems. In the literature, a large number of interconnection networks have been proposed for interprocessor communication. They include hypercubes [119], ring [103], tree [40], [69], cube connected cycles [115], omega network [92], data manipulator network [47], generalized cube network [125], [128], De Bruijn network [121], single bus multiprocessor [25], and the multiple bus multiprocessor [42], [66], [109]. Each topology has its own merits, and the selection of a topology is rather application dependent. For example, mesh connected systems are better suited for applications like image processing and weather simulation, while hypercube systems are better suited for applications like *FFT* computation and bitonic sorting. For survey and comparison of the relative merits of these topologies, the reader is referred to [20], [43], [46], [127].

1.1 Classification of Interconnection Networks

Since it is a very difficult task to perform a comprehensive study on every available parallel architecture, it is necessary to divide them into several general classes. Different classifications of interconnection networks can be made by viewing them from

different perspectives. The works reported in [20], [48], [50], [59], and [72] present different kinds of taxonomy for interconnection networks. Here, we do not attempt to exhaust all possible classifications of interconnection networks. For the purpose of this dissertation, parallel architectures will be classified depending on their *topology*, *operation mode*, and *communication strategy*. These classifications are not very strict and certain overlappings may exist among different classes.

1.1.1 Topology

The topology of an interconnection network tells us how processors are connected to each other. The topology also determines the diameter, the node degree, the bisection width, and the connectivity of the interconnection network. In this dissertation we divide interconnection networks into three broad topological classes, called *direct link*, *switched*, and *bused*.

An interconnection network can be built by establishing a direct link (either unidirectional or bidirectional) between certain pairs of processors. This type of interconnection network is called a *direct link* (or a dedicated path) interconnection network. Link connections do not change over time. Therefore, they are also called *static* interconnection networks. If the source processor and the target processor are connected by a common link, data will be sent along that link. If they do not have a common link, they will communicate using some intermediate processors. Some examples of such direct link interconnection networks are the hypercube [61], cube connected cycles [115], tree [69], and the ring [103].

In the second topological class, switches are used to establish connections among processors. No two processors are connected by a direct link. Switch settings can be

dynamically changed to implement various interconnection patterns. Usually, switches are arranged in stages, and the networks are referred to as single or multistage interconnection networks. Some examples of such networks are generalized cube network [128], banyan network [55], omega network [92], benes network [15], data manipulator network [47], and *STARAN* flip network [12].

In the third topological class, processor interconnections are achieved by one or more buses and a set of interfaces. Depending on whether one bus or more than one bus are used, the multiprocessor system will be called a single bus system or a multiple bus system, respectively. In a single bus system, each processor, as well as each memory module, is connected to a common bus via interfaces. Some commercially available single bus systems are Encore Multimax [118], Sequent [111], *SPUR* [63], and Firefly [139]. In a multiple bus system, each processor, as well as, each memory module, is connected to a (not necessarily proper) subset of buses via interfaces. By connecting different processors to different buses, a rich set of interconnection networks can be formed. Some examples are the conventional multiple bus system [109], partial multiple bus system [33], hierarchical multiple bus system [147], and orthogonal multiple bus system [71].

1.1.2 Operation Mode

Most parallel processing systems reported operate in one of the two modes of parallelism called *SIMD* and *MIMD* [50]. In the *SIMD* (single instruction stream - multiple data stream) mode, all processors execute the same instruction stream issued by a central control unit. However, each processor operates on a different data set. Communication capabilities of the interconnection network of an *SIMD* architecture is determined by the set of interconnection functions associated with it. Each interconnection

function is considered as a set of source-destination processor pairs which can perform direct communications concurrently. Some examples of *SIMD* parallel architectures are *ILLIAC IV* [11], *GF*11 [14], and Connection Machine [62].

In the *MIMD* (multiple instruction stream - multiple data stream) mode of operation, each processor executes its own set of instructions and uses its own data. There is no control unit to effect global synchronization among individual processors. Communication among processors is done by handshaking. Therefore, *MIMD* mode of operation is asynchronous. Some examples of *MIMD* parallel architectures are EMPRESS [26], Cm* [137], Cosmic Cube [122], and Cedar [82].

There are certain other parallel systems that can operate both in the *SIMD* mode and/or in the *MIMD* mode. These are called *SIMD/MIMD* architectures. Usually these architectures comprise of processors connected in a hierarchical fashion. Processors functioning within one hierarchical level are in the *SIMD* mode while those in another hierarchical level are in the *MIMD* mode. Some other architectures, while executing a parallel algorithm, may operate in the *SIMD* mode for a certain amount of time and then switch to the *MIMD* mode. Some examples of *SIMD/MIMD* architectures are *SNAP*-1 [39], *PASM* [126], and *DCA* [75].

1.1.3 Interprocessor Communication Models

There are two basic communication strategies in multiprocessor systems. In the *shared memory model*, as the name implies, processors communicate via a shared global memory. A source processor needing to communicate places its output data in an area in the shared memory and the destination processor(s) read the data from that memory area. Usually, shared memory multiprocessors are tightly coupled and use a single address

space. Some examples of shared memory multiprocessors are *NYU* Ultra Computer [57], Alliant FX/80 [152], *HEP* [80], *KSR-1* [116], *BBN* Butterfly [36], and *DASH* [96]. The advantages of shared memory multiprocessors are flexibility and the ease of programming. A major disadvantage of a shared memory multiprocessor is the cost. In the context of special purpose architectures, shared memory models are not cost effective. They cannot efficiently exploit the communication structures of the applications under consideration. Thus, shared memory multiprocessors are more suitable for general purpose applications.

In the *message passing model*, each processor is provided with a local memory. The source processor sends data to the destination processor via the interconnection network. Usually, message passing multiprocessors are loosely coupled. Some examples of such machines are the Cosmic Cube [122], *CHiP* [130], *J*-Machine [38], and *iPSC/2* [9]. Since there is no requirement of concurrent access of data, implementation of memory is less expensive. The message passing model is more suitable for distributed environments. Also, the designer of a message passing (special purpose) multiprocessor can make use of the inherent communication structure of the application domain. The message passing model very well exploits the locality of reference present in many algorithms. This led many researchers to favor the message passing model over the shared memory model for interprocessor communication [93], [94], [99]. A disadvantage of the message passing paradigm is that an algorithm with a communication structure that does not match the topology of the machine will poorly run on the machine.

There are certain parallel architectures which use both the shared memory model and the message passing model for interprocessor communication [39], [51], [87]. It should be pointed out that, irrespective of the hardware model, any machine can simulate

either a message passing model or a shared memory model using software [16], [97], [98]. Good surveys on comparison of the two communication strategies can be found in [78], [93], [98], [105].

1.2 Classification of Parallel Algorithms

Parallel algorithms can be classified by the properties of algorithms such as communication pattern, data formats and structures, data size, types of operations, and control flow strategy [45], [110], [115]. For the purpose of designing an interconnection network, only the communication pattern of an algorithm faithfully reflects the fundamental characteristic of the problem solving method. Classification of parallel algorithms according to their communication structure was reported in [115]. Algorithms such as *FFT* computations, bitonic sort, and merge sort possess similar communication patterns and therefore belong to the *Ascend/Descend* class of parallel algorithms. Algorithms such as searching, summation, and min/max computation belong to the *Fan-in/Fan-out* class of parallel algorithms [144].

1.3 Bused Interconnection Networks

From the three topological classes of interconnection networks, the focus of this dissertation is on bus based interconnection networks. Even though they have not received as much attention as the other two topological classes, bus based architectures have been addressed in several occasions in the literature [8], [33], [68], [71], [109], [148]. Those systems can potentially have many obvious advantages, among which are ease of broadcasting, ease of incremental expansion, feasibility for efficient *VLSI* layout, fault tolerance, high memory bandwidth, and flexibility [33], [74], [90], [151]. Some of the limitations of bused interconnections, such as, bandwidth, stray capacitance, and reflection

waves produced at interfaces can be overcome by using the emerging optical technology [34], [58], [135]. Depending on the number of buses used for interprocessor communication, bused interconnection networks are divided into two categories.

1.3.1 Single Bus Systems

In a single bus system (also called shared bus system), communication among processors and communication between processors and memory modules is done over the common shared bus. More than one processor attempting to use the bus at the same time will result in a bus conflict. This will be resolved by an arbiter. There are several shared bus systems commercially available today. Encore Multimax [118], Sequent Symmetry [101], and *SPUR* [63] are some examples of such systems. A major reason for their popularity is the ease of implementation compared with multiprocessor systems with other types of interconnection networks. Their major disadvantage is that the number of processors is limited due to the limited bandwidth of the bus. Increasing the bandwidth by using a wider bus is not cost effective [68]. A cost effective solution is to use several buses instead of a single wide bus.

1.3.2 Multiple Bus Systems

Multiple bus systems have largely been left unexplored in the literature despite the fact that the single bus system has been the most prevalent interconnection method used for commercial releases of multiprocessor systems. With multiple buses, we can increase the number of processors in the system beyond what is possible in a single bus system, thereby potentially increasing the performance. To allow for multiple accesses to the memory simultaneously, the standard procedure is to divide the memory into several modules. In the conventional multiple bus system, each processor and each memory

module is connected to every bus [18], [42], [66], [109], [150]. This method becomes prohibitively costly when the number of processors and that of buses increase. When the number of buses is large, the number of ports per processor in a conventional multiple bus system may exceed physical limits. When the number of processors is large, the number of tappings in a bus becomes large. This creates problems such as bus loading, large stray capacitances, and wave reflections. Furthermore, with many bus connections, complex arbitration is needed.

Because of the above mentioned drawbacks of the conventional multiple bus system, several other different configurations have been suggested in the literature. These configurations reduce the number of bus connections without degrading the performance considerably. In [90], a method is introduced to reduce a substantial number of bus connections (compared to the conventional multiple bus system) while keeping the same memory bandwidth (provided that proper arbitration is present). In the partial multiple bus system, the set of buses and the set of memory modules are partitioned into several groups such that a memory module is only connected to the buses within the group, while each processor is connected to all buses [33], [89]. This is called memory oriented partial multiple bus system (*MPMB*). In another variation, called processor oriented partial multiple bus system (*PPMB*), the set of buses and the set of processors are divided into several groups such that each processor is connected to the buses within its own group only, while each memory module is connected to all the buses [74]. In the hierarchical multiple bus system, several levels of buses are used, where lowest level buses are connected to processors and highest level buses are connected to memory modules [147], [148]. In the orthogonal multiple bus system, one bus is dedicated to each processor and

buses are arranged in rows and columns [26], [71], [143]. All these multiple bus systems are aimed at general purpose applications. For special purpose multiple bus systems, communication patterns of the algorithms running on the system are usually known a priori. Therefore, the number of bus connections can be further reduced.

In the conventional multiple bus system, [18], [109], and in many of the newer versions (such as *MPMB* [33], *PPMB* [74], and hierarchical multiple bus system [148]), the mode of operation is *MIMD* and the communication is via shared memory. Since in the *MIMD* mode of operation processors access memory modules asynchronously, bus conflicts and memory conflicts occur. These conflicts are usually resolved by a two-stage arbitration scheme [88], [104], [109], [112]. If a certain source processor wants to send data to a target processor, the source processor will get the privilege to access a certain memory location with the help of the first arbitration stage. Then it will obtain a bus by the second stage. Then the source processor will write the message into the memory location. Next, the target processor also will get access to the same memory location via a bus through the two-stage arbitration scheme. Finally, the target processor will read the message.

There is no reason why a multiple bus system cannot be used in the *SIMD* mode of operation with message passing for interprocessor communication. In the *SIMD* mode of operation, the control unit will select a bus for both the source processor and the target processor. Since bus conflicts do not occur in this mode of operation, no arbitration is required. If the (*SIMD*) algorithm is mapped optimally, buses will not be wasted and resources will be utilized efficiently. In the message passing model, each processor has its own local memory. The processors can be allowed to communicate directly without

the need for using a common shared memory. When two processors need to communicate, the control unit will select a free bus for that purpose, and the two processors will communicate through the selected bus. Usage of multiple bus systems in this context was not given much attention in the past. Recently, however, some researchers have explored the possibility of using multiple bus systems in *SIMD* and message passing environments. Works reported in [41], [44], [49], [77], [83], [84], [108], and [144] address multiple bus systems, directly or indirectly, in the above stated context.

1.4 Design Issues of Parallel Architectures

In designing a parallel architecture, the architect is faced with many design parameters such as physical size, power consumption, processor characteristics, instruction sets, memory design, interconnection strategy, the technology to be used, and cost [2], [65], [124]. Taking all these parameters into consideration, it is extremely difficult (if not impossible) to do a theoretical treatise on designing an optimal parallel architecture. At the functional level, the interconnection network truly represents the parallel architecture. In fact, often the distinguishing feature of a parallel processing system from a single processing system is the interconnection network. Therefore, in this dissertation, we only concentrate on the design of the interconnection network.

Design issues of an interconnection network vary depending on whether the target is a general purpose machine or a special purpose machine. A general purpose machine is one which can solve almost any problem with acceptable speedup. Due to their wide applicability, general purpose machines generally have a better market potential. A special purpose machine is one which can solve problems in a given problem domain with greater speed. Due to the smaller market potential and the special purpose hardware

needed, those machines are often more expensive. However, due to the speed demands of many applications (such as weather forecasting, image analysis, and neural computing) and the advancement of technology, the trend is to build more and more special purpose machines [52], [133]. The focus of this dissertation is on designing special purpose parallel machines.

In designing a general purpose architecture, some of the desirable properties of an interconnection network are small diameter, small degree, larger bisection width, fault tolerance, regularity, and expandability. Some of these requirements are conflicting with each other. For example, decreasing node degree tends to decrease the bisection width and to increase the diameter. For a special purpose architecture, the architect should consider some other requirements in addition to those required by a general purpose architecture. A special purpose architecture should solve problems belonging to a particular application domain much faster than a general purpose architecture does. A particular problem domain is characterized by a set of algorithms, each one having the same pattern of communication. Therefore, a special purpose architecture is also an algorithmically specialized architecture.

For the computer architect, to design a special purpose architecture, only the knowledge of only the problem domain at hand will not be adequate. The architect should also have some knowledge of the target architecture as to its topological class, mode of operation, and communication strategy. Depending on the choice of the target architecture, the architect usually comes up with a different interconnection network for the same problem domain. In this dissertation, we narrow down the target architecture's topology to multiple bus systems.

1.5 Research Objectives

In a real computing environment, parallel algorithms and parallel architectures are inseparable from one another. On the one hand, a good parallel algorithm may not be effective in solving a problem if the selected architecture (interconnection topology) does not efficiently support the communications required by the algorithm. On the other hand, a good architecture may not be effective if the algorithm does not efficiently utilize the facilities of the architecture. Therefore, designing a parallel architecture which can efficiently solve problems belonging to a particular problem domain is of paramount importance. The literature on the design of algorithmically specialized parallel architectures is mainly aimed at array processors [79], [100], [124]. The research community has not paid much attention to the possibility of using other topologies as algorithmically specialized architectures.

In the most general sense, the objective of this dissertation is to develop a formal methodology to design optimal multiple bus architectures favoring a given class of parallel algorithms. Our treatise is of combinatorial nature. We do not specifically address hardware issues. A closely related subject is the optimal mapping of parallel algorithms onto existing multiple bus systems. The problem of optimally mapping a given algorithm into a given architecture (direct link) has been well studied in the literature [23], [31], [120]. For the mapping problem, the target architecture is known a priori. But for designing an architecture which can run the source algorithm(s) with maximum speed, only knowledge of the type of the architecture is known.

In [95], a method was discussed to design reconfigurable interconnection networks to realize given algorithms. In [144], the problem of designing an optimal multiple bus

architecture to realize a class of algorithms, called *Fan-in* computations, is addressed. Also in [145], a similar method was developed for the *Ascend/Descend* class of parallel algorithms. In [49], a multiple bus system emulating the *SIMD* hypercube was proposed. The work reported in [77] shows how to reduce the number of pins per chip for realizing n cyclic shifts on a multiple bus system.

Our approach is rather general; we do not restrict our attention only to a particular algorithmic class. The source class can be arbitrary. From a given algorithmic class, subject to some optimality constraints, we can extract a set of interconnection functions. This is the first stage in the design process. The set of interconnection functions so extracted dictates the potential communication capability of the target architecture which can run the source algorithm(s) with maximum speed. Once the set of interconnection functions has been determined, we will aim to construct an optimal multiple bus system which realizes the interconnection function set. This is the second stage. We will analyze the computational difficulties of both stages. We will also investigate how the design process can be methodically performed by exploiting certain regularities present in many parallel algorithms. We will also report on the merits of the multiple bus system over a direct link interconnection network realizing the same algorithmic class. We will further analyze the fault tolerance of the constructed multiple bus system.

Another objective of the dissertation is to study the trade off between cost and speed. We will analyze how an optimal multiple bus system favoring a given algorithmic class can be constructed when certain components in the target multiple bus system are specified.

1.6 Model and Design Criteria

In this section, we first present the model to be used throughout the dissertation. The model serves several purposes. First, it states all the assumptions made in the dissertation. Second, it allows us to make research objectives more specific and more detailed. Third, it gives the foundation for the mathematical approach used to achieve the research objectives. The model presented consists of two parts. In the computational model, we regard a parallel algorithm as a collection of computational nodes. We represent a parallel algorithm by a graph whose vertices represent the computational nodes of the algorithm and whose edges represent the flow dependencies of the algorithm. In the architectural model, we specify all of the relevant features of the target multiple bus system. Later in this section, we present our design criteria, in which we specify our objective functions conforming to the presented model.

1.6.1 Computational Model

Parallel algorithms can be represented at different levels of abstraction. In the lowest level of abstraction, a parallel algorithm consists of a set of computations (also called *computational nodes*) and a set of data transfers among those computational nodes. In the highest level of abstraction, a parallel algorithm consists of a set processes and a set of edges among processes, where each process is a coherent set of computations. Several methods have been used in the literature for representing parallel algorithms, such as, the Problem Graph [23], Task Precedence Graph [76], Task Interaction Graph [120], and the Data Dependence Graph [149]. These graphs represent parallel algorithms at different abstract levels. For example, the Task Precedence Graph is a higher abstraction of the algorithm than the Data Dependency Graph. Vertices of the Task Precedence Graph

are processes while those of the Data Dependency Graph are computational nodes. For our purpose, we will represent the source algorithm in the lowest level of abstraction. Therefore, our representation will consist of a set of computational nodes and a set of edges among those nodes.

Depending on the source algorithm, a single computation could be a simple operation such as comparison of two numbers or a compound operation such as multiplication of two matrices. Irrespective of the actual operation of the computational node, however, we assume that computational nodes are indivisible. That is, a computational node must be executed by a single processor and that processor will do so without any interruption. We also assume that the source algorithm is *homogeneous* from the computational point of view; that is, each computational node takes the same amount of time on a single processor. This is a reasonable assumption since we can always break heterogeneous computational nodes into smaller homogeneous nodes.

In the Data Dependency Graph, two vertices u and v have a directed edge from u to v if and only if the computation at v depends on the computation at u. In a such a graph, four kinds of dependencies can exist between two vertices u and v [149].

(*a*) *Flow dependence* : The value of a variable computed by u is used by v.

(*b*) *Antidependence* : The value of a variable used by u is later changed by v.

(*c*) *Output dependence* : The value of a variable computed by u is later changed by v.

(*d*) *Control dependence*: Computation u is a conditional statement whose output decides the computation v.

Only flow dependence requires that data be transferred from u to v. Our target multiple bus system would have a direct communication from one processor to another

if and only if the source algorithm requires a computation assigned to the first processor to transfer data to a computation assigned to the second processor. Therefore, in the graphical representation of the source algorithm, only flow dependencies must correspond to edges. We assume that the order of execution of the computational nodes in the given parallel algorithm is known a priori. In this dissertation, the graphical representation of a parallel algorithm will be called *Computation Flow Graph* (*CFG* for short).

If A is a parallel algorithm, then the corresponding *CFG* C_A is constructed as follows. Each vertex of C_A is a computation in algorithm A. Each vertex of C_A will be assigned an integer label such that two vertices have the same label if they are to be executed concurrently to achieve maximum speed. There will be a directed edge from vertex u to vertex v if and only if computation v is flow dependent on computation u. Other dependencies (anti, output, and control dependencies) in the algorithm will not correspond to edges in C_A. However, those dependencies will be reflected in the labeling of the vertices in C_A. Labeling of the vertices should satisfy the requirement that vertex v has a higher label than vertex u if there is a dependency from u to v in the source algorithm. We make the assumption that each edge in C_A corresponds to the same amount of data transfer.

From our algorithm representation, it is implied that concurrent computations of the algorithm are globally known. For example, computational nodes with the same label must be executed concurrently in order to achieve maximum speed. Thus our model implies that the source algorithm operates in the *SIMD* environment.

Let there be n labels in the *CFG* C_A. There are no dependencies in algorithm A among computations with the same label. That is, there are no edges among vertices with

the same label. Thus C_A is a directed n-partite graph. Vertices in the i^{th} partite set, that is, vertices with label i, belong to *concurrency level i*. By the definition, edges will be directed from a vertex at a lower level to a one at a higher level.

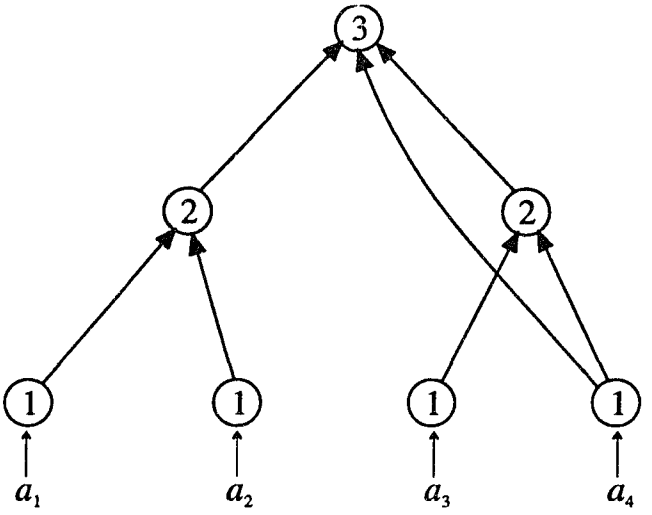


Figure 1.1: A *CFG* C_1

Figure 1.1 shows a *CFG* (denoted by C_1) with 7 vertices and three levels. It can be considered as the *CFG* corresponding to an algorithm which solves the following simple problem: given four numbers a_1, a_2, a_3, and a_4, find the maximum which is less than a_4. Vertex labels are written inside the circles representing the vertices of the *CFG*.

It should be emphasized that our graphical representation of parallel algorithms, with respect to the presented computation and communication models, agrees with the classification of algorithms given in [115]. Two algorithms represented by the same *CFG* belong to the same algorithmic class. For example, a single *CFG* represents all the algorithms in the *Ascend/Descend* class. In the rest of the dissertation, however, we use the word "algorithm" to mean either a single algorithm or a class of parallel algorithms. Since the *CFG* contains all the information of the algorithm needed for the construction

of an optimal multiple bus system, we can use the *CFG* without referring to the algorithm from which it was constructed.

1.6.2 Architectural Model

In this dissertation, our attention is on *SIMD* multiple bus systems which use message passing model for interprocessor communication. We use the abbreviation *MBS* to represent such a multiple bus system. An *MBS* consists of a set of processors, a set of buses and a set of interfaces. An interface is a device which connects a processor to a bus. In combinatorial analysis of *MBS*s in the literature, interfaces are frequently referred as "pins" or "ports" [49], [77]. In this dissertation, however, we use the term "interface" consistently. An interface connecting a bus to a processor will be called a *driver* or a *receiver* depending on whether the data transfer is from the processor to the bus or from the bus to the processor. Each processor is identical and assumed to have its own local memory. Figure 1.2 shows an *MBS* with eight processors and four buses.

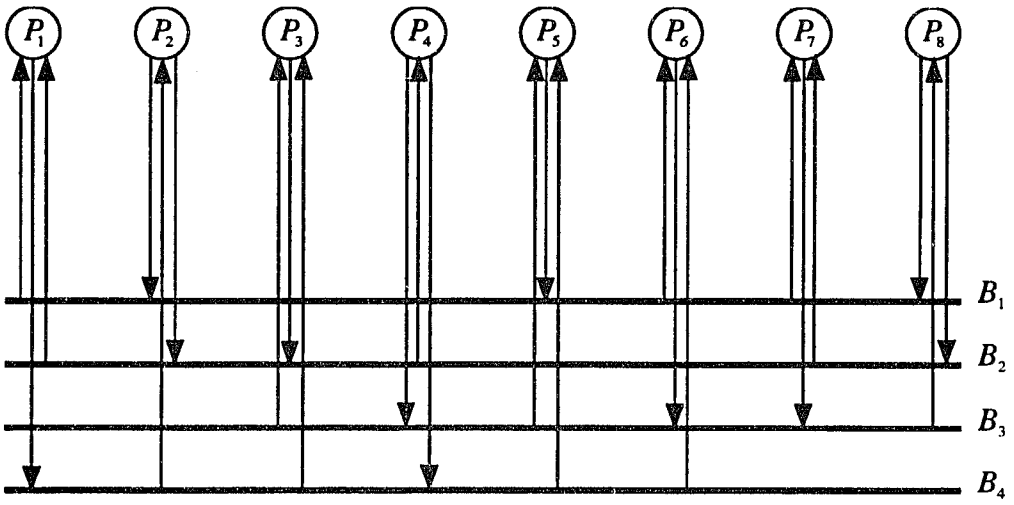


Figure 1.2: An *MBS* with eight processors and four buses.

In our model, buses are assumed to be half duplex in the sense that they can carry data only in one direction at a time; however, data can be transferred in opposite directions at different times. If two processors are involved in bidirectional communication, they should use two buses for data transfers.

Since our target *MBS* uses message passing model for communication, it does not contain global memory. All communications are done between processors using buses. However, if one wishes to include global memory to an *MBS*, it can be easily done. For example, we can include global memory modules such that each memory module is connected to every bus. Obviously, there are other possibilities. Construction of an *MBS* which can use a message passing model as well as a shared memory model is a possible extension to this work.

Some researchers have investigated the potential advantages of an *MBS* as an interconnection network. In [19], [49], [138], an *MBS* has been conveniently represented by a hypergraph. The vertices of the hypergraph corresponds to processors and hyperedges correspond to buses. But for the optimal design of *MBS*s, the hypergraph representation has a major drawback. When a bus is represented by a hyperedge, the only information it represents is whether a certain processor is connected to a certain bus. The direction of the connecting port (interface) is not shown. Therefore, unless all interfaces are assumed to be bidirectional, the hypergraph does not convey all the information of the *MBS*. In this dissertation, for optimality considerations, we need to distinguish an input port (receiver) from an output port (driver). Therefore, hypergraph representation cannot be used. We represent an *MBS* graphically by a directed bipartite graph called Multiple Bus Graph (*MBG* for short). If $G_1 = (X, Y, E)$ is an *MBG*, then X, Y, and E correspond

to the set of processors, set of buses, and the set of interfaces, respectively, of the represented *MBS*. An edge in E from a vertex in X (Y) to a vertex in Y (X) represents a driver (receiver) in the *MBS*. Figure 1.3 represents the *MBG* corresponding to the *MBS* shown in Figure 1.2. Notice that vertex x_i represents processor P_i, for $1 \leq i \leq 8$, and vertex y_j represents bus B_j, for $1 \leq j \leq 4$. For our analysis, the *MBG* is only a convenient graphical representation of the *MBS*. On many occasions, we will use *MBS* and *MBG* synonymously.

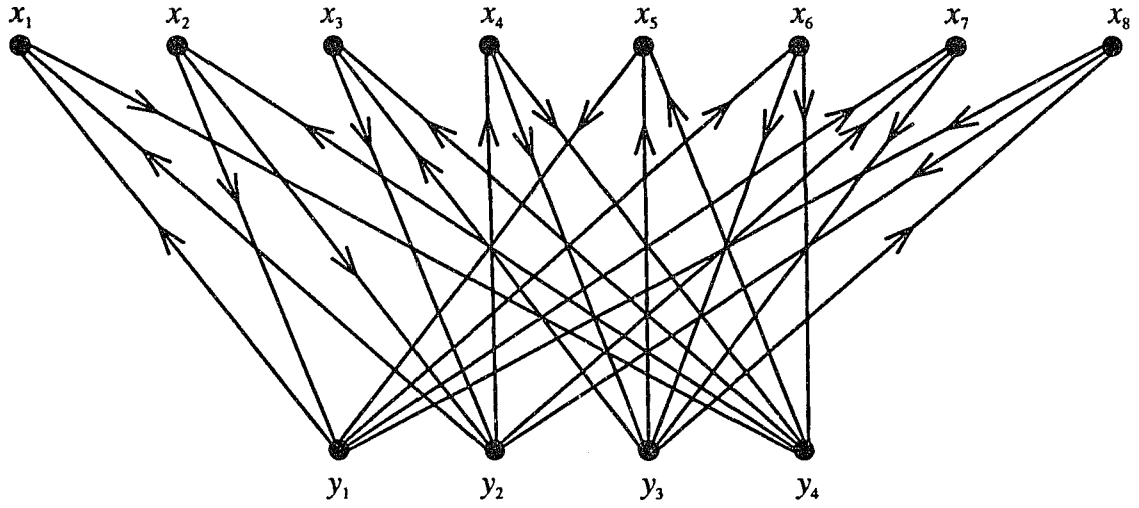


Figure 1.3: *MBG* corresponding to the *MBS* shown in Figure 1.2.

1.6.3 Design Criteria

As stated in Section 1.5, given a source algorithm, our objective is to design an optimal *MBS* which will run the source algorithm with maximum possible speed. One of the basic assumptions we make in this dissertation is that the given parallel algorithm A is not designed for any particular (underlying) architecture. Suppose, on the contrary, that algorithm A which solves problem $\mathscr{X}$ is written for a particular architecture $\mathscr{Y}$. Let M be the optimal *MBS* designed to realize A. Then, other than solving problem $\mathscr{X}$ efficiently,

M will try to emulate architecture $\mathcal{Y}$, which is an unnecessary side effect. If the design of *M* was not restricted by architecture $\mathcal{Y}$, we would possibly have obtained a better *MBS* to solve problem $\mathcal{X}$. Thus, in this research we assume that the source algorithm *A* is not written for any particular architecture and is the best algorithm to solve the problem at hand.

Let *M* be the target *MBS*. Then, computations in algorithm *A* must be carried out by the processors in *M*. Similarly, data transfers (communications) in *A* must be carried out by buses and interfaces in *M*. Construction of *M* should adhere to the following optimization criteria.

(1) Maximum speed

(2) Minimum cost

Unfortunately, the above two represent conflicting requirements. Since we do not modify the given source algorithm for designing an optimal *MBS*, there is an upper bound on the speed (and a corresponding lower bound on the execution time) with which one can run the source algorithm on any *MBS*, and that bound is determined by algorithm *A* itself. This is the *ideal speed* of algorithm *A*. Similarly, there is a lower bound on the cost of an *MBS*, which corresponds to the cost of a single processor without buses and interfaces (excluding the buses and interfaces necessary for devices other than the processor). These are the two extremes in designing an *MBS* to realize a given algorithm. Our main reason for having a parallel processing system is to increase the speed of applications beyond what could be achieved using a single processor system. So, one primary objective of the dissertation is to construct a minimal cost *MBS* which can run the given source algorithm *A* at its ideal speed. One secondary objective is the design of an *MBS* with specified number of components, rather than with the number of

components dictated by the ideal speed requirement. When practical issues are taken into consideration, the latter would be the best approach. An *MBS* which can run source algorithm *A* at its ideal speed may not be cost optimal.

1.7 Outline of the Dissertation

In this section, we briefly outline how the intended research objectives will be achieved subject to the presented model. As mentioned before, one primary objective is to construct an optimal *MBS* which can run a given algorithm (or a class of algorithms) at the ideal speed. An *MBS* hosting a given parallel algorithm will perform computations using processors and communications using buses and interfaces. Given an arbitrary *CFG* C_A, construction of the minimal cost *MBS*, which can run algorithm *A* at its ideal speed, is an extremely difficult computational problem. To somewhat alleviate the difficulty, we break the problem into two stages.

In the first stage, we assign a set of processors to the vertices of the given *CFG*. This is called *processor assignment* and is addressed in Chapter 2. The first criterion for processor assignment is that the total number of processors required to perform the computations of the *CFG* is minimum. From the *CFG*, we construct another graph called interconnection function graph (*IFG* for short). Vertices of the *IFG* are the processors allocated to computations of the *CFG*. Edges of the *IFG* correspond to data transfers among processors necessitated by the source algorithm. In the processor assignment stage, we do not completely disregard the number of buses and interfaces required for the target *MBS*. We use the number of edges in the *IFG* as an approximate measure of the number of buses and interfaces in the target *MBS*. As a consequence, the second criterion for the processor assignment is to minimize the number of edges in the *IFG*. We prove that

optimal processor assignment is an *NP*-Hard problem. However, we show that certain regularities inherent with every, well behaved, parallel algorithm lend themselves to polynomial time solutions. For such algorithms, we provide an efficient algorithm to perform the processor assignment.

In the second stage, we pay attention to minimizing the number of buses and interfaces. We start the second stage with an *IFG*, which is obtained from a *CFG* by an optimal processor assignment. Our objective in this stage is to realize the communications dictated by the edges of the *IFG* using buses and interfaces. This is called *bus assignment* and is addressed in Chapter 3. The first criterion for the bus assignment is to minimize the number of buses and interfaces in the target *MBS*. The second criterion is to minimize the time the target *MBS* takes to perform the communication primitives dictated by the given *CFG*. We prove that the optimal bus assignment is an *NP*-Hard problem.

The construction of an optimal *MBS* realizing a given *IFG* automatically addresses two other important issues. First, it addresses the problem of designing an optimal *MBS* which emulates an existing *SIMD* architecture. Second, it addresses the problem of designing an optimal *MBS* which can perform a specified set of interconnection functions. Due to these reasons, we will address the bus assignment problem in more detail. We will analyze important cases where a polynomial time solution can be obtained. We show that the problem is solvable in polynomial time when the number of interconnection functions associated with the *IFG* is two. Based on that algorithm we provide an efficient heuristic algorithm to solve the general bus assignment problem.

In Chapter 4, we show how to perform bus assignment in polynomial time when the *IFG* is vertex symmetric. We show that an *IFG* is vertex symmetric if and only if it

is the Cayley color graph of a finite group and its generating set. We utilize this property to analyze vertex symmetric *IFG*s. Due to the importance of vertex symmetric *IFG*s, we will perform a detailed analysis of the bus assignment problem for this case. We will give a polynomial time algorithm which performs the bus assignment of a vertex symmetric or regular *IFG*. Furthermore, we will show the superiority of an *MBS* over a static interconnection network realizing vertex symmetric *IFG*s, in terms of the number of ports per processor, the number of neighbors per processor, and the diameter.

In Chapter 5, we address the fault tolerance capabilities of an *MBS* constructed from a vertex symmetric *IFG*. We study the behavior of the *MBS* in the case of a single bus failure, single interface failure, and a single processor failure. We obtain necessary and sufficient conditions for the *MBS* to be fault-tolerant in each of the above cases. We will also obtain a measure of performance degradation due to such component failures. Furthermore, we will show how to add redundancy to the *MBS* in order to improve its fault tolerance.

In Chapter 6, we address a secondary objective of the dissertation, namely the problem of constructing an *MBS* with a given number of processors and a given number of buses which can run a given source algorithm at maximum possible speed given the restrictions. In this case, the cost function for optimality includes only the number of interfaces. We first consider the case, where only the number of processors is specified. In that case, we show that the number of buses can also be proportionally decreased without any performance penalty. We show how to construct a regular *IFG* with the specified number of processors. We then consider the case, where the number of buses is specified. In that case, the number of processors needed for maximum speed is equal

to the optimal number of processors. Here we show how to combine buses of the optimal *MBS* to form a new *MBS* which has the specified number of buses.

In Chapter 7 we provide the summary of the dissertation along with concluding remarks. We point out how several concepts addressed in the dissertation can be extended for future research work.

CHAPTER 2

PROCESSOR ASSIGNMENT

In this chapter, we consider the first stage of the primary objective of the dissertation. Given the *CFG* C_A of a parallel algorithm A, we will assign the set of computational nodes in C_A to a set of processors such that the target architecture has minimum cost and can execute algorithm A in its ideal speed. We will show that the processor assignment problem is equivalent to that of partitioning the vertex set of the *CFG* with certain constraints. By partitioning the vertex set of the *CFG*, we will obtain an *IFG*. After stating the optimization criteria, we will prove that the optimal processor assignment is an *NP*-Hard problem. We will then show how to exploit certain regularities present in every well-behaved parallel algorithm in order to perform the processor assignment in polynomial time. For the case when the *CFG* possesses such regularities, we will present an efficient algorithm which solves the processor assignment problem in polynomial time. Furthermore, we will show that, when the *CFG* is regular the resulting *IFG* is also regular. Finally, in this chapter, we will present a proper scheduling for the assigned processors.

2.1 Preliminaries

The underlying undirected graph of a *CFG* is connected. This is because the algorithm which produced the *CFG* solves a single problem and therefore its computational nodes are interrelated. Given a *CFG* C_A, we need to find a set of processors such that each vertex in C_A is assigned to exactly one processor in the set. One naive approach would be to assign one processor for each vertex in C_A. Another naive approach would be to assign a single processor to all the vertices of C_A. The former approach would have

the fastest computation time but would be the most expensive. On the other hand, the latter approach would be the least expensive but also the slowest. Our intention is to construct a least expensive *MBS* which runs the source algorithm at its ideal speed. Since each processor must be allocated to a disjoint subset of vertices of the *CFG*, the processor assignment is equivalent to the partition of the vertex set of C_A. Therefore, we will use the terms "vertex partition" and "processor assignment" interchangeably.

Computations of the algorithm are performed by processors. Since we do not alter the source algorithm, the amount of computation required by the algorithm, whether executed by one processor or more, is fixed. We are only allowed to allocate computations to different processors. Therefore, the *computation time* is only affected by the number of processors and the computation schedule. The source algorithm allows certain computations to be executed in parallel. For example, all the computations of the first concurrency level of the *CFG* can be executed in parallel. Similarly, all the computations in the second level can be executed in parallel. However, no computation in the second level can be executed before all the computations in the first level are done. This is due to possible dependencies (not necessarily flow dependencies) among computational nodes in different concurrency levels. The following lemma gives a straightforward result.

Lemma 2.1: Let n_A be the number of concurrency levels in the *CFG* C_A. Then the minimum computation time is $n_A t_A^{cp}$, where t_A^{cp} is the computation time of a single node of the *CFG* on a single processor.

Therefore, the algorithm stipulates a lower bound on the computation time. The minimum computation time $n_A t_A^{cp}$ required by the algorithm will be called its *ideal computation time*. To run algorithm *A* in its ideal speed, computations must be performed

in time $n_A t_A^{cp}$. Therefore, the assignment of processors to *CFG* C_A should be such that the processors can perform the computations of the algorithm A with computation time $n_A t_A^{cp}$. Note that there is very little or no significance of the term *ideal communication time*, because, the minimum communication time is zero and it corresponds to the uniprocessor case.

If two vertices in the same level of C_A were assigned to the same processor, then those two computations cannot be performed at the same time. Therefore, unless each processor is assigned vertices belonging to distinct levels of C_A, we cannot attain the ideal computation time of the source algorithm. We provide the following definition in order to characterize a processor assignment which will attain ideal computation time.

Definition 2.1: A partition ζ on the vertex set of a *CFG* C_A will be called a *level-disjoint vertex partition* if each subset has vertices from distinct partite sets.

In this chapter, unless otherwise stated, a partition of the vertex set of a *CFG* is always assumed to be a level-disjoint partition. Let $C_A = (V, E)$ be a *CFG*. Let ζ be a (level-disjoint) vertex partition of V. Associated with ζ, we will define an edge colored directed graph G_A^{ζ} called *Interconnection Function Graph* (*IFG* for short). If V_1, V_2, ..., and V_x are the subsets of partition ζ, then for each subset V_i, there exists a unique vertex v_i in G_A^{ζ}, and vice versa. There is an edge of color r from v_i to v_j in G_A^{ζ} if and only if there is an edge from u_i to u_j in C_A, where $u_i \in V_i$, $u_j \in V_j$, and u_i is in concurrency level r. An *IFG* may have multiple edges from one vertex to another. Since the vertex partitioning of C_A uniquely determines G_A^{ζ}, with slight abuse of notation, we may write $G_A^{\zeta} = \zeta(C_A)$. Since the *CFG* C_A is connected, so is the *IFG* $\zeta(C_A)$. Every vertex of $\zeta(C_A)$

corresponds to a set of computations, at most one from each concurrency level, of algorithm A assigned to a single processor.

Definition 2.2: For a given parallel algorithm A and a partition ζ of C_A, the collection of computational nodes corresponding to each subset of the partition is called a *subtask* of algorithm A.

Example 2.1: Figure 2.1 shows a possible vertex partition of the *CFG* C_1 shown in Figure 1.1. Dotted lines enclose the vertices belonging to a subset. Figure 2.2 represents the corresponding *IFG*, which we denote by G_1. The three computational nodes in the subset V_4 form the substask associated with vertex v_4.

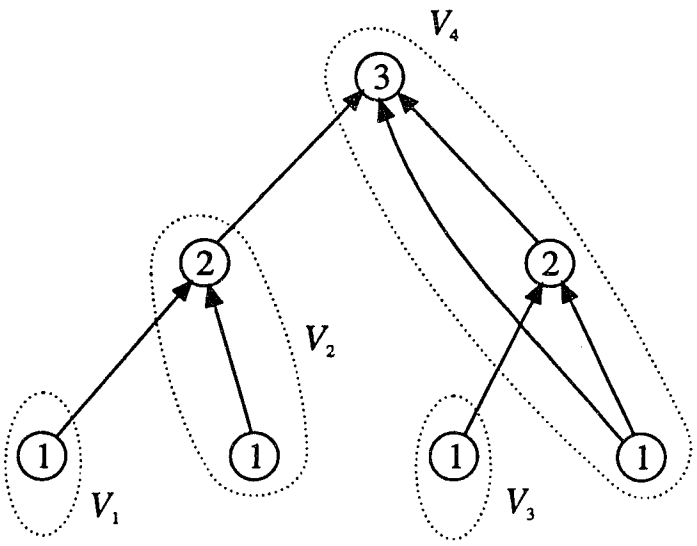


Figure 2.1: A vertex partition of the *CFG* C_1.

Even though the *IFG* is an intermediate step of our design process, it is an important entity in its own right. Vertices of the *IFG* correspond to processors, and edges correspond to data transfers among processors. Therefore, an *IFG* can be considered as a direct link interconnection network with each edge corresponding to a unidirectional

link. Colors of the edges represent concurrency of data transfers, that is, in the *IFG*, the set of edges belonging to a certain color corresponds to an interconnection function (in the most general sense) in the represented *SIMD* machine. Thus the *IFG* $\zeta(C_A)$ represents a direct link, *SIMD* interconnection network which can run algorithm *A* in its ideal speed.

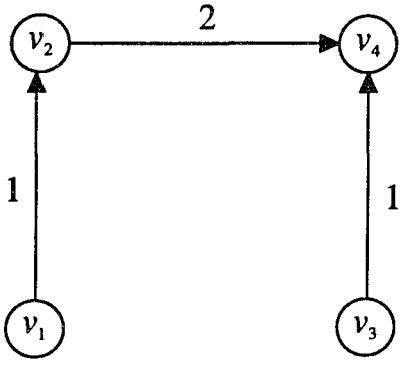


Figure 2.2: The *IFG* G_1 corresponding to the vertex partition of Figure 2.1.

By definition, $\zeta(C_A)$ has enough processors to execute algorithm *A* in its ideal computation time. Addition of more processors will not reduce the computation time. This is true because, if two computations are to be executed sequentially, whether they were assigned to the same processor or to different processors, the computation time will not be affected. Recall that we do not modify the source algorithm; specifically, we do not distribute a single computation among several processors. Therefore, any interconnection network with the number of processors equal to the number of vertices in $\zeta(C_A)$ can attain the ideal computation time of algorithm *A*. Next we consider the criteria for partition ζ to be optimal.

2.2 Optimal Processor Assignment

Any partition ζ of the vertex set of a *CFG* will produce an *IFG* which corresponds to a system with enough processors to reach the ideal computation time. We are interested

in constructing a minimal cost *MBS* which has ideal computational time and minimum possible communication time. Let M_A^ζ be an *MBS* which can perform each of the interconnection functions associated with $\zeta(C_A)$ in one step. To evaluate the merits of the partition, we must obtain the cost of the M_A^ζ as well as the computation and communication times of algorithm *A* running on M_A^ζ. Since we haven't constructed the *MBS* M_A^ζ yet, some of the information regarding M_A^ζ are unavailable at this stage.

The computation time and the number of processors in M_A^ζ are directly available from partition ζ. Only an *approximate measure* for the communication time, the number of buses, and the number of interfaces in M_A^ζ are available from partition ζ. Edges of the *IFG* $\zeta(C_A)$ correspond to data transfers among processors of M_A^ζ dictated by algorithm *A*. The larger the number of edges in $\zeta(C_A)$, the larger is the amount of communication required. Therefore, as an approximation, we consider the number of edges in $\zeta(C_A)$ as a measure of the communication time of algorithm *A* running on M_A^ζ. Since the communication in M_A^ζ is effected by buses and interfaces, for the present stage, we also approximate the number of edges in $\zeta(C_A)$ as a measure of the number of buses and interfaces in M_A^ζ. Therefore, our goal of the processor assignment is to find a vertex partition ζ such that $|V(\zeta(C_A))|$ and $|E(\zeta(C_A))|$ are minimum. As the following lemma shows, obtaining the minimum value of $|V(\zeta(C_A))|$ is straightforward.

Definition 2.3: Let C_A be a *CFG* with n concurrency levels. Then, the number of vertices in level r is denoted by $\alpha_r(C_A)$, $1 \le r \le n$. Also $max\{\alpha_r(C_A) : 1 \le r \le n\}$ is denoted by $\alpha(C_A)$.

Lemma 2.2: The minimum number of processors needed to run algorithm *A* in its ideal speed is $\alpha(C_A)$.

Proof: To obtain minimum computation time, concurrent computations needed by the source algorithm must be executed by distinct processors. □

For example, in Figure 1.1, $\alpha_1(C_1) = 4$, $\alpha_2(C_1) = 2$, and $\alpha_3(C_1) = 1$. Therefore, $\alpha(C_A) = 4$ and an optimal *MBS* needs 4 processors. Any machine with fewer processors than $\alpha(C_A)$ cannot execute the source algorithm A in ideal computation time. Since one of our design criteria is that the target machine runs A in the ideal computational time, the number of processors must be at least $\alpha(C_A)$.

Unlike the case for $|V(\zeta(C_A))|$, there is no straightforward method to find the minimum value of $|E(\zeta(C_A))|$. Furthermore, in general, both $|V(\zeta(C_A))|$ and $|E(\zeta(C_A))|$ cannot be minimized at the same time. This fact will be clarified in the following example.

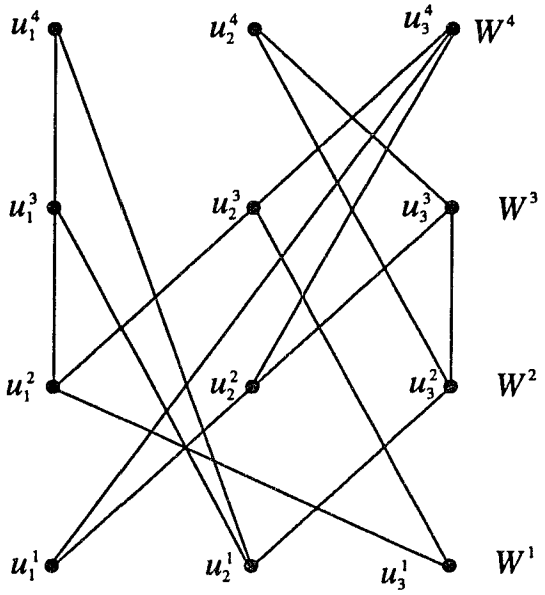


Figure 2.3: A *CFG* C_2.

Example 2.2: Figure 2.3 shows a four level *CFG*, denoted by C_2, with twelve vertices (directions of the edges are implied). Clearly, $\alpha_1(C_2) = \alpha_2(C_2) = \alpha_3(C_2) = \alpha_4(C_2) = \alpha(C_2)$

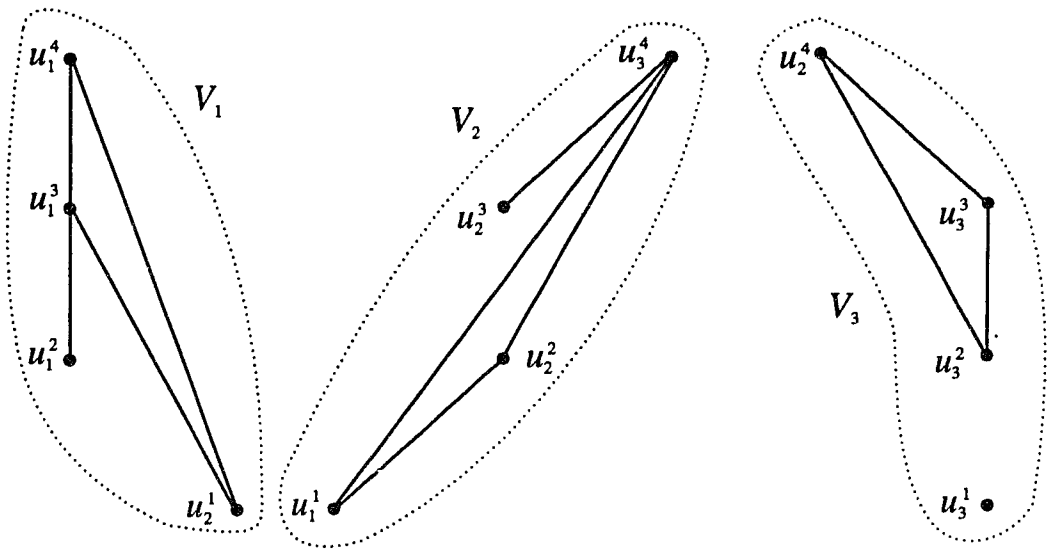


Figure 2.4: Induced subgraphs of partition ζ_1 on $V(C_2)$.

= 3. Therefore, any level-disjoint partition would have at least three subsets. There are 16 edges in C_2. A partition of C_2 with cardinality three† would have four vertices in each subset, one vertex from each level. Since the total number of edges in C_2 is fixed, the number of edges in $\zeta(C_2)$ would be minimum if the number of edges induced by the subsets of partition ζ is maximum. It is clear from the figure that the maximum number of edges which can be induced by a 4-element subset of vertices is four. An example is the subset $\{u_2^1, u_1^2, u_1^3, u_1^4\}$. One can be easily convinced that there does not exist a vertex partition with cardinality three such that each subset induces four edges. The best possible, cardinality-three partition would induce four edges on two of the subsets and three edges on the other subset. Figure 2.4 shows the induced subgraphs of such a partition, denoted by ζ_1. The three subsets are $\{u_2^1, u_1^2, u_1^3, u_1^4\}$, $\{u_1^1, u_2^2, u_2^3, u_3^4\}$, $\{u_3^1, u_3^2, u_3^3, u_2^4\}$. Since partition ζ_1 induces 11 edges, the *IFG* $\zeta_1(C_2)$ resulting from

†The cardinality of a partition is the number of subsets it generates.

partition ζ_1 would have 5 edges. Now consider the induced subgraphs of the cardinality four partition $\zeta_2 = \{\{u_3^1, u_1^2, u_2^3\}, \{u_1^1, u_2^2, u_3^4\}, \{u_2^1, u_1^3, u_1^4\}, \{u_3^2, u_3^3, u_2^4\}\}$ shown in Figure 2.5. That partition induces 12 edges, three on each subset. Therefore, the *IFG* $\zeta_2(C_2)$ corresponding to partition ζ_2 shown would have only four edges.

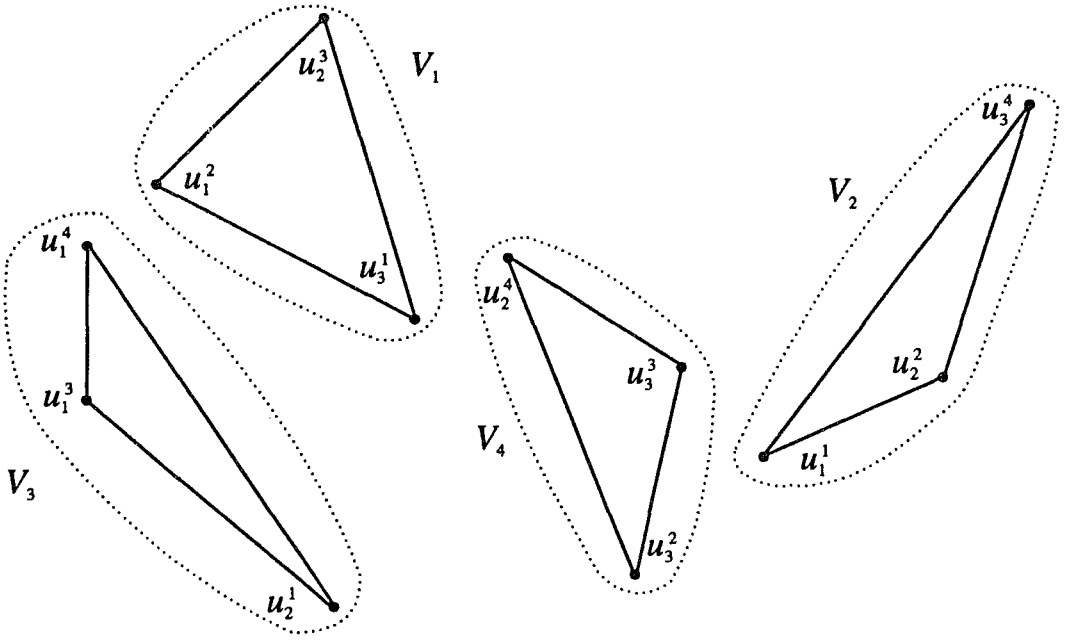


Figure 2.5: Induced subgraphs of partition ζ_2 of $V(C_2)$.

Thus, in general, one cannot find a single partition ζ which minimizes both $|V(\zeta(C_A))|$ and $|E(\zeta(C_A))|$ simultaneously. Another conclusion that can be drawn from the above example is that by increasing the number of processors above $\alpha(C_A)$, we may be able to decrease the communication time.

Since both $|V(\zeta(C_A))|$ and $|E(\zeta(C_A))|$ cannot be minimized simultaneously (in general), the cost function for the processor assignment must include terms to reflect the relative costs of vertices and edges of the target *IFG*. For each vertex of the *IFG*, we will associate a cost κ_1. Also, for each edge we will associate a cost κ_2. Therefore, the cost

of an *IFG* C_A is $\kappa_1|V(C_A)| + \kappa_2|E(C_A)|$. Using this cost measure, we can define the optimal processor assignment as follows.

Definition 2.4: A vertex partition ζ of a given *CFG* C_A is called an *optimal vertex partition* (or simply an *optimal partition*) if $\kappa_1|V(\zeta(C_A))| + \kappa_2|E(\zeta(C_A))|$ is minimum, taken over all partitions ζ. The processor assignment corresponding to an optimal partition is called an *optimal processor assignment.*

Optimal processor assignment is very similar to the problem of program partitioning covered in the literature [10], [17], [24], [60], [81], [123], [132]. Solving the partitioning problem for only two processors with the aid of network flow algorithms is addressed in [132]. In [17], a partition strategy was proposed for a rectangular grid with unequal weights. The general partition problem has been shown to be *NP*-Hard in [53], [81]. The processor assignment problem we consider here is a restricted version of the general partition problem. Our model for the processor assignment requires that no two vertices from the same level belong to the same subset. We next prove that the optimal processor assignment problem, although a restricted version of the general partition problem, is also *NP*-Hard.

2.3 Computational Complexity of the Optimal Partition Problem

According to the definition of the optimal partition, directions of the edges in a *CFG* do not play any role. Therefore, in obtaining the computational complexity of the problem, we will ignore the directions of the edges of the *CFG*. To use known results from computational complexity theory, we convert the above optimization problem into a decision problem [53]. The decision problem, which we call the *Level Partitioning* (*LP*) problem, is defined next.

Level Partitioning (*LP*) Problem:

INSTANCE: An undirected n-partite graph G and non negative integers κ_1, κ_2, and K.

QUESTION: Can we find a partition ζ on the vertex set of G such that each subset contains vertices from distinct partite sets and $\kappa_1 * |V(\zeta(G))| + \kappa_2 * |E(\zeta(G))| \leq K$?

Theorem 2.1: *LP* is *NP*-Complete.

Proof: It is straightforward to show that *LP* belongs to class *NP*. To show that it is *NP*-Hard, we will transform an instance of the Three Dimensional Matching (3*DM*) problem [53] into an instance of the *LP* problem and show that the 3*DM* instance has a matching if and only if the *LP* instance has a solution. For completeness we will state the 3*DM* problem [53].

INSTANCE: Set $M \subseteq X \times Y \times Z$, where, X, Y and Z are disjoint sets each having p elements.

QUESTION: Does M contain a matching, i. e., a subset $M' \subseteq M$ of cardinality p such that no two elements in M' agree in any coordinate?

For the 3*DM* instance, let $|X| = |Y| = |Z| = p$ and $|M| = q$. From this, we will construct an instance C of *LP*, that is, a *CFG* C and constant K. For each element $x_i \in X$, there is a vertex x_i in C with label 1. Similarly, for each element $y_i \in Y$ and $z_i \in Z$ there exist vertices y_i and z_i with labels 2 and 3, respectively. For each element $m_j = (y^j, x^j, z^j)$ in M there are 9 vertices $a_j[1]$ through $a_j[9]$ in C. Figure 2.6 shows how those vertices are connected and labeled. We define $C = (V, E)$ by

$$V = \{x_i\} \cup \{y_i\} \cup \{z_i\} \cup \bigcup_{j=1}^{p} \{a_j[k] : 1 \leq k \leq 9\}, \text{ and}$$

$$E = \bigcup_{j=1}^{p} E_j, \text{ where,}$$

$E_j = \{(x^j, a_j[1]), (a_j[1], a_j[2]), (a_j[2], a_j[3])\} \cup \{(y^j, a_j[4]), (a_j[4], a_j[5]), (a_j[5], a_j[6])\} \cup \{(z^j, a_j[7]), (a_j[7], a_j[8]), (a_j[8], a_j[9])\} \cup \{(a_j[3], a_j[6]), (a_j[6], a_j[9])\}$.

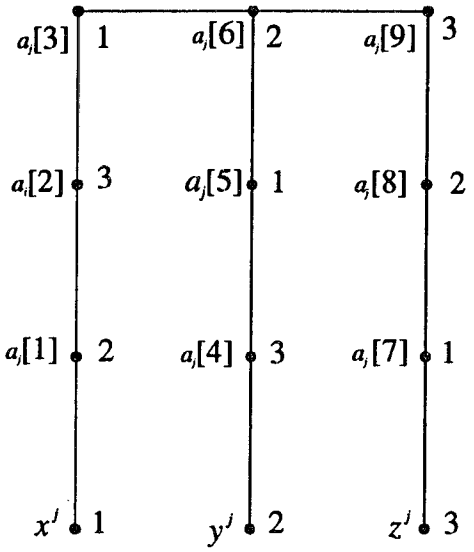


Figure 2.6: Component of C corresponding to $m_j \in M$.

According to the way C was constructed, $|V(C)| = 3p + 9q$, where each level contains $p + 3q$ vertices. Therefore, $\alpha_1(C) = \alpha_2(C) = \alpha_3(C) = \alpha(C) = p + 3q$. The total number of edges in C is $11q$. Since there are no triangles in C, any 3-element subset of vertices in C can induce at most two edges. We will attempt to form a vertex partition of C such that each subset has vertices with distinct labels and induce two edges. Therefore, we set the value of K to be equal to $\kappa_1(p + 3q) + \kappa_2(11q - 2(p + 3q)) = \kappa_1(p + 3q) + \kappa_2(5q - 2p)$.

Suppose that the $3DM$ instance has a matching $M' \subseteq M$. For each element $m_j \in M'$, form the 3-element subsets of the vertices $\{x^j, a_j[1], a_j[2]\}$, $\{y^j, a_j[4], a_j[5]\}$, $\{z^j, a_j[7], a_j[8]\}$, and $\{a_j[3], a_j[6], a_j[9]\}$ in C. These subsets are shown in Figure 2.7 by enclosing the elements of each subset in a dotted curve. For each element $m_j \notin M'$ form

the 3-element subsets of the vertices $\{a_j[1], a_j[2], a_j[3]\}$, $\{a_j[4], a_j[5], a_j[6]\}$, and $\{a_j[7], a_j[8], a_j[9]\}$ in C (see Figure 2.8). There are p elements in M' and $(q - p)$ elements which are in M but not in M'. Hence the above partition has $4p + 3(q - p) = p + 3q$ subsets. Also, each subset induces two edges. Therefore, the total number of edges induced by the partition is $2(p + 3q)$. It can be easily seen that the smallest cycle in C has 14 vertices. Therefore, there can be at most one edge from one subset to another. Hence $|E(\zeta(C))| = 11q - 2(p + 3q) = 5q - 2p$. Also, $|V(\zeta(C))| = p + 3q$. Therefore, ζ is a solution to the *LP* problem.

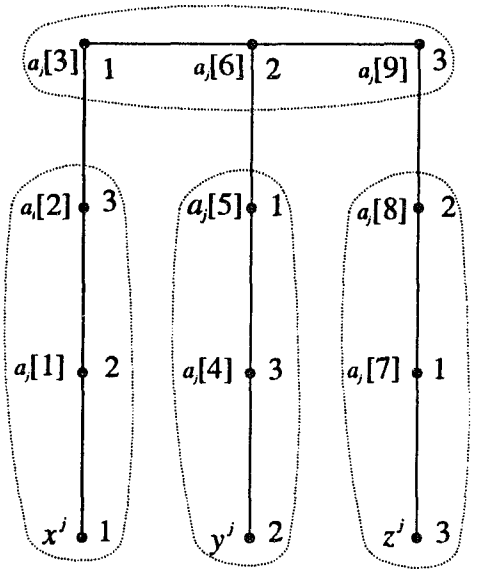


Figure 2.7: Partition of a component of C corresponding to $m_j \in M'$.

Now suppose that the *LP* instance C has a solution, that is, there exists a partition ζ such that $\kappa_1|V(\zeta(C))| + \kappa_2|E(\zeta(C))| \leq K = \kappa_1(p + 3q) + \kappa_2(5q - 2p)$. We first claim that ζ consists of $p + 3q$ 3-element subsets and each subset induces two edges. Since there are only three labels in C, one subset can have at most three elements. Let s_1, s_2, and s_3 be the number of 1-element, 2-element, and 3-element subsets in ζ, respectively. Then,

$s_1 + 2s_2 + 3s_3 = |V(C)| = 3(p + 3q)$. Therefore, $2s_2 + 3s_3 \leq 3(p + 3q)$; equality holds only when $s_1 = 0$. The inequality can also be written as $\frac{4}{3}s_2 + 2s_3 \leq 2(p + 3q)$. Therefore,

$$s_2 + 2s_3 \leq 2(p + 3q); \text{ equality holds only if } s_1 = s_2 = 0. \quad (1)$$

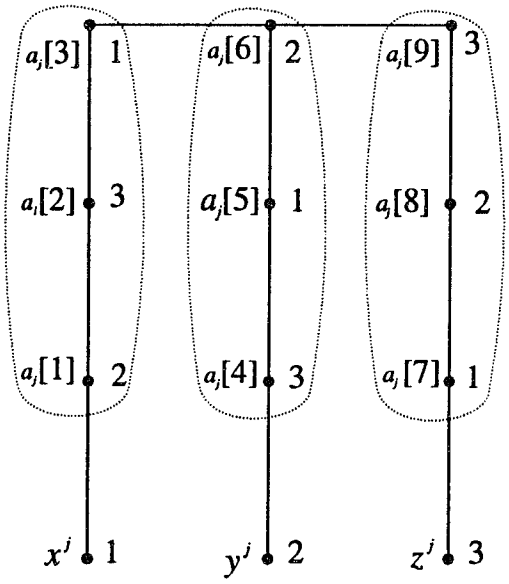


Figure 2.8: Partition of a component of C corresponding to $m_j \notin M'$.

A 1-element subset does not induce any edges. A 2-element subset can induce at most 1 edges. Also, a 3-element subset can induce at most two edges. Therefore,

$$|E(\zeta(C))| \geq 11q - (2s_3 + s_2) \quad (2)$$

Combining the two inequalities (1) and (2), we get $|E(\zeta(C))| \geq 11q - 2(p + 3q)$, that is, $|E(\zeta(C))| \geq 5q - 2p$; equality holds only if $s_1 = s_2 = 0$. From Lemma 2.2, $|V(\zeta(C))| \geq p + 3q$. Therefore, $\kappa_1|V(\zeta(C))| + \kappa_2|E(\zeta(C))| \leq K = \kappa_1(p + 3q) + \kappa_2(5q - 2p)$ can be true only if $s_1 = s_2 = 0$, $|V(\zeta(C))| = p + 3q$, and $|E(\zeta(C))| = (5q - 2p)$. This proves our claim that ζ contains $p + 3q$ subsets and each subset induces two edges. Consider element x_i in X. Vertex x_i must be included with vertices $a_j[1]$ and $a_j[2]$ for some value of j to form a 3-element subset. This is possible when x_i is in m^j, that is, $x_i = x^j$ (see Figure 2.6).

Once this is done, $a_j[3]$ must be included with $a_j[6]$ and $a_j[9]$ to form a 3-element subset. Now $a_j[5]$ and $a_j[4]$ must be combined with some y_k such that $y_k = y^j$ to form a 3-element subset. Similarly, $a_j[8]$ and $a_j[7]$ must be combined with some z_l such that $z_l = z^j$. From the construction of C, (x_i, y_k, z_l) must be an element of M. Taking all the elements in X, we can thus obtain p (x^j, y^j, z^j) tuples which will exhaust all the elements in X, Y and Z. Thus we have a matching for the $3DM$ instance. □

Since the general problem is *NP*-Hard, we can utilize two approaches in finding the solution to the processor assignment problem. One approach is to use some features which are inherently present in many of the well known parallel algorithms, so that the processor assignment problem can be solved in polynomial time. Another approach is to use a polynomial time algorithm (probably heuristic) such that the solution is not necessarily optimal but does not deviate from the optimal solution by more than a fixed percentage.

2.4 Exploiting Regularities in the Source Algorithm

For the optimal processor assignment problem, we use the first of the above approaches. The choice of the first approach can be justified as follows. Even though obtaining a solution to the general problem has some theoretical interest, it is of very little practical interest. The vast majority of practical algorithms show some degree of regularity in their structures. In fact, certain regularities are present in every well behaved algorithm since spaghetti code writing is completely obsolete. We also favor having regular features in the target *MBS*. An irregular *MBS* would be unattractive in many ways. Thus, by paying attention only to *CFG*s with some regular features (to be defined precisely next), we do not reduce the generality of the design procedure insofar as actual

algorithms and actual architectures are concerned. However, if one is interested in finding a partition of a general *CFG*, Algorithm 2.1 given in Section 2.5 will serve the purpose.

Definition 2.5: A *CFG* C_A is said to be *locally regular* if the following three conditions are satisfied.

(*a*) Every vertex in the same level has the same indegree (outdegree).

(*b*) If there is an edge from a vertex in partite set W^j to a vertex in partite set W^k, then $k = j + 1$.

(*c*) If the relation $|W^j| \leq |W^{j+1}|$ ($|W^j| \geq |W^{j+1}|$) holds for some j, then the same holds for every j.

The above definition characterizes almost every well-behaved parallel algorithm. Statement (*a*) stipulates that all concurrent computations in the algorithm behave similarly. This is true for many algorithms such as binary search, bitonic sort, matrix multiplication, and prefix sum computations. Statement (*b*) stipulates that a certain computation can receive data only from computation(s) in the immediately preceding level. This is also generally true for most practical *SIMD* algorithms. Statement (*c*) stipulates that the number of concurrent computations does not vary irregularly as we move along the algorithm from the beginning to end. This is also true for many parallel algorithms. For example, in *Ascend/Descend* class of algorithms [115], the width of the computation is constant, that is $|W^j| = |W^{j+1}|$ for all j. Also, *CFG*s belonging to many algorithms are (binary) fan in trees [3], [22], [29], [136], [144]. For such *CFG*s, the width of the algorithm gradually decreases, that is, $|W^j| > |W^{j+1}|$ for all j. These algorithms belong to the *Fan-in/Fan-out* algorithmic class [144].

2.5 Optimal Vertex Partitioning of a Locally Regular *CFG*

We will show that if a *CFG* is locally regular, its optimal partition can be found in polynomial time. In order to do that, we will use some graph matching techniques. For completeness we will state some fundamental definitions and results related to graph matching.

Definition 2.6: Two distinct edges in a graph G are *independent* if they are not adjacent in G. A set of pairwise independent edges of G is called a *matching* in G. A matching of maximum cardinality is called a *maximum matching* [30].

For a given graph G, a maximum matching can be found in polynomial time [30], [56]. For the present purpose, we are only interested in matchings in bipartite graphs. A maximum matching of a bipartite graph can be found in $O(n^{5/2})$ time, where n is the number of vertices in the graph [67].

Definition 2.7: The set of all vertices adjacent to a vertex v in a graph is called *neighborhood of* v and is denoted by $N(v)$. The set of all vertices adjacent to a set S of vertices is similarly called the *neighborhood of* S and is denoted by $N(S)$.

We state the following theorem due to Hall without proof [30].

Theorem 2.2: Let $G = (X, Y, E)$ be a bipartite graph. Then X can be matched to a subset of Y, if and only if, for each subset S of X, $|N(S)| \geq |S|$. □

Theorem 2.3: Let $G = (X, Y, E)$ be a bipartite graph such that each vertex in X has degree p and each vertex in Y has degree q. Then, either X can be matched to a subset of Y, or Y can be matched to a subset of X depending on whether $|X| \leq |Y|$ or $|Y| \leq |X|$.

Proof: For any subset B of vertices of G, let E_B be the set of edges adjacent with the vertices of B. Suppose that $p \geq q$. Then, since $p|X| = q|Y|$, it follows that $|X| \leq |Y|$. Let S

be a subset of X. Then $|E_S| = p|S|$, and $|E_{N(S)}| = q|N(S)|$. Since $N(S)$ represents all the neighbors of S, it follows that $E_S \subseteq E_{N(S)}$. Therefore $|E_S| \leq |E_{N(S)}$, and hence $p|S| \leq q|N(S)|$. Therefore, $|S| \leq |N(S)|$, and from Theorem 2.2, X can be matched to a subset of Y. Similarly, if $p \leq q$, we can show that Y can be matched to a subset of X. □

Recall that a *CFG* is a directed n-partite graph with the property that vertices in partite set W^j has incoming edges originating from partite set W^k, $k < j$. A *CFG* C_A is locally regular if each vertex in partite set W^j has the same indegree/outdegree and all the edges incident on vertices in W^j are originating from partite set W^{j-1}.

Theorem 2.4: Let C_A be a locally regular *CFG*. Then, for any level-disjoint partition ζ, $|E(\zeta(C_A))| \geq |E(C_A)| - |V(C_A)| + \alpha(C_A)$.

Proof: Let a be the cardinality of partition ζ. Let V_i be any subset of partition ζ. Then V_i contains at most one vertex from a given partite set, say W^j. Furthermore, a vertex in V_i belonging to W^j can be adjacent with a vertex in either W^{j-1} or W^{j+1}. Therefore, V_i cannot contain any undirected cycles. Therefore, if E_i is the set of edges induced by V_i, then $|E_i| \leq |V_i| - 1$. Therefore, $\sum_{i=1}^{a} |E_i| \leq \sum_{i=1}^{a} (|V_i| - 1) = \sum_{i=1}^{a} |V_i| - a = |V(C_A)| - a$. According to Lemma 2.2, $a \geq \alpha(C_A)$. Therefore, $\sum_{i=1}^{a} |E_i| \leq |V(C_A)| - \alpha(C_A)$. There can be at most one edge of a given color from a vertex in one subset V_1 to a vertex in another subset V_2. Therefore, $|E(\zeta(C_A))| = |E(C_A)| - \sum_{i=1}^{a} |E_i|$. Hence, $|E(\zeta(C_A))| \geq |E(C_A)| - |V(C_A)| + \alpha(C_A)$. □

It should be noted that the above theorem is true not only for locally regular *CFG*s. Any *CFG* satisfying Condition (b) of Definition 2.5 will have the lower bound stated in the above theorem. In the next theorem we show that the lower bound can always be reached for a locally regular *CFG*.

Theorem 2.5: Let C_A be a locally regular *CFG*. Then there exists a vertex partition ζ such that $|E(\zeta(C_A))| = |E(C_A)| - |V(C_A)| + \alpha(C_A)$.

Proof: We will provide a constructive proof. Suppose that $|W^{j+1}| \leq |W^j|$, for $1 \leq j \leq n-1$. Then, $\alpha(C_A) = \alpha = |W^1|$. Let L^j be the subgraph of C_A induced by the vertex set $W^{j+1} \cup W^j$, for $1 \leq j \leq n-1$. Clearly, L^j is a bipartite graph. We construct the subset of vertices V_1 through V_α as follows. Initially, V_i contains exactly one vertex from W^1, for $1 \leq i \leq \alpha$. According to Theorem 2.3, since $|W^2| \leq |W^1|$ by the hypothesis, vertex set W^2 can be matched to a subset of W^2 in L^1. Let M^1 be such a matching. Each vertex in W^1 is contained in exactly one subset V_i, $1 \leq i \leq \alpha$. If $u_1 \in V_i$, and $(u_1, u_2) \in M^1$, then let $V_i = V_i \cup \{u_2\}$. In other words, if a vertex in the subset V_i is included in the matching M^1, insert the complement vertex also in V_i.

Since $|W^3| \leq |W^2|$, W^3 can be matched to a subset of W^2 in L^2. Let M^2 be such a matching. If $u_2 \in V_i$ and $(u_2, u_3) \in M^2$, then let $V_i = V_i \cup \{u_3\}$. Repeat this procedure until all the vertices in C_A are inserted into the subsets V_1 through V_α. Every time we update V_i, we insert a new vertex which is adjacent with exactly one vertex in the existing V_i. Therefore, the number of edges induced by V_i is $|V_i| - 1$. Hence the total number of edges induced by the partition is $\sum_{i=1}^{\alpha} (|V_i| - 1) = |V(C_A)| - \alpha(C_A)$. Thus, $|E(\zeta(C_A))| = |E(C_A)| - |V(C_A)| + \alpha(C_A)$. The proof is similar if $|W^{j+1}| \geq |W^j|$, $1 \leq j \leq n-1$. The only difference is that we start with each V_i, $1 \leq i \leq \alpha$, containing exactly one vertex from W^n. □

Thus, when C_A is a locally regular *CFG*, there exists a partition ζ which minimizes both $|V(\zeta(C_A))|$ and $|E(\zeta(C_A))|$ simultaneously. The following algorithm provides an optimal partition of a locally regular *CFG*.

Algorithm 2.1

INPUT: Locally regular *CFG* C_A with n levels.
OUTPUT: Optimal partition $\{V_1, V_2, ..., V_\alpha\}$ of $V(C_A)$.

begin
  If $|W^1| < |W^n|$ **then**
    begin $\alpha = |W^1|$; $j = 1$; $\odot$ = '+'; **end**;
  else
    begin $\alpha = |W^n|$; $j = n$; $\odot$ = '−'; **end**;
  for $i = 1$ **to** α **do**
  $V_i = \{v_i^j\}$, where $W^j = \{v_1^j, v_2^j, ..., v_\alpha^j\}$;
  L^j := subgraph induced by $W^{j\odot 1} \cup W^j$;
  while ($\odot$ = '+' **and** $j \neq n$) **or** ($\odot$ = '-' **and** $j \neq 1$) **do**
    begin
      $M^j = \{(v_1^{j*1}, v_1^j), (v_2^{j*1}, v_2^j), ..., (v_{|W^{j*1}|}^{j*1}, v_{|W^{j*1}|}^j)\}$
        matches W^{j*1} to a subset of W^j;
      for $i = 1$ **to** α **do**
      if $((x, y) \in M^j)$ **and** $(x \in V_i)$ **then** $V_i = V_i \cup \{y\}$;
      $j = j + 1$;
    end;
end.

We can analyze the computational complexity of Algorithm 2.1 as follows. The first two compound statements take a constant amount of time. By using the algorithm for finding a maximum matching in a bipartite graph given in [67], the while loop takes $O(\alpha^{5/2})$ time. Furthermore, the while loop will be executed n times. Therefore, the total time it takes is $O(n\alpha^{5/2})$. Since $|V(C_A)|$ is $O(n\alpha)$, the time complexity can be expressed as $O(|V(C_A)|.\alpha^{3/2})$.

Optimality of the output of Algorithm 2.1 is guaranteed only if the *CFG* is locally regular. When the *CFG* is not locally regular, the algorithm produces a nearly optimal partition. Therefore, Algorithm 2.1 also provides a heuristic method for solving the general level-disjoint vertex partition problem.

Example 2.3: Consider the *CFG*, denoted by C_3, shown in Figure 2.9. Clearly, it is a locally regular *CFG*. It has four levels, i.e., four partite sets. Figure 2.10 shows an optimal

partitioning of its vertex set using our algorithm. The subsets are named 1, 2, 3, 4, 5, and 6. Figure 2.11 shows the corresponding *IFG*, which we denote by G_2. In G_2, edges of colors 1, 2, and 3 are represented by solid, broken, and dotted lines, respectively.

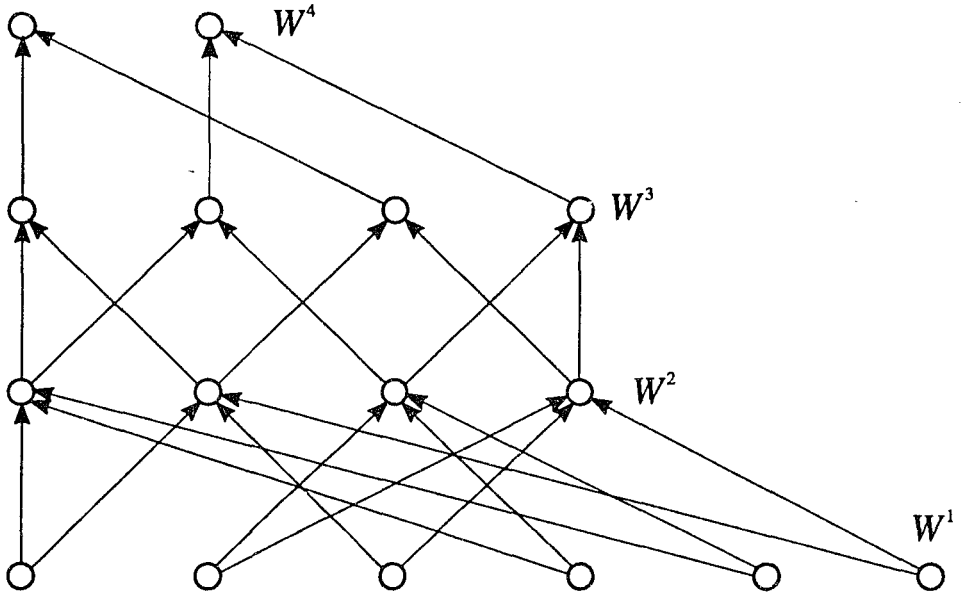


Figure 2.9: A locally regular *CFG*, C_3.

It is interesting to notice that the *CFG*s of many parallel algorithms such as bitonic sorting, *FFT*, and convolution, have stronger regularities. Specifically, each computational node (except in the first and the last concurrency levels) has two incoming edges and two outgoing edges. To emphasize the properties of such parallel algorithms, we provide the following definition.

Definition 2.8: A *CFG* is said to be *regular* if it is locally regular and every level has the same indegree (outdegree) 2, except that the indegree of vertices in the first level and the outdegree of the vertices in the last level are equal to zero.

Lemma 2.3: If a *CFG* is regular, then each partite set has the same number of vertices.

Proof: Let W^1, W^2, ..., W^n be the partite sets of the *CFG*. Then, there are $2|W^1|$ edges incident with the vertices in W^1. This is equal to the number of incoming edges to W^2. Since each vertex of W^2 has indegree 2, the number of vertices in W^2 is equal to $|W^1|$. Similarly, we can show that $|W^2| = |W^3| = ... = |W^{n-1}| = |W^n|$. □

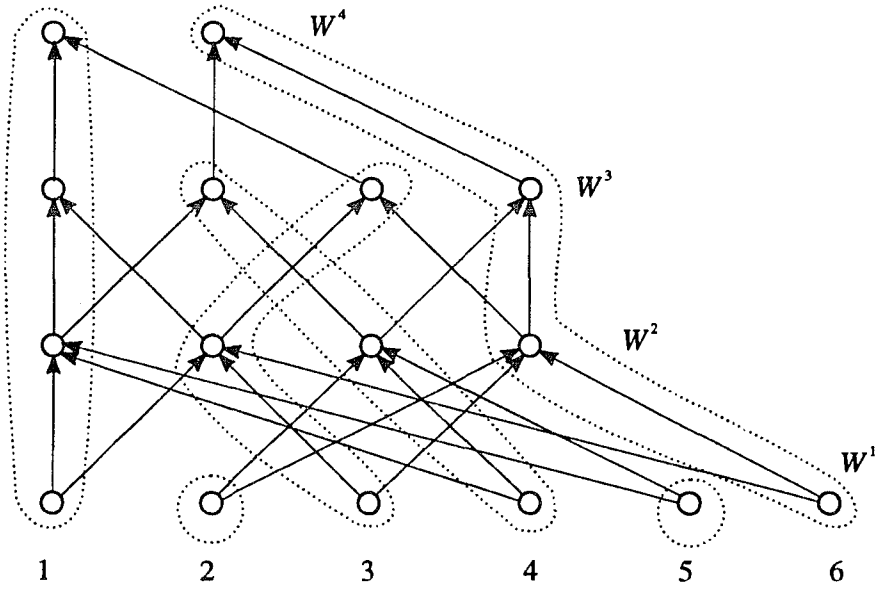


Figure 2.10: Optimal partition of $V(C_3)$ using Algorithm 2.1.

Therefore, an algorithm corresponding to a regular *CFG* has the same number of computations at each concurrency level. In other words, the algorithm has the same "width" throughout the computation. If we assign one processor per computational node at each level, then the same number of processors is needed at each level. This kind of algorithm uses processors very efficiently; no processor stays idle throughout the execution of the algorithm. When the *CFG* is regular, subtasks (see Definition 2.2) corresponding to any optimal partition consists of one node from each concurrency level. Therefore, every processor performs the same amount of computation. Next, we present two important properties of an *IFG* obtained by optimal partitioning of a regular *CFG*.

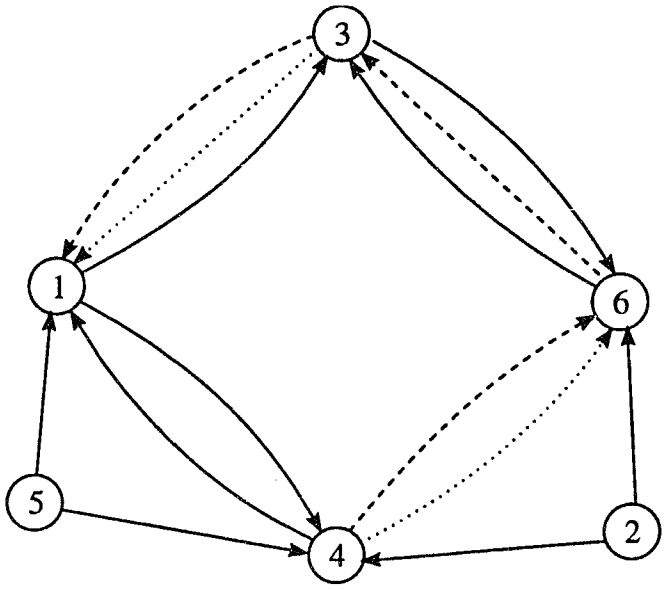


Figure 2.11: The *IFG* G_2 corresponding to the optimal partition shown in Figure 2.10.

Definition 2.9: An *IFG* is said to be *regular* if every vertex has exactly one outgoing edge from each color.

Theorem 2.6: Let C_A be a regular *CFG*. Then the *IFG* obtained by optimal partitioning of C_A is regular.

Proof: For any optimal partitioning of C_A, each subset V_i will have n vertices, one from each partite set, where n is the number of partite sets. In the subgraph induced by V_i, there is exactly one edge from a vertex belonging to W^j to a vertex belonging to W^{j+1}, $1 \leq j \leq n-1$. Therefore, V_i has one incoming edge of color 1, one outgoing edge of color 1, one incoming edge of color 2, one outgoing edge of color 2, and so on. Thus each vertex of the resulting *IFG* has one incoming edge and one outgoing edge from each color. Therefore, the resulting *IFG* is regular. □

Theorem 2.7: Let C_A be a regular *CFG* with n concurrency levels. Then the *IFG* obtained by optimally partitioning C_A has $n-1$ colors.

Proof: According to the model presented in Section 1.6, edges of C_A originating at concurrency level r are of color r, $1 \leq r \leq n-1$. Therefore, C_A has at most $n-1$ colors. By Theorem 2.6, every vertex of the resulting *IFG* has one outgoing edge and one incoming edge from each color r, $1 \leq r \leq n-1$. □

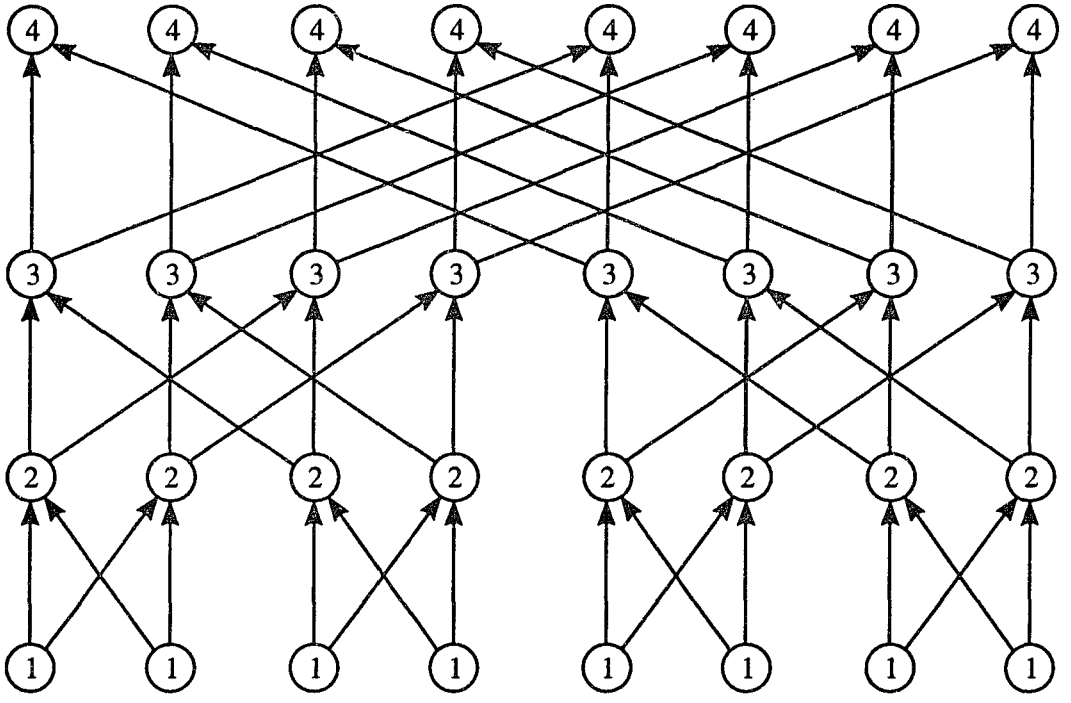


Figure 2.12: A *CFG*, C_4.

Example 2.4: Figure 2.12 shows the *CFG*, denoted by C_4, corresponding to *Ascend/Descend* class of parallel algorithms [115]. It is a 2-regular graph with 4 concurrency levels and 8 computational nodes in each level. Figure 2.13 shows an optimal partition of the vertex set of the *CFG* of Figure 2.12. Subsets are labeled with binary numbers from 000 through 111. Figure 2.14 shows the corresponding *IFG*, which we denote by G_3. It has three colors represented by solid, broken, and dotted lines. We will later see that the *IFG* G_3 shown in Figure 2.14 is not only regular but also vertex symmetric. We

will pay special attention to vertex symmetric *IFGs* in Chapter 4. If we replace each pair of unidirectional edges between adjacent vertices of G_3 by an undirected edge and ignore colors, we would obtain the 3-dimensional boolean hypercube.

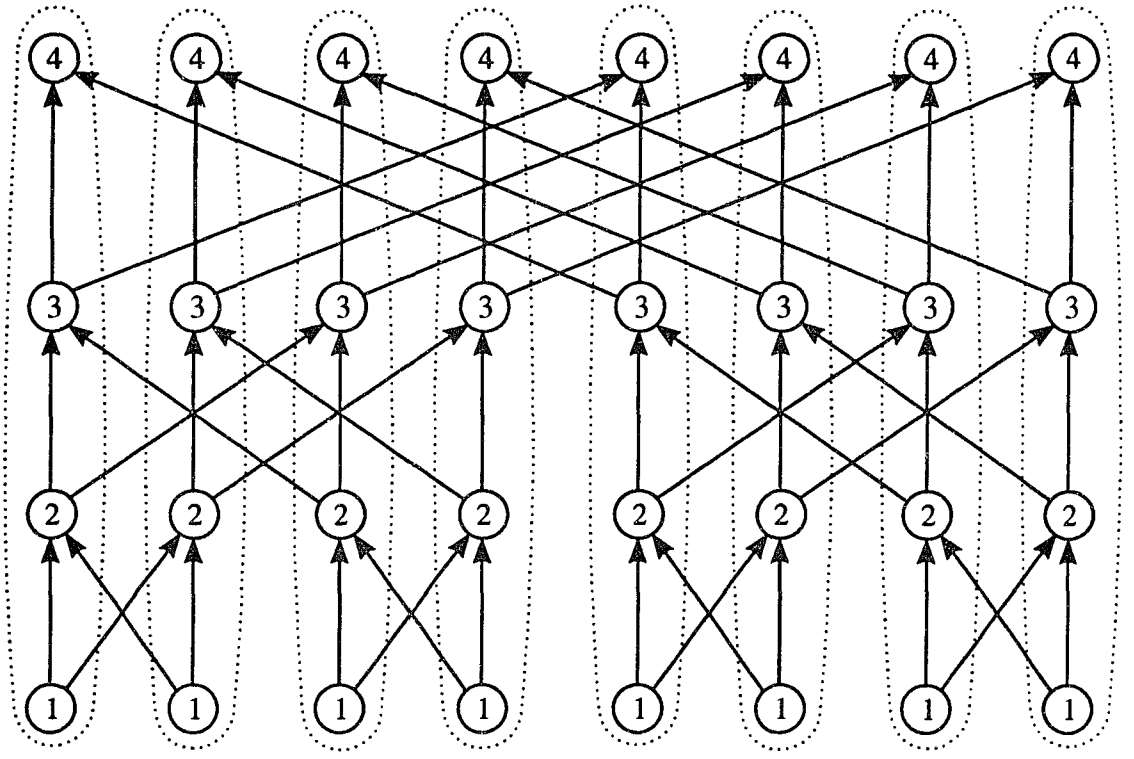


Figure 2.13: An optimal vertex partition of C_4.

2.6 Processor Scheduling

An important issue closely related to processor assignment is that of processor scheduling. Once an *IFG* has been constructed from a *CFG*, each processor is assigned a single subtask. Therefore, the assignment of processors to individual computational nodes is known. Let computational node u_i be assigned to processor P_j. Let t_i be the time at which processor P_j starts to compute u_i. For each node u_i, the knowledge of P_j and t_i

completely specifies the execution of the computation u_i. The set $\{(u_i, P_j, t_i) : u_i \in V(C_A)\}$ is called a *processor schedule* for algorithm A. When we have a vertex partition ζ of $V(C_A)$, all the pairs (u_i, P_j) are fixed. The execution time of the algorithm is decided by the third parameter t_i. Therefore, for the processor schedule to be complete, a time t_i must be assigned to every computational node u_i.

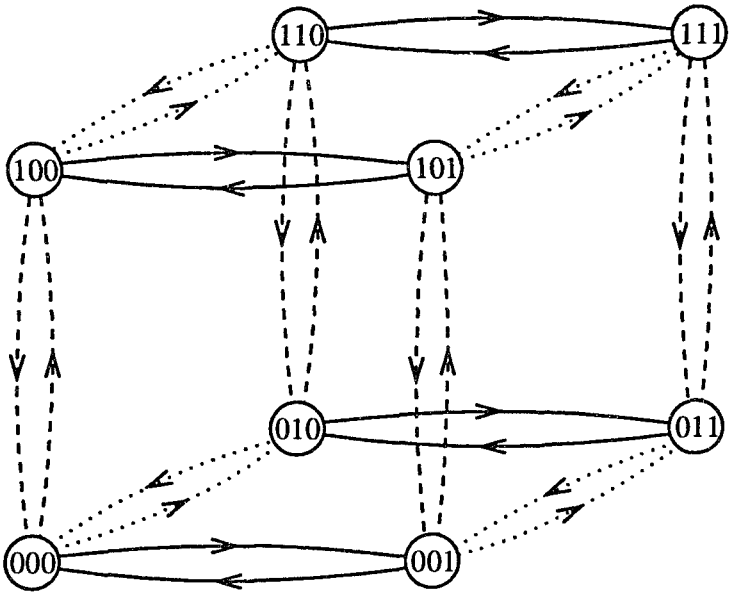


Figure 2.14: The *IFG* G_3 corresponding to the optimal partition shown in Figure 2.13.

As one must have expected, allocation of t_i to u_i is quite straightforward. Merely set $t_i = jt_A^P$, where j is the concurrency level to which u_i belongs and t_A^P is the time a computational node takes on a single processor. According to this scheduling, each processor will first execute its computational node for level 1. Then each processor executes its computational node for level 2, and so on. When a processor is executing its r^{th} computational node, we say that the processor is executing its subtask at level r. When

one processor is outputing data at level r, every other processor should be outputing data at level r. These transfers correspond to edges of color r in the *IFG*.

By the optimal processor assignment, each processor is allocated a single subtask. If the number of processors in the target *MBS* is limited, we may have to assign more than one subtask to a processor. In that case, in order to achieve the best speed, the order of execution of the subtasks by each processor must be correctly determined. This issue will be addressed in Chapter 6.

CHAPTER 3

BUS ASSIGNMENT

In this chapter, we address the second stage of the optimal *MBS* construction process. From a given *IFG* we construct an optimal *MBS* such that the *MBS* can perform each of the concurrent data transfers associated with the *IFG* in one time step. This will correspond to an optimal *MBS* which can run the source algorithm (from which the *IFG* was derived) at its ideal sped. We assume that the *IFG* has been constructed from a *CFG* by finding an optimal (or nearly optimal) level-disjoint vertex partition.

Bus assignment has other related applications also. An interconnection network for an *SIMD* machine is associated with a set of interconnection functions, where an interconnection function corresponds to concurrent data transfers from a set of source processors to a set of target processors [125]. Clearly, an *IFG* contains all the information about an interconnection network for an *SIMD* machine. Vertices represent processors and edges of the same color represent an interconnection function. This is the reason why the *IFG* (Interconnection Function Graph) has been given that name in this dissertation. The construction of an *MBS* from a given *IFG* addresses three different *MBS* design related issues.

(*a*) It is the second stage of the process of designing an optimal *MBS* realizing a given parallel algorithm.

(*b*) It provides a method to construct an optimal *MBS* emulating a given static interconnection network.

(*c*) It provides a method to construct an optimal *MBS* realizing a given set of interconnection functions.

Therefore, in the remainder of this dissertation, we will regard an *IFG* as representing either a set of interconnection functions or the communication pattern of an algorithmic class. Due to the wide applicability of the *IFG*, the tools developed in this chapter will be useful for the designers of *MBS*s in many respects. In [49], an *MBS* was proposed which emulates the *SIMD* hypercube. Using the tools developed here, we can design many different configurations of *MBS*s that emulate hypercubes with superior properties.

In this chapter, we show that the bus assignment problem is equivalent to the problem of partitioning the edge set of the *IFG* with specific constraints. This will be called color partition. We will show how broadcasting information can be incorporated into the *IFG*. We prove that the optimal color partition problem is *NP*-Hard. We also prove that when the *IFG* has only two colors, an optimal color partition can be found in polynomial time. This corresponds to the construction of an optimal *MBS* realizing two interconnection functions such as shuffle and exchange. Furthermore, based on the two color case, we develop an efficient heuristic algorithm to solve the general color partition problem. In Chapter 4, we show that the optimal color partition problem can be solved in polynomial time when the *IFG* is vertex symmetric.

3.1 Preliminaries

The vertices of the *IFG* directly represent to the processors in the target *MBS*. The edges of the *IFG* represent the communication requirement of the source algorithm. From the edge set of the *IFG*, we need to determine the set of buses and the set of interfaces of the target *MBS*. We need to address the following question: How can we determine buses, drivers and receivers to realize the communication pattern dictated by the *IFG*? In other words, how can we assign buses to the *IFG* $\zeta(C_A)$ such that the resulting *MBS* will

run algorithm A at its ideal speed? In order to answer this question, we next define some terminology used throughout the remainder of the dissertation.

Definition 3.1: In an *MBS*, we say that processor P_1 can transfer data to processor P_2 in *one step* if and only if there exists a bus B which is connected to processors P_1 and P_2 via a driver and a receiver, respectively.

Definition 3.2: Let A be a parallel algorithm and M be an *MBS*. Then M is said to *realize* A if the computational nodes of A can be assigned to processors in M such that concurrent computational nodes of A are assigned to distinct processors and concurrent data transfers of A can be carried out in a single step on M.

Therefore, for an *MBS* M to realize algorithm A, a necessary condition is: for every edge (v_i, v_j) in the *IFG* $\zeta(C_A)$, the *MBS* must be capable of transferring data from processor v_i to processor v_j in a single step. Otherwise, one communication edge in the source algorithm may correspond to more than one communication step in the *MBS*. Therefore, associated with every edge of the *IFG*, there must be a bus in the target *MBS*. However, this condition is not sufficient to guarantee that M realizes algorithm A. If two edges in the *IFG* correspond to concurrent data transfers (non broadcasting), there must be two distinct buses in the *MBS* to carry out those two data transfers.

One naive approach would be to assign a distinct bus for each edge in the *IFG*. But this may result in redundant buses and therefore may not be optimal. If two edges in the *IFG* have different colors, the source algorithm does not require the corresponding data transfers to be carried out concurrently. Hence, a single bus may be used for data transfers corresponding to both edges. If two edges in the *IFG* have the same color with different originating vertices, then we should assign a distinct bus for each edge for the

data transfers to proceed in parallel. If two edges in the *IFG* have the same color and originate from the same vertex, then whether we need only one bus or two buses for the data transfer is ambiguous. If the data (information) corresponding to the two edges are different, we still need two buses; because, otherwise, data collision will occur. On the other hand, if the two data items corresponding to the two edges are the same, then the situation corresponds to a broadcasting operation and only one bus is required. From the way an *IFG* is constructed, broadcasting information is not available in the *IFG*. In Section 3.3, we show how edge coloring rules of the *IFG* can be modified in order to convey broadcast information. Until then we will assume that the given *IFG* does not model broadcasting operations. In the following, we define the concept of an *MBG* realizing an *IFG*. It should be noted that *MBG* is merely a graphical representation of an *MBS*.

Definition 3.3: Let G be an *IFG* and G^* be an *MBG*. Then G^* is said to *realize* G if there exists a bijective mapping ψ: $V(G) \rightarrow X(G^*)$ satisfying the following conditions.

(1) For every directed edge $(v_i, v_j) \in E(G)$, there exists a vertex $y_{ij} \in Y(G^*)$ such that the vertices $\psi(v_i)$, y_{ij}, $\psi(v_j)$ represent a directed path in G^*.

(2) If (v_{i_1}, v_{j_1}) and (v_{i_2}, v_{j_2}) are edges in $E(G)$ of the same color, then $y_{i_1 j_1}$ and $y_{i_2 j_2}$ (as defined in (1)) are distinct vertices in $Y(G^*)$.

It is clear from Definitions 3.2 and 3.3 that the *MBS M* realizes algorithm A if and only if M realizes the *IFG* $\zeta(C_A)$. Therefore, we can refer to an *IFG* without referring to its origin. Next we show how graph theory can be utilized to construct an *MBG* realizing a given *IFG*.

3.2 Construction of an *MBG* from a given *IFG*

Definition 3.4: A partition π of the edge set of an *IFG* G will be called a *color partition* of G if no subset of π has more than one edge of the same color.

Let $E_j<\pi>$, $1 \leq j \leq b$, be the subsets of edges corresponding to the color partition π of G. Denote by $H_j<\pi>$ the subgraph of G induced by $E_j<\pi>$, $1 \leq j \leq b$. (Subgraph $H_j<\pi>$ itself can be considered as an *IFG* having all distinct color edges). Construct an *MBG* G^* as follows. For each vertex $v_i \in V(G)$, associate a unique vertex $x_i \in X(G^*)$. Also, for each subgraph $H_j<\pi>$ associate a unique vertex $y_j<\pi> \in Y(G^*)$. Establish an edge from x_i to $y_j<\pi>$ (from $y_j<\pi>$ to x_i) if vertex v_i is in $H_j<\pi>$ and there is an edge in $H_j<\pi>$ which is directed away from (towards) x_i. It is easy to verify that *MBG* G^* realizes *IFG* G since they satisfy Conditions (1) and (2) of Definition 3.3. Thus, constructing an *MBS* realizing a given *IFG* is equivalent to finding a color partition π of the given *IFG*.

Since π uniquely determines the *MBG*, we may write (with slight abuse of notation) $G^* = \pi(G)$. We will often not distinguish between the *MBG* and the corresponding *MBS*. We may use $\pi(G)$ to represent either. In Section 2.1 we stated that an *IFG* is connected. Therefore, any graph can be considered as an *IFG* if it is connected, directed and edge colored. Notice that any subset $E_1 \subseteq E(G)$ of edges can be assigned to a single bus. The color partition merely imposes the condition that only non concurrent data transfers are assigned to the same bus.

Definition 3.5: Let E_1 be a subset of edges in the *IFG* G and H_1 be the subgraph induced by E_1. If E_1 is assigned to a single bus, then the set of interfaces attached to that bus will be denoted by either $J(E_1)$ or $J(H_1)$.

Therefore, for color partition π, $J(E_j<\pi>)$ (or $J(H_j<\pi>)$ represents the set of interfaces connected to bus $B_j<\pi>$, $1 \le j \le b$. We can also consider $J(E_j<\pi>)$ as the set of edges incident with $y_j<\pi>$ in the *MBG* $\pi(G)$. The set of processors connected to bus $B_j<\pi>$ is the set of all vertices in $H_j<\pi>$. The following lemma gives a straightforward result.

Lemma 3.1: $E(\pi(G)) = \bigcup_{j=1}^{b} J(E_j<\pi>)$, and $|E(\pi(G))| = \sum_{j=1}^{b} |J(E_j<\pi>)|$.

Definition 3.6: Let G be an *IFG* with c colors. Then the number of edges of G belonging to color r will be denoted by $\beta_r(G)$, $1 \le r \le c$. Also, $\max\{\beta_r(G) : 1 \le r \le c\}$ will be denoted by $\beta(G)$.

Throughout the remainder of the dissertation, unless otherwise stated, we will adhere to the following notation. G represents an *IFG*. The number of edges of color r is denoted by $\beta_r(G)$ and $\beta(G) = \max_r \{\beta_r(G)\}$. The cardinality of color partition π is b. Subsets of edges of $E(G)$ corresponding to color partition π are represented by $E_j<\pi>$, $1 \le j \le b$. The subgraph of G induced by $E_j<\pi>$ is $H_j<\pi>$. To show a certain color partition graphically, we will merely draw the induced subgraphs H_j, $1 \le j \le b$. The vertex in $Y(\pi(G))$ corresponding to $H_j<\pi>$ is $y_j<\pi>$. The bus corresponding to $H_j<\pi>$ (or $E_j<\pi>$, or $y_j<\pi>$) is denoted by $B_j<\pi>$. The set of edges incident with vertex $y_j<\pi>$ of $\pi(G)$ is denoted by $J(E_j<\pi>)$ or $J(H_j<\pi>)$, $1 \le i \le b$. When there is no ambiguity as to the color partition π, we will always omit $<\pi>$ from the respective names.

Example 3.1: Consider the *IFG* (which we denote by G_4) shown in Figure 3.1. It contains 6 vertices and 8 edges of four different colors. Colors of the edges are represented by integers 1, 2, 3, and 4. It is clear that $\beta_1(G_4) = \beta_2(G_4) = \beta_3(G_4) = \beta_4(G_4) = \beta(G_4) = 2$. Figure 3.2 shows the subgraphs H_1 and H_2 induced by the subsets of a certain color

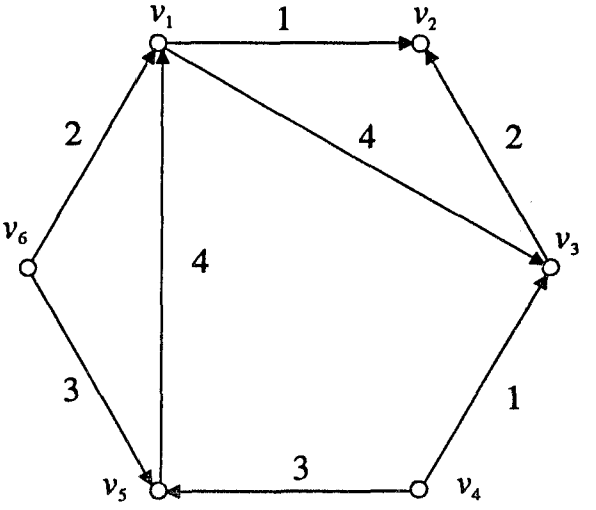


Figure 3.1: *IFG* G_4 with 4 colors 1, 2, 3, and 4.

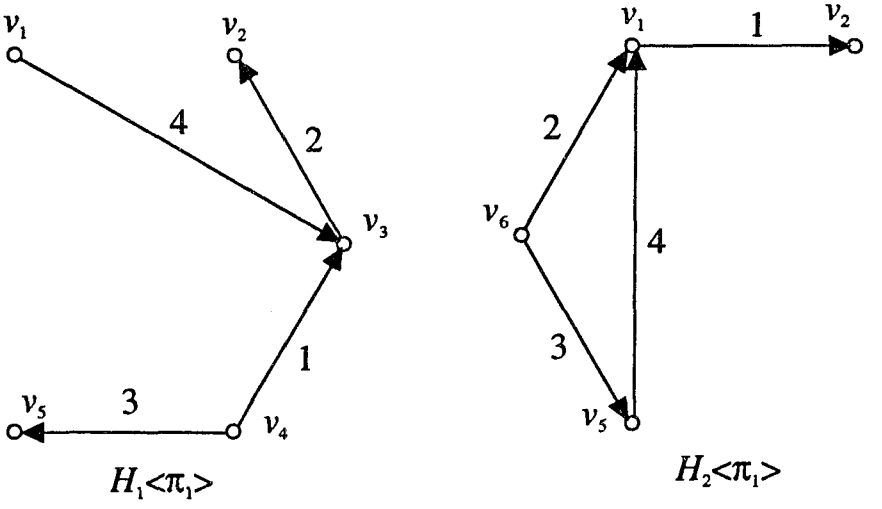


Figure 3.2: Cardinality two color partition π_1 of G_4.

partition, denoted by π_1, of G_4. In other words, Figure 3.2 shows the color partition π_1. The *MBS* corresponding to π_1 has two buses: one associated with H_1 and the other associated with H_2. Figure 3.3 shows the corresponding *MBG*, $\pi_1(G_4)$. Notice that y_1 is incident with 6 edges, that is, $|J(H_1)| = 6$. Therefore, bus B_1 is connected to 6 interfaces,

three of them being drivers and the other three being receivers. Similarly, bus B_2 is also connected to three receivers and three drivers.

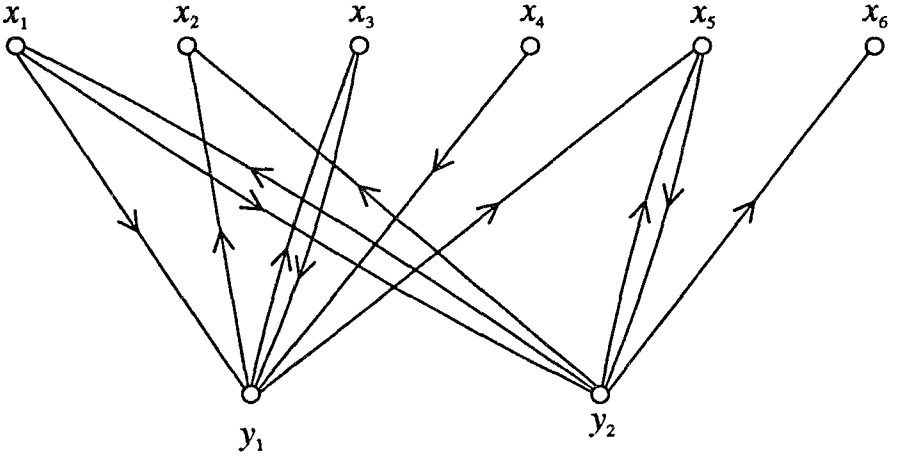


Figure 3.3: *MBG* $\pi_1(G_4)$

Next we show how broadcasting information can be incorporated in the *IFG* for the *MBG* construction.

3.3 Incorporating Broadcasting Operations

So far we did not make any assumptions on the identity of the data transfers denoted by the edges in the *IFG*. If two edges are of the same color, we allocate two buses to them in order to carry out both data transfers concurrently. Suppose that the source algorithm requires a certain computational node to send the same datum concurrently to different computational nodes. Also suppose that the processor assignment is such that the source node and destination nodes are assigned to distinct processors. This corresponds to edges of the same color originating from the same vertex of the *IFG*. Since the same datum is to be transferred, distinct buses are not required. If the connectivity allows, a single bus can broadcast the datum to all destination processors. A single source processor sending one data item to a set of target processors concurrently

will be called an *instance* of broadcasting. There may be more than one instance of broadcasting occurring concurrently. In such a case, more than one source processor is involved.

To include broadcasting information in the *IFG*, edge coloring should be modified as follows. Instead of a single color r, a 3-tuple of colors (r, τ, σ) will be assigned to each edge of the *IFG*. Let v_i be the source and v_{j_σ}, $1 \leq \sigma \leq h$, be the destinations of a certain broadcasting operation. Then edge (v_i, v_{j_σ}) will be assigned the color tuple $(r, 1, \sigma)$. Here, the first coordinate r represents the *primary color* that distinguishes the concurrent data transfer†. The second coordinate 1 represents the broadcasting instance number. If there is another broadcasting instance to be carried out concurrently, the edges belonging to that will have a first coordinate r and a second coordinate 2. The third coordinate σ represents the index of the data transfer involved with that particular instance of broadcasting. For generality in this section, we assume single target data transfers also as broadcasting operations, where the number of target processors is one. With slight abuse of notation, we represent an edge e with color tuple (r, τ, σ) by $e(r, \tau, \sigma)$.

The notion of color partition was introduced to capture the idea that, to achieve minimum communication time, concurrent data transfers are to be assigned to different buses. When broadcasting operations are involved, this is not necessarily true. Therefore, the definition of the color partition must also be modified.

---

†In Chapter 6, we will use the notation of primary color in a different context. Since the two issues are treated separately, no ambiguity will occur.

Definition 3.4′: A partition of the edge set of the *IFG* will be a *color partition* if and only if the two edges $e(r_1, \tau_1, \sigma_1)$ and $e(r_2, \tau_2, \sigma_2)$ belonging to the same subset of the partition satisfy one of the following conditions.

(1) $r_1 \neq r_2$ (data transfers are not concurrent); or

(2) $r_1 = r_2$ and $\tau_1 = \tau_2$. (they belong to the same instance of broadcasting)

When there are no broadcasting operations, $\beta_r(G)$ represent the number of edges with color r. This notation was introduced to count the number of buses needed to realize edges of color r. When broadcasting operations are involved, the number of edges with primary color r does not necessarily correspond to the number of buses needed for those data transfers. Therefore, we modify the definition of $\beta_r(G)$ as follows.

Definition 3.6′: $\beta_r(G) = |\{e(r_i, \tau_i, \sigma_i) : r_i = r \; \forall i;\; i \neq j \Rightarrow \tau_i \neq \tau_j \; \forall i,j\}|$

Therefore, by relabeling the colors of the edges and modifying the notion of the color partition and $\beta_r(G)$, we can incorporate broadcasting operations into the design process. It is to be understood that a color partition can always be found in a straightforward manner (whether broadcasting operations are involved or not) if we do not restrict the number of interfaces and buses involved. The difficult problem is to find an optimal color partition. As we show later, the optimal color partition problem with no broadcastings involved is a computationally difficult problem. Therefore, when we address the optimal color partition problem in the next section, we will not consider broadcasting operations. However, in Section 3.7, when we implement a heuristic algorithm for the general case, we will incorporate broadcasting operations.

3.4 Optimal Color Partition of an *IFG*

Not every color partition π results in an optimal *MBS*. This section addresses the problem of finding an optimal color partition, that is, a color partition which results in a minimal cost *MBS* which can run the algorithm in minimum time. Our objective is to find a minimum cost *MBS* which realizes a given source algorithm in minimum time. The cost of an *MBS* is the sum of the costs of its individual components; namely, the set of processors, the set of buses, and the set of interfaces. Since processor assignment is already done, the set of processors is specified. Therefore, we will exclude the number of processors from the cost function. Let G be an *IFG*. Then for any color partition π, $Y(\pi(G))$ and $E(\pi(G))$ represent the set of buses and the set of interfaces, respectively of an *MBS* realizing G. Therefore, our objective is to find a partition π such that $|Y(\pi(G))|$ and $|E(\pi(G))|$ are minimum. The following lemma gives a lower bound on $|Y(\pi(G))|$.

Lemma 3.2: Let π be a color partition of *IFG* G. Then $|Y(\pi(G))| \geq \beta(G)$.

Proof: Since π is a color partition, $\beta(G)$ edges of the same color must be spread in at least $\beta(G)$ subsets. Also there is a distinct vertex in $Y(\pi(G))$ for every subset of the partition. □

Unfortunately, we do not have such a straightforward method to find a lower bound on $|E(\pi(G))|$ for an arbitrary *IFG* G. Minimization of $|E(\pi(G))|$ is also called "pin minimization of buses" and was addressed on several occasions in the literature. In [141], a dynamic programming method was proposed for the pin minimization problem. In [106], the same problem was approached using switching theory. In [77], pin minimization was performed when the *IFG* (posed in a different form) has some specific features.

In this dissertation, we use a graph theoretic approach. Let π be a color partition of G. If two edges in a subset of π are directed towards (away from) the same vertex in G, data transfers corresponding to those two edges can share the same driver (receiver). In order to use this strategy in constructing a minimal cost *MBS*, we introduce the following definition.

Definition 3.7: Two edges in an *IFG* are said to be *head* (*tail*) *compatible* if they are directed towards (away from) the same vertex. Two edges are said to be 0-*compatible*, 1-*compatible*, or 2-*compatible* depending on whether they are neither head nor tail compatible, head or tail compatible, or both head and tail compatible†, respectively.

In order to minimize the number of interfaces, our criterion is to include edges in subsets such that the number of edges in each subset which are head and/or tail compatible are maximized. In general, given an arbitrary *IFG* G, one cannot find a single color partition π which minimizes both $|Y(\pi(G))|$ and $|E(\pi(G))|$ at the same time. This fact is illustrated in the following example.

Example 3.2: Consider the *IFG* G_4 shown in Figure 3.1. Clearly, $\beta(G_4) = 2$. The number of edges in the *MBG* $\pi_1(G_4)$ shown in Figure 3.3 is 12. One can easily be convinced that there does not exist any color partition of G_4 of cardinality 2 which results in an *MBG* with less than 12 edges. Figure 3.4 shows a color partition π_2 of G_4 whose cardinality is 3. The corresponding *MBG* $\pi_2(G_4)$ has only 11 edges as shown in Figure 3.5. Therefore, there is no single color partition π which minimizes both $|E(\pi(G_4))|$ and $|Y(\pi(G_4))|$ simultaneously.

---

†Two edges can be 2-compatible iff they are parallel edges in the *IFG*.

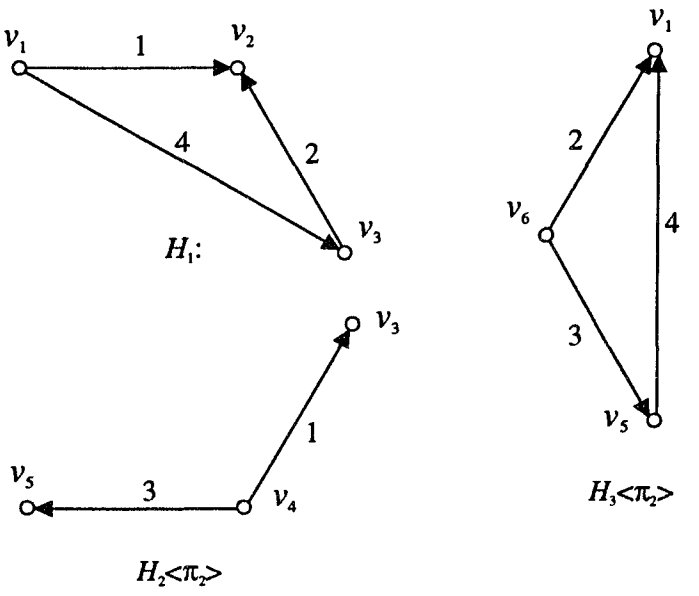


Figure 3.4: Cardinality three color partition π_2 of G_4.

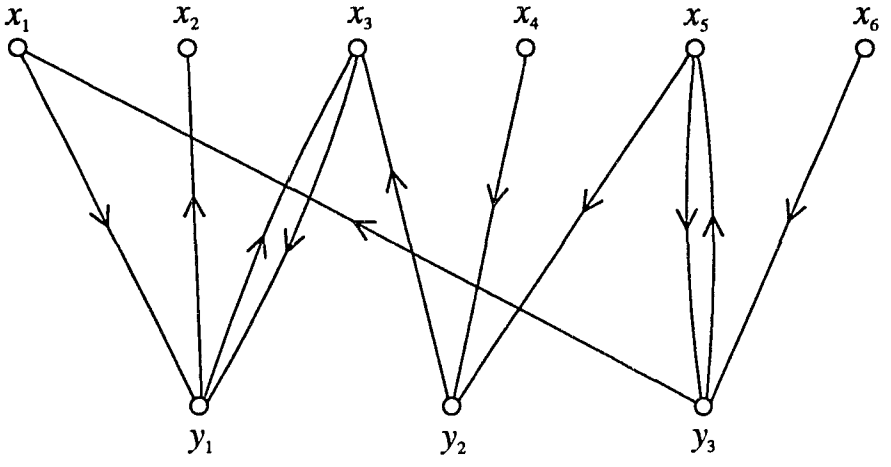


Figure 3.5: *MBG* $\pi_2(G_4)$

For the special case, when the *IFG* has only two colors, there always exists a multiple bus system with minimum number of buses and minimum number of interfaces. We will address this issue in Section 3.6. Since, in general, one cannot find an *MBS* such

that both the number of buses and the number of interfaces are minimum (for an *IFG* with arbitrary number of colors), the optimization function to be used for the minimization should contain terms to represent relative costs of buses and interfaces. Therefore, we define $\kappa_3*|Y(\pi(G))| + \kappa_4*|E(\pi(G))|$ as our optimization function, where constants κ_3 and κ_4 are chosen to reflect the relative costs of buses and interfaces, respectively.

Definition 3.8: For the *IFG G*, a color partition π with a minimum value of $\kappa_3*|Y(\pi(G))| + \kappa_4*|E(\pi(G))|$ will be called an *optimal color partition.* The corresponding bus assignment will be called an *optimal bus assignment.*

Now our design problem is to find a color partition π of *IFG G* such that $\kappa_3*|Y(\pi(G))| + \kappa_4*|E(\pi(G))|$ is minimum. This problem, although posed in a different form, has been addressed in the literature [106], [141]. But none of these works have addressed the computational complexity of the problem. In the next section, we prove that the optimal color partitioning problem is *NP*-Hard.

3.5 Computational Complexity of the Optimal Color Partitioning problem

We show that the problem of finding an optimal color partition π of a given *IFG* is a computationally difficult problem. To show this, we convert the optimal partitioning problem into a decision problem which is called Color Edge Partitioning (*CEP*) problem. We formally define the *CEP* problem as follows.

Color Edge Partitioning problem:

INSTANCE: An *IFG G* and three constants κ_3, κ_4 and K.

QUESTION: Can we find a color partition π of G such that

$$\kappa_3*|Y(\pi(G))| + \kappa_4*|E(\pi(G))| \leq K?.$$

Theorem 3.1: The *CEP* problem is NP-Complete.

Proof: It is straightforward to construct a non deterministic algorithm to solve *CEP* in polynomial time. Thus *CEP* belongs to class NP. To show that the *CEP* problem is NP-Hard, we transform an instance of the Three Dimensional Matching problem (3*DM*) into an instance of *CEP* and show that the 3*DM* instance has a matching if and only if the corresponding *CEP* instance has a solution. The 3*DM* problem [53] was already stated in Section 2.3. Let X, Y, Z be the three sets and $M \subseteq X \times Y \times Z$ be the collection of three element subsets. Let the cardinality of each of X, Y, Z be p. Also let the cardinality of M be q.

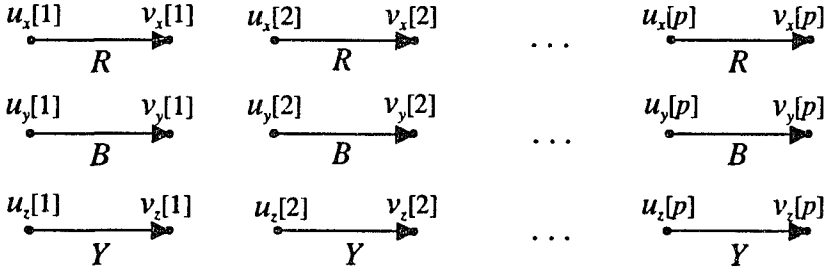


Figure 3.6: Type 1 edges of G

From the 3*DM* instance we will construct an instance G of *CEP*. Let x, y, and z be the representative elements of the sets X, Y, and Z, respectively. G will have four types of edges. For each element w_i in $X \cup Y \cup Z$, form a directed edge $(u_w[i], v_w[i])$ of type 1 (along with its end vertices) as shown in Figure 3.6. Here, w can be x, y or z, and i can range from 1 through p. Type 1 edges corresponding to the elements in set X, set Y and, set Z will be of colors Red, Blue and Yellow, respectively. Letters R, B, and Y in the figure stand for the colors Red, Blue, and Yellow, respectively. For each element $m_j = (x_{j(1)}, y_{j(2)}, z_{j(3)})$ in M, form seven vertices $a[j]$, $b[j]$, $c[j]$, $d[j]$, $e[j]$, $f[j]$ and $g[j]$ and nine

edges as shown in Figure 3.7. The types and colors of the respective edges are also shown in Figure 3.7. Since the resulting graph G is directed, edge colored, and connected, it represents an *IFG*. After the construction, G will have, from each color, p edges of type 1, q edges of type 2, q edges of type 3, and q edges of type 4. Therefore, the constructed *CEP* instance G has $6p + 7q$ vertices and $3(p + 3q)$ edges, $p + 3q$ edges from each color. Also, $\beta_R(G) = \beta_B(G) = \beta_Y(G) = \beta(G) = p + 3q$. We set the value of K equal to $(\kappa_3 + 4\kappa_4)(p + 3q)$.

More notation is needed here. A set of three edges is said to form an *edge triplet* (or simply a *triplet*) if one edge in the set is head compatible with another edge and tail compatible with the third edge in the set. In the proof, we try to partition the edge set of G into triplets.

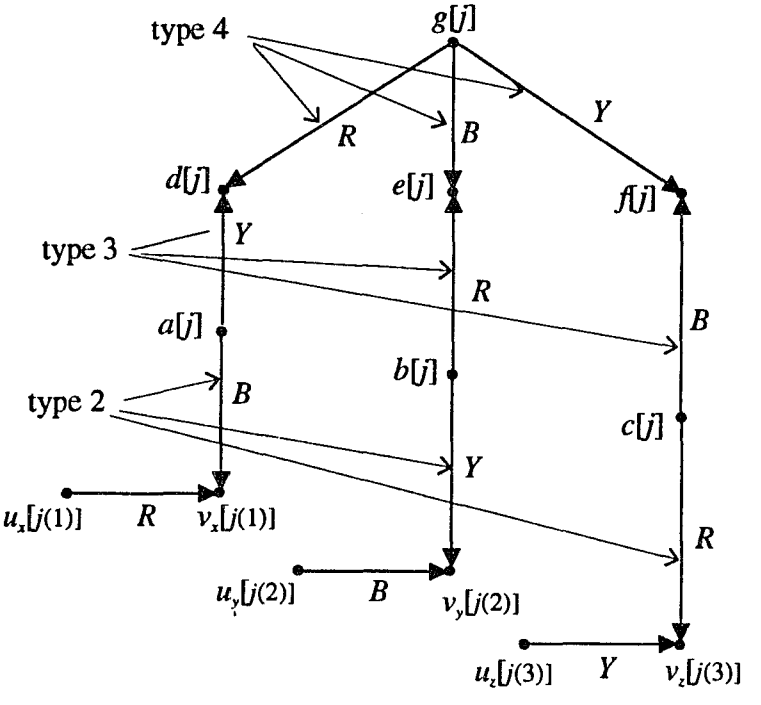


Figure 3.7: Types 2, 3, and 4 edges of G corresponding to an element $m_j \in M$.

First we show that G has the required partition if $3DM$ has a matching. Let $M' \subseteq M$ be a matching for the $3DM$ instance. Consider the following color partition. For each element $m \in M'$, construct four edge triplets as shown in Figure 3.8. This will cover all the edges in G of type 1 (that is, p edges from each color). This will also cover p edges from each color belonging to each of the types 2, 3 and 4. For each element $m \notin M'$, construct three triplets as shown in Figure 3.9. This will cover all the remaining edges in G of types 2, 3 and 4. The partition has $4p + 3(q - p)$ subsets (triplets), that is, $|Y(\pi(G))| = 4p + 3(q - p) = p + 3q$. Each subset contributes 4 edges to $E(\pi(G))$. Therefore, $\kappa_3 * |Y(\pi(G))| + \kappa_4 * |E(\pi(G))| = (\kappa_3 + 4\kappa_4)(p + 3q) = K$. Hence we have a solution to the *CEP* instance.

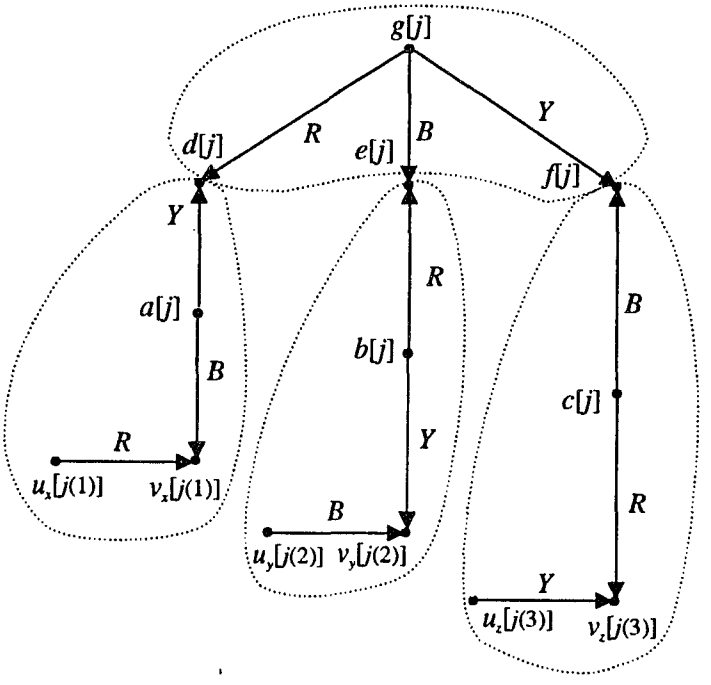


Figure 3.8: Triplets corresponding to an element $m \in M'$.

Next we show that $3DM$ instance has a matching if G has a solution. Let π be a color partition of G such that $\kappa_3 * |Y(\pi(G))| + \kappa_4 * |E(\pi(G))| = K = (\kappa_3 + 4\kappa_4)(p + 3q)$. Let the cardinality of π be b. Each subset of the color partition π can have at most 3 edges. Since there are no triangles in the underlying undirected graph of G, the induced subgraph of each subset underlies a tree. Therefore, the number of vertices of the subgraph induced by the subset E_i of vertices is at least $|E_i| + 1$, $1 \leq i \leq b$. Thus $|J(E_i)| \geq |J(E_i)| + 1$. Hence, by Lemma 3.1, $|E(\pi(G))| \geq \sum_{i=1}^{b} (|E_i| + 1) = \sum_{i=1}^{b} |E_i| + b = |E(G)| + b = 3(p + 3q) + b$. From Lemma 3.2, $|Y(\pi(G))| = b \geq \beta(G) = p + 3q$. Therefore, $|E(\pi(G))| \geq 4(p + 3q)$. Since $|Y(\pi(G))| \geq p + 3q$, we have, $\kappa_3 * |Y(\pi(G))| + \kappa_4 * |E(\pi(G))| \geq \kappa_3(p + 3q) + 4\kappa_4(p + 3q) = (\kappa_3 + 4\kappa_4)(p + 3q)$. Since $\kappa_3 * |Y(\pi(G))| + \kappa_4 * |E(\pi(G))| = (\kappa_3 + 4\kappa_4)(p + 3q)$ by hypothesis, it follows that $|E(\pi(G))| \geq 4(p + 3q)$ and that $|Y(\pi(G))| = p + 3q$. Hence each subset is a triplet.

From the construction of G it is clear that the only way to include a type 1 edge in a triplet is to use it with the type 2 edge adjacent with it and the type 3 edge adjacent with the type 2 edge. Thus p pairs of adjacent type 2 and type 3 edges from each color will be consumed to cover all the edges of type 1. The remaining $(q - p)$ pairs of adjacent type 2 and type 3 edges from each color must be combined with $(q - p)$ edges of type 4 from each color. After using all the edges of type 2 and type 3, p edges of type 4 from each color will remain. If the edge set $E(G)$ has the required partition, we must be able to form p triplets from those remaining type 4 edges. Each such triplet corresponds to an element in M', thereby creating M'. □

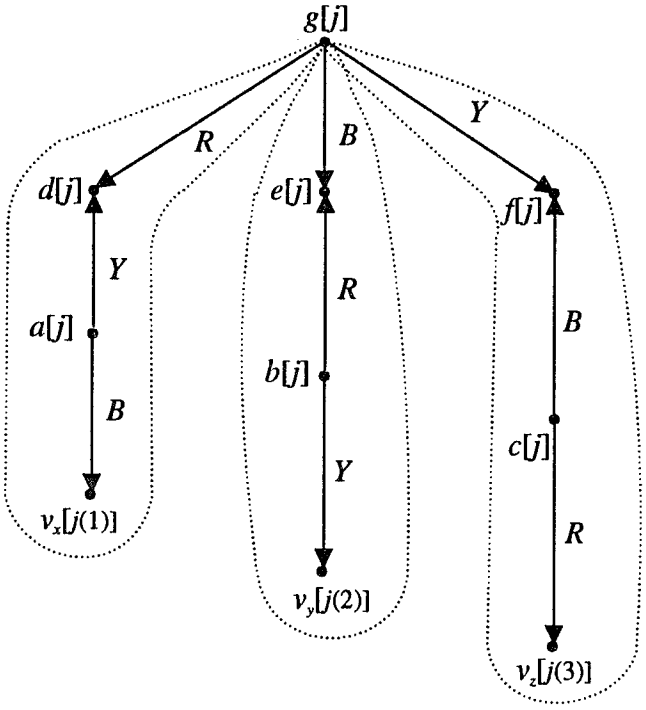


Figure 3.9: Triplets corresponding to an element $m \notin M'$

If we can solve the optimization problem in polynomial time, then we can also solve the decision problem in polynomial time. But, since the decision problem is NP Complete, it is very unlikely that we find a polynomial algorithm to solve the decision problem and hence the optimization problem. For the above proof, we constructed an *IFG* with three colors. Therefore, the optimal bus assignment for an *IFG* with greater than or equal to three colors is an *NP*-Hard problem. We later (Section 3.6) show that the optimal bus assignment problem for a two color *IFG* can be solved in polynomial time.

Computational complexity of some related problems have been established in the literature. For example, the problem of designing a bounded degree graph to map maximum number of edges from a `problem graph' is shown to be computationally

difficult. But the computational complexity of the problem of mapping a `problem graph' into a target graph (which is equivalent to the graph isomorphism problem) has not been established yet [23], [53].

As we have mentioned in Section 2.3, once a problem is proven to be *NP*-Hard, we usually have two different approaches to solving the problem. In the first approach we study how the problem can be polynomially solved in special cases. In the second approach, we use a polynomial time algorithm (probably heuristic) such that the solution is not necessarily optimal but does not deviate from the optimal solution by more than a certain amount. As for the first approach, it is practically impossible to generalize the properties an *IFG* should possess in order to be partitionable in polynomial time. In this dissertation, we will consider two important properties of an *IFG* each of which guarantees polynomial time solution to the optimal color partition problem. The first property, which is addressed in the next section, is that the *IFG* has only two colors. The second property, which will be addressed in Chapter 4, is that the *IFG* is vertex symmetric. As for the second approach, in this chapter, we devise a heuristic algorithm to solve the general problem. The heuristic algorithm is based on the solution to the two color case.

3.6 Optimal Color Partition on an *IFG* with only Two Colors

If the *IFG* has only two colors, an optimal color partition can be found in polynomial time. We analyze the two color case for three major reasons. First, it answers the combinatorial question: what is the maximum number of colors an *IFG* can have so that the optimal color partition problem is solvable in polynomial time? Second, it provides a polynomial time algorithm to construct an optimal *MBS* realizing any two

interconnection functions such as the shuffle and exchange functions [134]. Third, it provides a guideline for a heuristic algorithm for solving the general color partition problem. We assume that the two colors of the *IFG* are denoted by 1 and 2. When the *IFG* has only two colors, as the following theorem shows, both $|Y(\pi(G))|$ and $|E(\pi(G))|$ can be minimized at the same time.

Theorem 3.2: Let G be an *IFG* with two colors. Then there always exists a color partition π of G which minimizes both $|Y(\pi(G))|$ and $|E(\pi(G))|$ simultaneously.

Proof: Let π_1 be a color partition of G such that $|E(\pi_1(G))| \leq |E(\pi(G))|$, for all color partitions π. If $|Y(\pi_1(G))| = \beta(G)$, then π_1 is the required partition. Otherwise (that is, if $|Y(\pi_1(G))| > \beta(G)$), there should exist a subset $E_1<\pi_1>$ of partition π_1 which does not contain a color 1 edge. This is because, if every subset of π_1 contains a color 1 edge, then the cardinality of partition π_1, $|Y(\pi_1(G))| = \beta_1(G) \leq \beta(G)$. Similarly, there should exist a subset $E_2<\pi_1>$ of the partition π_1 which does not contain a color 2 edge. Let π_2 be the color partition of G obtained from π_1 by replacing the two subsets $E_1<\pi_1>$ and $E_2<\pi_1>$ by the single subset $E_1<\pi_2> = E_1<\pi_1> \cup E_2<\pi_1>$. This is possible because $E_1<\pi_1> \cup E_2<\pi_1>$ contains edges of distinct colors. There are two edges in $E(\pi_1(G))$ associated with each of the subsets $E_1<\pi_1>$ and $E_2<\pi_1>$ (that is, $|J(E_1<\pi_1>)| = |J(E_2<\pi_1>)| = 2$). There are at most four edges in $J(E_1<\pi_2>)$. Hence, from Lemma 3.1, $|E(\pi_2(G))| \leq |E(\pi_1(G))|$. Since $|E(\pi_1(G))|$ is minimum by hypothesis, we have $|E(\pi_2(G))| = |E(\pi_1(G))|$. Furthermore, $|Y(\pi_2(G))| = |Y(\pi_1(G))| - 1$. If $|Y(\pi_2(G))| = \beta(G)$, then the required partition is π_2. Otherwise, repeat the above procedure to construct another color partition π_3 of G such that $|E(\pi_3(G))| = |E(\pi_2(G))|$, and $|Y(\pi_3(G))| = |Y(\pi_2(G))| - 1$. Therefore, by repeatedly

applying the above procedure, we can find a color partition π such that $|Y(\pi(G))| = \beta(G)$ and $|E(\pi(G))| = |E(\pi_1(G))|$. □

In order to perform an optimal color partition of a two color *IFG*, we provide the following notation.

Definition 3.9: Let G be an *IFG* with two colors. Then $L(G)$ is a weighted bipartite graph defined as follows. For each edge e_i^1 of color 1 in G, there exists a unique vertex v_i^1 in the first partite set $X(L(G))$, and vice versa. Similarly, for each edge e_j^2 of color 2 in G, there exists a unique vertex v_j^2 in the second partite set $Y(L(G))$, and vice versa. There is an (undirected) edge between v_i^1 and v_j^2 in $L(G)$ if and only if their corresponding edges e_i^1 and e_j^2 are either head or tail compatible (see Definition 3.7) in G. An edge (v_i^1, v_j^2) of $L(G)$ will be assigned a weight $w(v_i^1, v_j^2) = z$ if and only if e_i^1 and e_j^2 are z-compatible. If v_i^1 and v_j^2 are not adjacent in $L(G)$, then $w(v_i^1, v_j^2) = 0$.

Clearly, a color partition of G is equivalent to a vertex partition of $L(G)$ such that each partition contains at most one vertex from any partite set. In this dissertation, unless otherwise stated, a vertex partition of a bipartite graph is always meant to be done such that no subset contains more than one vertex from the same partite set.

Theorem 3.3: Let G be a two color *IFG* and π be a color partition of G. Let W^π be the sum of the weights of the edges induced by the vertex partition of $L(G)$ corresponding to color partition π. Then $|E(\pi(G))| = 2|E(G)| - W^\pi$.

Proof: Let E_j, $1 \le j \le b$, be a subset of edges of G due to color partition π, where b is the cardinality of color partition π. Define w_j as follows. If E_j contains two edges e_1 and e_2, then $w_j = w(v_1, v_2)$, where v_1 and v_2 are the vertices of $L(G)$ corresponding to edges e_1 and e_2, respectively. If E_j contains only one edge, then $w_j = 0$. It is clear that W^π =

$\sum_{j=1}^{b} w_j$. We claim that $|J(E_j)| = 2|E_j| - w_j$, for $1 \leq j \leq b$. We prove the claim by considering all possible cases.

Case 1: E_j has only one edge:

A driver and a receiver must be connected to bus B_j for the corresponding data transfer, that is, $|J(E_j)| = 2$. Since $w_j = 0$, it follows that, $2|E_j| - w_j = 2 - 0 = 2$.

Case 2: E_j has two edges e_1 and e_2 which are not compatible:

Two drivers and two receivers are needed for corresponding data transfers, that is, $|J(E_j)| = 4$. Since $w_j = 0$, it follows that, $2|E_j| - w_j = 4 - 0 = 4$.

Case 3: Two edges e_1 and e_2 are head compatible:

Only one receiver is needed for both data transfers, that is, $|J(E_j)| = 3$. Since $w_j = 1$, it follows that, $2|E_j| - w_j = 4 - 1 = 3$.

Case 4: Two edges e_1 and e_2 are tail compatible:

Only one driver is needed for corresponding data transfers, that is $|J(E_j)| = 3$. Similar to Case 3, $2|E_j| - w_j = 3$.

Case 5: Two edges e_1 and e_2 are both head and tail compatible:

Only one receiver and one driver are needed for corresponding data transfers, that is $|J(E_j)| = 2$. Since $w_j = 2$, it follows that, $2|E_j| - w_j = 4 - 2 = 2$.

Hence, the claim is true and therefore, $\sum_{j=1}^{b} |J(E_j)| = 2\sum_{j=1}^{b} |E_j| - \sum_{j=1}^{b} w_j = 2|E(G)| - W^{\pi}$. Since $\sum_{j=1}^{b} |J(E_j)| = |E(\pi(G))|$ (by Lemma 3.1), the theorem follows. □

Definition 3.10: A vertex partition of $L(G)$ is said to be an *optimal vertex partition*, if the sum of the weights of the edges induced by the subsets of the partition is maximum.

According to Theorem 3.3, the problem of finding an optimal color partition of a two color *IFG* G is equivalent to that of finding an optimal vertex partition of the

bipartite graph $L(G)$. To find such a vertex partition, some terminology from graph theory is necessary.

For a given graph G, a maximum matching can be found in polynomial time [30], [56]. In this paper we consider matchings in bipartite graphs only. For our purpose, what we need to maximize is not the number of edges but the sum of the weights of the edges in a matching. To distinguish between these two concepts, we introduce the following definition.

Definition 3.11: If the sum of the weights of a matching in $L(G)$ is maximum, that matching will be called a *weighted maximum matching*.

If a weighted maximum matching M of $L(G)$ is known, it is straightforward to construct an optimal vertex partition of $L(G)$. Simply construct a vertex partition which contains the matching M. Thus the problem of finding an optimal color partition of a two color *IFG* G is reduced to that of finding a weighted maximum matching in $L(G)$. We will next show how to find a weighted maximum matching in $L(G)$. In the following, $W(M)$ represents the sum of the weights of the edges in matching M.

Lemma 3.3: Let G be a two color *IFG*. Let M_2 be the set of edges in $L(G)$ whose weights are 2. Then M_2 is a matching in $L(G)$.

Proof: We need to prove that the edges in M_2 are pairwise independent. Suppose there are two edges in M_2 with a common end vertex v_i. Assume that v_i is in the first partite set. Therefore edge e_i (which is of color 1) of G must be parallel with two distinct edges of color 2. Then those two edges of color 2 must be parallel to one another. Since this is not possible, no two edges in M_2 have a common vertex. □

Theorem 3.4: Let G be an *IFG* with two colors. Then there is a weighted maximum matching in $L(G)$ which contains all edges of $L(G)$ with weight 2.

Proof: Let M be a weighted maximum matching in $L(G)$. Let M_2 be the set of edges of $L(G)$ of weight 2. If $M_2 \subseteq M$, then we have nothing to prove. Otherwise, let $e \in M_2$ be such that $e \notin M$. Construct a new matching M' from M by removing the edges adjacent with e and inserting e. From Lemma 3.3, no edge adjacent with e can have weight 2. Also, there can be at most two edges adjacent with e which are in M. Therefore, $W(M') \geq W(M)$. Since M is assumed to be a weighted maximum matching, it follows that $W(M') = W(M)$. If $M_2 \subseteq M'$, we have the required matching. Otherwise, repeat the above procedure until we get a weighted maximum matching containing M_2. □

Theorem 3.5: Let M_2 be the set of edges of $L(G)$ of weight 2. Let $L_1(G)$ be the graph obtained from $L(G)$ by deleting the end vertices of all the edges in M_2. (Deleting a vertex will automatically delete all the edges adjacent with it). Let M_1 be a maximum matching in $L_1(G)$. Then $M_1 \cup M_2$ is a weighted maximum matching in $L(G)$.

Proof: Suppose there exists another matching M in $L(G)$ such that $W(M) > W(M_1 \cup M_2)$. According to Theorem 3.4, we can assume that M contains M_2. Let $M = M_1' \cup M_2$. Therefore, $W(M) > W(M_1 \cup M_2)$ implies $M_1' > M_1$. None of the edges in M_1' is adjacent with end vertices of any of the edges in M_2. Because, otherwise, $M_1' \cup M_2$ will not be a matching in $L(G)$. Therefore, $M_1' > M_1$ implies the existence of a matching in $L_1(G)$ whose cardinality is greater than M_1. This is not possible since M_1 is a maximum matching in $L_1(G)$. Thus, $M_1 \cup M_2$ is a weighted maximum matching in $L(G)$. □

The above theorem provides us with a method to find an optimal color partition of a two color *IFG*. First, construct $L(G)$ from G. Then find a maximum matching M_1 in

$L_1(G)$, where $L_1(G)$ is obtained from $L(G)$ by deleting all edges of weight 2. If M_2 is the set of deleted edges, then $M = M_1 \cup M_2$ is a weighted maximum matching in $L(G)$. The vertex partition of $L(G)$ associated with matching M corresponds to an optimal color partition of G. In [67], an algorithm is presented to find a maximum matching in a bipartite graph with n vertices in $O(n^{2.5})$ time. Utilizing that algorithm, we can find an optimal color partition of a two color *IFG* G in $O(|E(G)|^{2.5})$ time. Next we will develop a heuristic algorithm for the general case based on the two color case. The algorithm will give us a nearly optimal color partition of a general *IFG*.

3.7 Heuristic Algorithm for the General Case

In developing a heuristic algorithm for the general case, it is assumed that the *IFG* under consideration contains broadcasting information. We base the algorithm on the following two assumptions.

(1) For optimality, data transfers associated with the same broadcasting instance must be assigned to the same bus.

(2) The local optimization is not very far away from the global optimization.

In Section 3.3, we showed how broadcasting information can be included in an *IFG*. But we did not illustrate how to make use of these for finding a color partition. Therefore, while developing the heuristic algorithm, we will clearly show how broadcasting information is incorporated into the color partition.

Since broadcasting operations are considered, each edge is assigned a 3-tuple of colors instead of a single color. Also, the meaning of the color partition is accordingly modified (see Definition 3.4′). To impose the first assumption on which our heuristic algorithm is based, if $e(r_i, \tau_i, \sigma_i)$, $\forall i$, are the edges of the given *IFG*, then the data

transfers corresponding to the edges in set $\{e(r_i, \tau_i, \sigma_i) : \forall i,j \;\; r_i = r_j \text{ and } \tau_i = \tau_j\}$ must be assigned to the same bus. In other words, the color partition must be performed such that the edges in set $\{e(r_i, \tau_i, \sigma_i) : \forall i,j \;\; r_i = r_j \text{ and } \tau_i = \tau_j\}$ belong to the same subset of the partition.

For simplicity, we will denote the set of edges in G with the first coordinate r (primary color) and the second coordinate τ by $q(G, r, \tau)$. That is, $q(G, r, \tau) = \{e(r_i, \tau_i, \sigma_i) : r = r_i, \tau = \tau_i\}$. In the last section, when there were no broadcasting operations involved, we constructed a bipartite graph $L(G)$ from the *IFG* G such that each vertex of $L(G)$ represents a unique edge of G. Here we will construct a bipartite graph such that each of its vertices represents a set of edges $q(G, r, \tau)$ for some r and τ. The vertex corresponding to $q(G, r, \tau)$ will be denoted by $v(q(G, r, \tau))$. Let $\{q(G, r, \tau)\}_\tau$† be denoted by $Q(G, r)$. Similarly, let $\{v(q(G, r, \tau))\}_\tau$ be denoted by $Q^v(G, r)$. Clearly, the cardinality of $Q(G, r)$, as well as, that of $Q^v(G, r)$, is $\beta_r(G)$ (see Definition 3.6′).

For the two color case, we constructed only one bipartite graph $L(G)$. For the general case, we will construct a set of bipartite graphs $L(G)^r$, $1 \le r \le c - 1$, where c is the number of primary colors in G. As will be shown in the algorithm, the partite sets of $L(G)^r$, $1 \le r \le c - 1$, will be constructed recursively. Determination of the edges and their weights of $L(G)^r$, $1 \le r \le c - 1$, are based on the following discussion.

When there is no broadcasting involved, we established an edge between $v_i^1 \in X(L(G))$ and $v_j^2 \in Y(L(G))$ iff the two edges e_i^1 and e_j^2 are compatible, where v_i^1 and v_j^2 are the representative vertices of the edges e_i^1 and e_j^2, respectively. With broadcasting operations involved, to establish the adjacency between vertices of $X(L(G)^r)$ and $Y(L(G)^r)$,

---

† $\{f(\tau)\}_\tau$ represents the set of elements $f(\tau)$ by taking all distinct values of τ.

we need to extend the definition of (head or tail) compatibility between the edges of an *IFG*.

Definition 3.12: Let q_1 and q_2 be two non-empty disjoint subsets of edges of an *IFG*. Then q_1 and q_2 are said to be *z-compatible* if $|J(q_1)| + |J(q_2)| - |J(q_1 \cup q_2)| = z$. Also, q_1 and q_2 are said to be *compatible* if they are z-compatible and $z > 0$.

Notice that the above definition explicitly states: if the number of interfaces saved by assigning the edges in q_1 and q_2 to the same bus is z, then the subsets q_1 and q_2 are z-compatible. With the extended definition of the term "compatible", we can determine the edges and their weights in $L(G)^r$. Note that each vertex of $L(G)^r$ corresponds to a set of edges from G. Let $v(q_1)$ and $v(q_2)$ be two vertices of $L(G)^r$. Then there exists an edge of weight z between $v(q_1)$ and $v(q_2)$ of $L(G)^r$ if q_1 and q_2 are z-compatible in G. The weights of the edges in $L(G)^r$ are not bounded above by 2 in contrast with the case when broadcasting operations were not considered. Therefore, we need to find a general weighted maximum matching for $L(G)^r$. For a bipartite graph, such a matching can be found in $O(n^3)$ time [56].

Now, we can present the heuristic algorithm. For notational convenience, we will use β_r instead of $\beta_r(G)$, $1 \le r \le c$. Also, without loss of generality, we assume that $\beta_1 \ge \beta_2 \ge ... \ge \beta_c$. The outline of the algorithm is as follows. Initially we form the subsets E_1, E_2, ..., E_{β_1} such that E_j contains edges of primary color 1 belonging to the j^{th} broadcasting instance. Then we construct $L(G)^1$, where $X(L(G)^1) = \{v(E_j) : 1 \le j \le \beta_1\}$ and $Y(L(G)^1) = Q^v(G, 2)$. The edge set of $L(G)^1$ is determined by the compatibility among the elements in $\{E_j : 1 \le j \le \beta_1\}$ and those in $Q(G, 2)$. Based on the maximum weighted matching of $L(G)^1$, we choose the edge set belonging to the best instance of broadcasting with primary

color 2 which can be combined with each E_j. This operation will update each E_j. Next, we construct $L(G)^2$, where $X(L(G)^2) = \{v(E_j) : 1 \le j \le \beta_1\}$ and $Y(L(G)^2) = Q^v(G, 3)$ and edges are determined by the compatibility among the elements in $\{E_j : 1 \le j \le \beta_1\}$ and those in $Q(G, 3)$. Then we find the maximum weighted matching in $L(G)^2$ and update the values of E_j, $1 \le j \le \beta_1$, accordingly. We repeat this procedure until all edges of G are exhausted. The final list E_j, $1 \le j \le \beta_1$, is the resulting color partition of G.

Algorithm 3.1

INPUT: An *IFG* G with c colors including broadcasting information.
OUTPUT: A color partition $\{E_1, E_2, ..., E_{\beta_1}\}$ of G.

```
begin
    for j = 1 to β1 do
    Ej = q(G, 1, τ);
    for r = 1 to (c − 1) do
    begin
        begin_construct L(G)^r
        X(L(G)^r) = {v(E1), v(E2), ..., v(Eβ1)}
        Y(L(G)^r) = Q^v(G, r + 1)
        E(L(G)^r) = {(x, y) : x ∈ X(L(G)^r), y ∈ Y(L(G)^r),
                        x and y are compatible}
        for all (x, y) ∈ E(L(G)^r)
        if x and y are z-compatible then w(x, y) = z;
        end_construct;
        find a maximum weighted matching Mr in L(G)^r;
        for j = 1 to β1 do
        if (v(Ej), v(q(G, r + 1, τ))) ∈ Mr
        then Ej = Ej ∪ {q(G, r + 1, τ)};
    end;
end.
```

Example 3.3: Suppose that we want to find an optimal color partition of the four color *IFG*, denoted by G_5, shown in Figure 3.10. The same line style is used to identify edges with the same primary color. For example, edges $e(1, 1, 1)$, $e(1, 1, 2)$, $e(1, 2, 1)$, $e(1, 2, 2)$, and $e(1, 3, 1)$, which have the same primary color 1, are denoted by solid lines. Edges

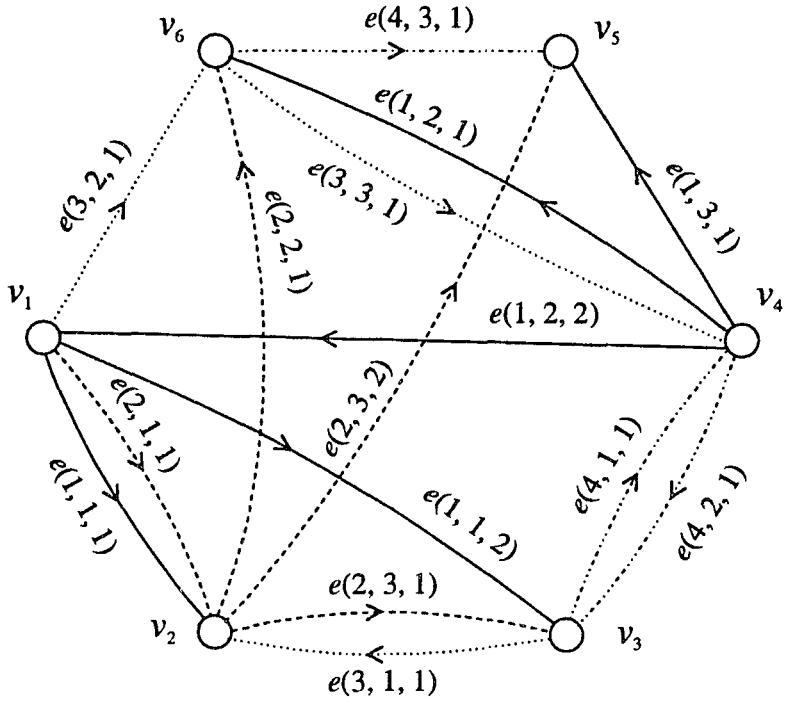


Figure 3.10: *IFG* G_5

$e(1, 1, 1)$ and $e(1, 1, 2)$ of G_5 belong to a single instance of broadcasting. Therefore, we set $E_1 = \{e(1, 1, 1), e(1, 1, 2)\}$. Also, edges $e(1, 2, 1)$ and $e(1, 2, 2)$ correspond to another concurrent broadcasting instance. Thus, we set $E_2 = \{(e(1, 2, 1), e(1, 2, 2)\}$. Edge $e(1, 3, 1)$ does not belong to a broadcasting operation. However, as we have stated earlier, we consider it as a trivial broadcasting operation of its own and set $E_3 = \{(1, 3, 1)\}$. The first partite set of $L(G_5)^1$ is $\{v(E_1), v(E_2), v(E_3)\}$. The second partite set is $Q^v(G_5, 2) = \{v(q(2, 1))^\dagger, v(q(2, 2)), v(q(2, 3))\}$, where $q(2, 1) = \{e(2, 1, 1)\}$, $q(2, 2) = \{e(2, 2, 1)\}$, and $q(2, 3) = \{(e(2, 3, 1), e(2, 3, 2)\}$. Bipartite graph $L(G_5)^1$ is shown in Figure 3.11. Since $E_1 = \{e(1, 1, 1), e(1, 1, 2)\}$ and $q(2, 1) = \{e(2, 1, 1)\}$ are 2-compatible, edge $(v(E_1), v(q(2, 1)))$

---

†Here we omit G_5 from the notation $q(G, r, \tau)$ for simplicity.

of $L(G_5)^1$ is assigned weight 2. Similarly, other edges of $L(G_5)^1$ are assigned their respective weights (see Figure 3.11).

$q(2, 1) = \{e(2, 1, 1)\}$
$q(2, 2) = \{e(2, 2, 1)\}$
$q(2, 3) = \{e(2, 3, 1), e(2, 3, 2)\}$

$E_1 = \{e(1, 1, 1), e(1, 1, 2)\}$
$E_2 = \{e(1, 2, 1), e(1, 2, 2)\}$
$E_3 = \{e(1, 3, 1)\}$

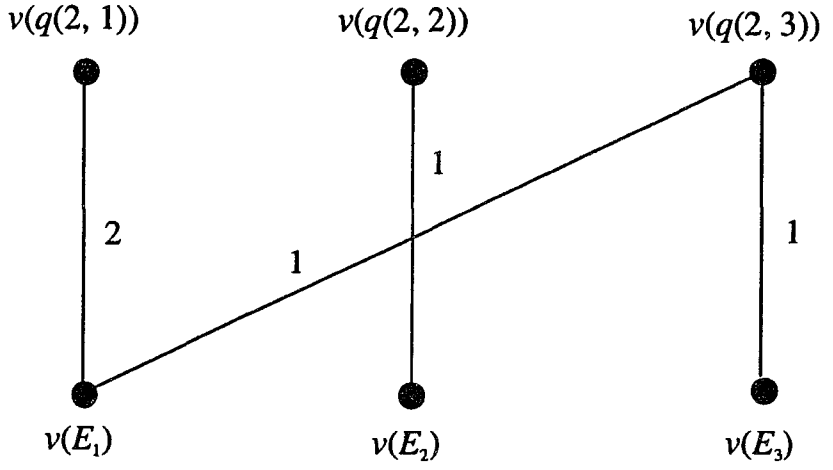


Figure 3.11: $L(G_5)^1$.

Clearly, the maximum weighted matching in $L(G_5)^1$ is $\{(v(E_1), v(q(2, 1))), (v(E_2), v(q(2, 2))), (v(E_3), v(q(2, 3)))$. Therefore, the updated values of E_j are:

$E_1 = \{e(1, 1, 1), e(1, 1, 2), e(2, 1, 1)\}$,

$E_2 = \{e(1, 2, 1), e(1, 2, 2), e(2, 2, 1)\}$,

$E_3 = \{e(1, 3, 1), e(2, 3, 1), e(2, 3, 2)\}$.

Bipartite graph $L(G_5)^2$ is shown in Figure 3.12. Again, the first partite set is $\{v(E_1), v(E_2), v(E_3)\}$ and the second partite set is $Q^v(G_5, 3)$. The corresponding weighted maximum matching is $\{(v(E_1), v(q(3, 1))), v(E_2), v(q(3, 2))), v(E_3), v(q(3, 3)))\}$. Now, the updated values for E_j are:

$E_1 = \{e(1, 1, 1), e(1, 1, 2), e(2, 1, 1), e(3, 1, 1)\}$,

$E_2 = \{e(1, 2, 1), e(1, 2, 2), e(2, 2, 1), e(3, 2, 1)\}$,

$E_3 = \{e(1, 3, 1), e(2, 3, 1), e(2, 3, 2), e(3, 3, 1)\}$.

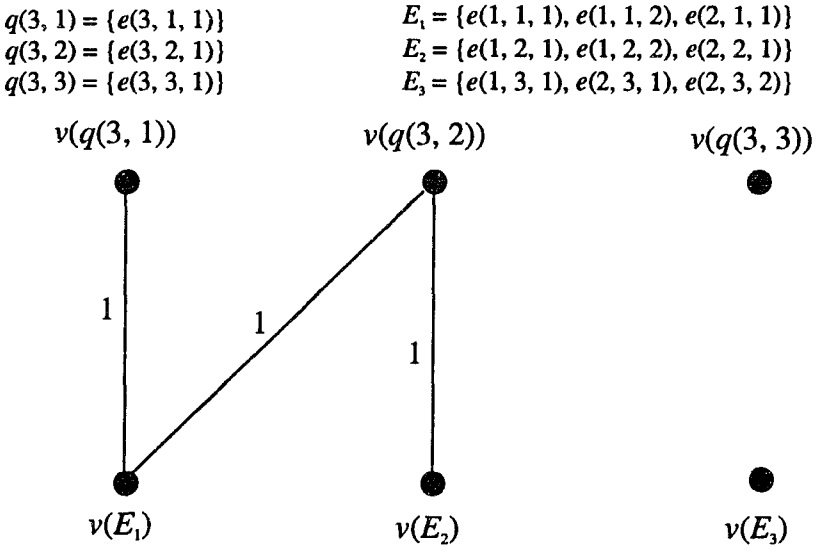


Figure 3.12: $L(G_5)^2$.

Bipartite graph $L(G_5)^3$ is shown in Figure 3.13. The weighted maximum matching is $\{(v(E_1), v(q(4, 1))), (v(E_2), v(q(4, 2))), (v(E_3), v(q(4, 3)))\}$. The updated final values of E_j are:

$E_1 = \{e(1, 1, 1), e(1, 1, 2), e(2, 1, 1), e(3, 1, 1), e(4, 1, 1)\}$,

$E_2 = \{e(1, 2, 1), e(1, 2, 2), e(2, 2, 1), e(3, 2, 1), e(4, 2, 1)\}$, and

$E_3 = \{e(1, 3, 1), e(2, 3, 1), e(2, 3, 2), e(3, 3, 1), e(4, 3, 1)\}$.

Figure 3.14 shows the induced subgraphs corresponding to the above partition. By exhaustive search, we found that the color partition shown in Figure 3.14 is optimal. Therefore, in our particular example, the heuristic algorithm produced an optimal color

partition. In Figure 3.14, induced subgraph H_1 corresponds to 2 drivers and 3 receivers. Induced subgraph H_2 corresponds to 3 drivers and 2 receivers. Induced subgraph H_3 corresponds to 3 drivers and 3 receivers. Therefore an optimal multiple bus system realizing *IFG* G_5 of Figure 3.10 consists of 3 buses, 8 drivers, and 8 receivers.

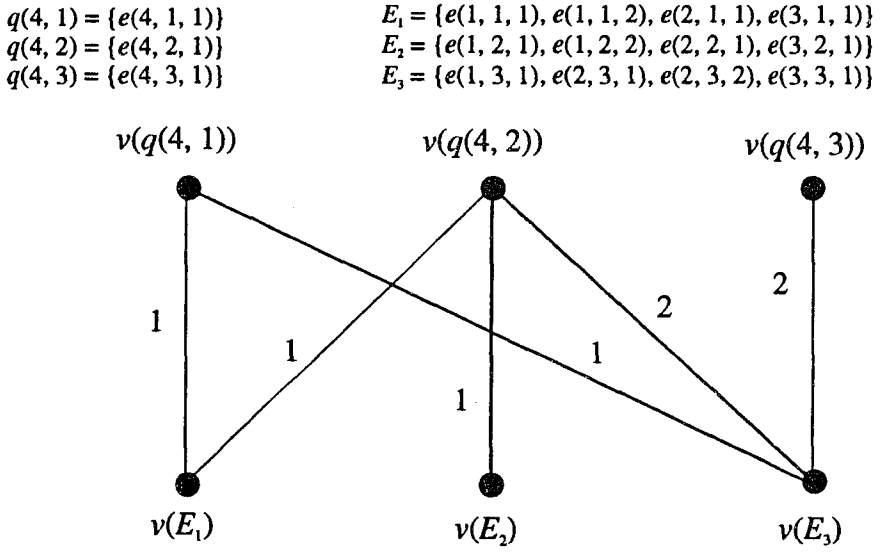


Figure 3.13: $L(G_5)^3$.

3.8 Performance of the Algorithm

In the above algorithm, we repeatedly find a maximum weighted matching of a bipartite graph. Each partite set has at most $\beta(G)$ vertices. Therefore, one step takes $O((\beta(G))^3)$ time using the algorithm given in [56]. Hence the time complexity of the heuristic algorithm is $O(c(\beta(G))^3)$. Our motive for presenting the above algorithm is not to provide the best possible algorithm to find a color partition of a general *IFG*. Instead, we meant to present an approach or methodology which can be used to find efficient

heuristic algorithms. We will discuss how close is our solution to the optimal solution and provide some ideas to improve on it.

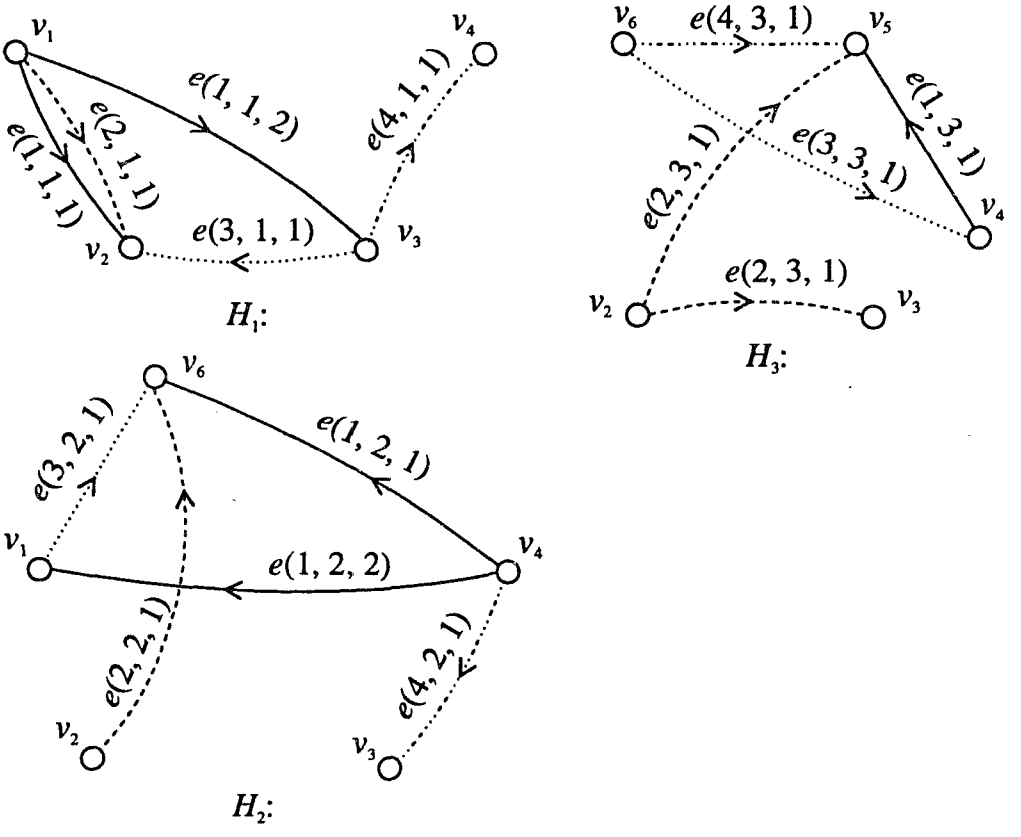


Figure 3.14: Induced subgraphs of G_5 corresponding to the color partition output by Algorithm 3.1.

For $r = 1$ through $c - 1$, we have constructed bipartite graph $L(G)^r$, where $X(L(G)^r) = \{v(E_i) : 1 \le i \le \beta_1\}$ and $Y(L(G)^r) = Q^v(G, r + 1)$. The outcome of the algorithm depends on the matching of vertices of $\{v(E_j) : 1 \le j \le \beta_1\}$ to those of $Q^v(G, r + 1)$. The criteria to match vertex $v(E_j) \in \{v(E_j) : 1 \le j \le \beta_1\}$ to vertex $v(q(G, r + 1, \tau)) \in Q^v(G, r + 1)$ is the compatibility of $q(G, r + 1, \tau)$ and E_j. The main reason for the outcome to possibly deviate from the optimal solution is the following. Set E_j contains edges of primary colors 1 through r and set $q(G, r + 1, \tau)$ contains edges of primary color $r + 1$

only. We did not consider the compatibility of $q(G, r + 1, \tau)$ with the edges of primary colors $r + 2$ through c. We can improve the outcome of the algorithm by introducing some kind of lookahead at the edges of primary colors $r + 2$ through c. For example, suppose that there exists an edge set, say $q'(G, r + 2, \tau')$, which is z_1-compatible with $q(G, r + 1, \tau)$ and z_2-compatible with E_j. Unless $q(G, r + 1, \tau)$ is matched to E_j at iteration r, we cannot include both subsets $q'(G, r + 2, \tau')$ and $q(G, r + 1, \tau)$ in E_j at iteration $r + 1$, thereby saving $z_1 + z_2$ interfaces. In order to increase the chance of matching $q(G, r + 1, \tau)$ with E_j at iteration r, we could add an extra weight δ to edge $(v(E_j), v(q(G, r + 1, \tau)))$ of $L(G)^r$. If the edge does not exist in $L(G)^r$, we can insert an edge with weight δ. Therefore, by suitably assigning weights to edges in the bipartite graphs in each iteration, one can design a color partition algorithm whose outcome is very close to the optimal solution. Given a constant $\in$, it seems possible that, one can choose suitable weights for edges in bipartite graphs in each iteration such that the cost of the resulting multiple bus system is guaranteed to be no more than $(1 + \in)opt$, where *opt* is the cost of an optimal bus system. This is an interesting and challenging problem in its own right. However, we do not pursue this problem further in this dissertation.

CHAPTER 4

OPTIMAL BUS ASSIGNMENT FOR VERTEX SYMMETRIC *IFGs*

Spaghetti code writing is generally considered as a bad practice. Structured, regular algorithms are considered most popular in today's standards. Furthermore, most efficient algorithms for many important problems exhibit inherent regularity. For example, Preparata and Vuillemin defined an algorithmic class called *Ascend/Descend* which is based on iterative rendition of divide and conquer [115]. They have shown that fundamental parallel algorithms such as merging, Fast Fourier Transform, sorting, convolution and matrix operations are either instances of the scheme (covered by the algorithmic class) or simple combinations of such instances. The *Ascend/Descend* class exhibits data exchange pattern which corresponds directly to cube permutations which have very regular *IFG* representation. Therefore, *IFGs* corresponding to most real algorithms of importance would have certain regularities. Besides, an optimal *MBS* realizing a heterogeneous *IFG* may also be heterogeneous, which is very undesirable from an implementation point of view. Modularity and regularity are very attractive features for *VLSI* implementations. Almost every interconnection network found in multiprocessor systems have certain amount of regularity and modularity. Sets of interconnection functions used in most existing *SIMD* machines (such as hypercube, torus, star network etc.) correspond to symmetric *IFGs*. Multiple bus systems which are regular in nature are also attractive from an implementation point of view. On the other hand, an irregular *MBS* is very unattractive as a multiprocessor system. An optimal *MBS* realizing a given irregular algorithm will also be irregular.

Thus, even though finding an optimal color partitioning for a general *IFG* has some theoretical interest, it has very little practical implication in regard to the design of an optimal *MBS*. In this chapter, we address the problem of finding an optimal color partition of a vertex symmetric or regular *IFG*. We show that an *IFG* is vertex symmetric if and only if it is a Cayley color graph associated with a finite group and its generating set. This allow help us to use some concepts from group theory to perform an optimal color partition of a vertex symmetric *IFG* and to analyze the properties of the resulting *MBS*. We show that there exist many optimal color partitions of a vertex symmetric *IFG*. We choose a particular partition which has many desirable properties other than being optimal. We show that the resulting *MBS* is symmetric. We also show the superiority of an *MBS* over a static direct link interconnection network realizing the same symmetric *IFG*, in terms of the number of ports per processor, the number of neighbors per processor, and the diameter. Furthermore, we present a polynomial time algorithm to perform an optimal color partition of a vertex symmetric or regular *IFG*.

4.1 Preliminaries

In this chapter we assume that the interconnection functions associated with an *IFG* does not contain broadcasting operations. When broadcasting operations are involved, Algorithm 3.1 can be used to find a near optimal color partition. Therefore, each vertex of an *IFG* is assumed to have distinct color outgoing edges.

Definition 4.1: A *color preserving automorphism* [146] of an edge colored digraph G is a permutation ψ on $V(G)$ such that (u, v) is an edge in $E(G)$ if and only if $(\psi(u), \psi(v))$ is an edge in $E(G)$ of the same color, for all $u, v \in V(G)$.

Definition 4.2: An edge colored digraph is said to be *vertex symmetric* (or *vertex transitive*) if for every pair of vertices u and v, there exists a color preserving automorphism of the graph which maps u to v.

Intuitively speaking, an edge colored digraph is vertex symmetric if it looks the same when viewed from any vertex. We will show how to find an optimal edge partition of a vertex symmetric *IFG*.

Lemma 4.1: If an *IFG* G is vertex symmetric, each vertex $v \in V(G)$ has incoming edges of distinct colors.

Proof: Let $|V(G)| = n$. Suppose a certain vertex in $V(G)$ has $x > 1$ incoming edges of color r. Since G is vertex symmetric, *each* vertex must have x incoming edges of color r. Therefore, the total number of edges of color r is rx. Since G is vertex symmetric, each vertex must have x outgoing edges of color r. Since in an *IFG* each vertex has outgoing edges of distinct colors, it is not possible for a vertex to have x outgoing edges of color r for $x > 1$. Thus each vertex in $V(G)$ has incoming edges of distinct colors. □

Recently, much attention has been focused on analyzing existing interconnection networks and designing new ones using group theoretic concepts [5], [6], [27]. Here we will utilize group theoretic concepts in analyzing symmetric *IFG*s for the purpose of constructing an optimal *MBS*. For completeness, we briefly introduce the very basics of group theory, rather informally.

Let $\Gamma = \{\gamma_1, \gamma_2, ...\}$ be a set and $\circ$ be a binary operator on the elements of the set. Then the algebraic structure $<\Gamma, \circ>$ is called a *group* if it satisfies the four axioms: 1) *closure*: $\gamma_i \circ \gamma_j \in \Gamma$, $\forall\ \gamma_i, \gamma_j \in \Gamma$; 2) *associativity*: $\gamma_i \circ (\gamma_j \circ \gamma_k) = (\gamma_i \circ \gamma_j) \circ \gamma_k \in \Gamma$, $\forall\ \gamma_i, \gamma_j, \gamma_k \in \Gamma$; 3) *identity*: there exists an element $e \in \Gamma$ such that $e \circ \gamma = \gamma \circ e = \gamma$, $\forall\ \gamma \in \Gamma$; and 4)

inverse: for every $\gamma \in \Gamma$, there exists $\gamma^{-1} \in \Gamma$ such that $\gamma \circ \gamma^{-1} = \gamma^{-1} \circ \gamma = e$. With minor abuse of notation (which usually is the case), we will use the same symbol Γ to represent both the set and the group; the binary operation is usually implied. Whether we are referring to the group or the set would be clear from the context. For simplicity, binary operator $\circ$, called group *composition*, is usually omitted from expressions. We write $\gamma_i\gamma_j$ instead of $\gamma_i \circ \gamma_j$. Also, $\gamma_i\gamma_j$ is called the *product* of γ_i and γ_j. We abbreviate the product $\gamma\gamma...\gamma$ by γ^i, where i is the number of terms in the product. The *order*, or the *cardinality* of a group Γ, denoted by $|\Gamma|$, is the number of elements in the set Γ. The order of an element $\gamma \in \Gamma$ is the smallest integer q such that $\gamma^q = e$. If the binary operation is commutative, that is, if $\gamma_i\gamma_j = \gamma_j\gamma_i$ for all $\gamma_i, \gamma_j \in \Gamma$, then Γ is called an *abelian* group. The group containing only the identity element is called the *trivial* group. A group of finite order is called a *finite* group. If a group is finite, all its elements are of finite order. In this paper, we only consider finite groups. A subset $\Delta \subseteq \Gamma$ is said to be a set of *generators* of Γ, if every element of Γ can be expressed as a product of the elements in Δ and their inverses.

Let Γ be a non trivial finite group with a generating set Δ. We associate with Γ and Δ an edge colored digraph $C_\Delta(\Gamma)$ called *Cayley color graph* [102], [146]. The vertices of $C_\Delta(\Gamma)$ are the group elements. Each of the generators in Δ is assigned a distinct color. There exists a directed edge (γ_1, γ_2) of color r in $C_\Delta(\Gamma)$ if and only if $\delta_r \in \Delta$ and $\gamma_2 = \gamma_1\delta_r$. Clearly, the number of vertices in $C_\Delta(\Gamma)$ is equal to the cardinality of Γ. Without loss of generality, we assume that Δ does not contain the identity element e of the group Γ (this disallows self-loops in the graph). A generator in Δ is said to be *redundant* [146] if it can be obtained from the remaining elements in Δ.

Not only edge colored directed graphs, but also undirected graphs can be represented using groups and generating sets. If $\delta \in \Delta$ implies $\delta^{-1} \in \Delta$, then the underlying undirected graph of $C_\Delta(\Gamma)$ (by possibly coalescing multiple edges) will be called *Cayley graph* [146] and is usually denoted by $G_\Delta(\Gamma)$. Many static interconnection networks encountered in the literature can be represented as Cayley graphs. Some examples of such networks are hypercube [119], cube connected cycles [115], star [5], pancake [54], and bubble sort [5].

Example 4.1: Consider the group, denoted by $\Gamma_{Q(n)}$, whose elements are the binary vectors of length n and the group composition is the bitwise exclusive-or operation. Clearly, $\Gamma_{Q(n)}$ is an abelian group and contains 2^n elements. The set of unit vectors in $\Gamma_{Q(n)}$ forms a non redundant generating set, denoted by $\Delta_{Q(n)}$. We will use notation $G_{Q(n)}$ to represent the Cayley color graph associated with group $\Gamma_{Q(n)}$ and generating set $\Delta_{Q(n)}$. In example 2.4, we constructed an optimal *IFG* from the *CFG* C_4 (see Figure 2.12) representing the *Ascend/Descend* algorithmic class. The constructed *IFG* (shown in Figure 2.14) is in fact $G_{Q(3)}$. If we regard *IFG* as the representation of interconnection functions of an *SIMD* machine, then $G_{Q(n)}$ is the *IFG* corresponding to the cube interconnection functions of an n-dimensional hypercube. Due to the importance of cube interconnection functions, we will refer to $G_{Q(n)}$ in several occasions in this dissertation.

Theorem 4.1: Cayley color graph $C_\Delta(\Gamma)$ is strongly connected.

Proof: Let $\Delta = \{\delta_1, \delta_2, ..., \delta_{|\Delta|}\}$. Let γ_a and γ_b be two arbitrary elements of Γ. Then $\gamma_b = \gamma_a h_1 h_2 ... h_r$, where h_1 through h_r are elements (not necessarily distinct) taken from the set $\{\delta_1, \delta_2, ..., \delta_{|\Delta|}\} \cup \{\delta_1^{-1}, \delta_2^{-1}, ..., \delta_{|\Delta|}^{-1}\}$. Since we assume that Γ is a finite group, each of its elements is of finite order. If q_i is the order of δ_i, then $\delta_i^{-1} = \delta_i^{q_i - 1}$, for $1 \le i \le |\Delta|$.

Thus $\gamma_b = \gamma_a \delta_{x_1}^{s_1} \delta_{x_2}^{s_2} \ldots \delta_{x_r}^{s_r}$, where s_1 through s_r are some finite positive integers and $x_i \in \{1, 2, \ldots, |\Delta|\}$, for $1 \leq i \leq r$. Hence there is a directed path from γ_a to γ_b, and therefore, $C_\Delta(\Gamma)$ is strongly connected. □

Lemma 4.2: Cayley color graph $C_\Delta(\Gamma)$ is an *IFG*.

Proof: Each vertex of $C_\Delta(\Gamma)$ has outgoing edges of distinct colors. In addition, Theorem 4.1 indicates that the underlying undirected graph of $C_\Delta(\Gamma)$ is connected. Thus the proof is reached. □

The following theorem will provide us the main tool in constructing an optimal *MBS* realizing a symmetric *IFG*.

Theorem 4.2: An *IFG* is vertex symmetric if and only if it can be represented as a Cayley color graph $C_\Delta(\Gamma)$ associated with a finite group Γ and its generating set Δ.

Proof: For sufficiency, let $G = C_\Delta(\Gamma)$ be the Cayley color graph associated with group Γ and its generating set Δ. We prove that, for any two vertices γ_1 and γ_2 in $C_\Delta(\Gamma)$, there exists a color preserving automorphism which maps γ_1 to γ_2. Consider the permutation ψ defined by $\psi(\gamma) = \gamma_2 \gamma_1^{-1} \gamma$, where $\gamma, \gamma_1, \gamma_2 \in \Gamma$. Let (x, y) be a color r edge in G. Then $y = x\delta_r$ for some $\delta_r \in \Delta$. From the definition, $\psi(y) = \gamma_2 \gamma_1^{-1} y = \gamma_2 \gamma_1^{-1} x\delta_r = \psi(x)\delta_r$. Therefore, $(\psi(x), \psi(y))$ is a color r edge in G. Therefore, ψ is a color preserving automorphism for G. Furthermore, $\psi(\gamma_1) = \gamma_2 \gamma_1^{-1} \gamma_1 = \gamma_2$. Hence γ_1 maps to γ_2, and therefore, G is vertex symmetric.

For necessity, let $G = (V, E)$ be a vertex symmetric *IFG* with c colors. We prove that G is a Cayley color graph. Let $V = \{v_1, v_2, \ldots, v_n\}$. Let γ_i be a color preserving automorphism which maps v_1 to v_i for $1 \leq i \leq n$. We will show that the elements in the set $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ are unique, that is, there is only one color preserving automorphism

which maps v_1 to v_i, for $1 \leq i \leq n$. Suppose there exists another color preserving automorphism γ_i' which maps v_1 to v_i. Let v_x be any vertex in V. Since the underlying undirected graph of G is connected, there exists an undirected path $v_1=u_1, u_2, ..., u_m=v_x$, where, either (u_i, u_{i+1}) or (u_{i+1}, u_i) is an edge in E for $1 \leq i \leq m - 1$. First suppose that edge (u_1, u_2) is in E and that it is of color r. From the hypothesis, $\gamma_i(u_1) = \gamma_i'(u_1) = v_i$. Since there is only one edge of color r directed away from v_i, the two edges $(\gamma_i(u_1), \gamma_i(u_2))$ and $(\gamma_i'(u_1), \gamma_i'(u_2))$ must be the same, and therefore, $\gamma_i(u_2) = \gamma_i'(u_2)$. Next suppose (u_2, u_1) is an edge in E of color r. Since there is only one edge of color r directed towards v_i (from Lemma 4.1), we get $\gamma_i(u_2) = \gamma_i'(u_2)$. Similarly, we can show that $\gamma_i(u_3) = \gamma_i'(u_3)$. By repeated application of the same argument to the remaining vertices of the path, $v_1=u_1$, $u_2, ..., u_m=v_x$, we get $\gamma_i(v_x) = \gamma_i'(v_x)$. This is true for every vertex $v_x \in E$. Therefore, γ_i and γ_i' represent the same automorphism. Therefore, the elements in the set $\{\gamma_1, \gamma_2, ..., \gamma_n\}$ are distinct and all the color preserving automorphisms of G are included in the set.

It is straightforward to verify that the elements in the set $\{\gamma_1, \gamma_2, ..., \gamma_n\}$ form a group with γ_1 as the identity element. Let that group be Γ. Let w_r be the vertex in V such that (v_1, w_r) is an edge of color r. Also, let the color preserving automorphism corresponding to vertex w_r be δ_r, for $1 \leq r \leq c$. Then $\Delta = \{\delta_1, \delta_2, ..., \delta_c\}$ is a generating set for Γ and G is the Cayley color graph $C_\Delta(\Gamma)$. □

The above theorem allows us to use group theoretic concepts to find an optimal color partition of a vertex symmetric *IFG*. If an *IFG* G is a Cayley color graph $C_\Delta(\Gamma)$, then the generating set Δ is isomorphic with the set of interconnection functions associated with G. This explains why we shouldn't include the identity element e in the generating set Δ. An identity interconnection function does not serve any purpose in

interprocessor communication. Usually, interconnection functions in a set are independent, that is, one function cannot be constructed from the remaining functions in the set. Therefore, we emphasize on generating sets which are non redundant. In the first part of this chapter, unless otherwise stated, Cayley color graphs are assumed to have non redundant generating sets. In Section 4.5, we will specifically address Cayley color graphs with redundant generating sets.

For an arbitrary *IFG* G, a lower bound on $|Y\pi(G))|$ is $\beta(G)$ (see Lemma 3.2). As we have already stated in Section 3.4, the existence of such a lower bound on $|E(\pi(G)|$ is not known. However, we can establish a lower bound on $|E(\pi(G))|$ if G belongs to a certain class of graphs.

Definition 4.3: A sequence of edges in an *IFG* is called an *alternately oriented path* if every pair of consecutive edges in the sequence are directed towards or directed away from the same vertex. If the first edge and the last edge of the sequence are also directed towards or directed away from the same vertex, the sequence is called an *alternately oriented cycle*. An alternately oriented path or cycle will also be denoted by a sequence of *vertices*, where vertices are selected from the sequence of edges in the obvious manner.

Definition 4.4: An *IFG* is said to belong to class $\mathscr{E}$ if it contains no alternately oriented cycles with distinct color edges.

Example 4.2: Figure 4.1(*a*) shows an *IFG* belonging to class $\mathscr{E}$. Notice that although cycle (v_1, v_2, v_3, v_4) of Figure 4.1(*a*) is an alternately oriented cycle, it contains two edges (v_1, v_2) and (v_3, v_4) of color 1. Figure 4.1(*b*) shows an *IFG* which does not belong to class $\mathscr{E}$.

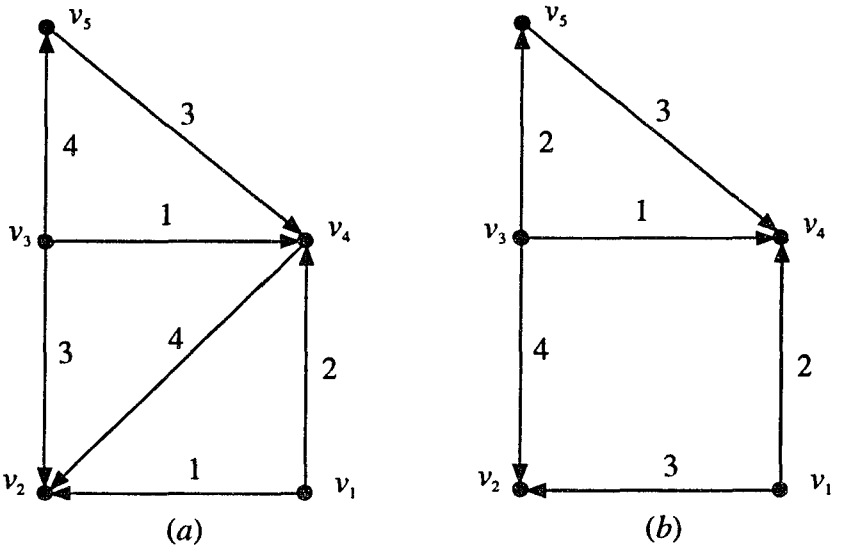


Figure 4.1: (*a*) An *IFG* in class $\mathscr{E}$, (*b*) An *IFG* not in class $\mathscr{E}$.

It is obvious that the cardinality of an alternately oriented cycle is even. It is interesting to notice that an alternately oriented path or cycle has the property that adjacent edges are compatible (see Definition 3.7). The following theorem establishes a lower bound on the number of interfaces of an *MBS* realizing an *IFG* belonging to class $\mathscr{E}$.

Theorem 4.3: Let G be an *IFG* belonging to class $\mathscr{E}$. Then for any color partition π, $|E(\pi(G))| \geq |E(G)| + \beta(G)$.

Proof: Let $E_1, E_2, ..., E_b$ be the subsets of $E(G)$ corresponding to color partition π. Let H_j be the subgraph of G induced by the subset E_j, $1 \leq j \leq b$. (Note that $|Y(\pi(G))| = b$.) From H_j, construct H_j^* by splitting every vertex v incident with *both* incoming and outgoing edges into two vertices v^{in} and v^{out} such that v^{in} is incident with only incoming edges and v^{out} is incident with only outgoing edges, for $1 \leq j \leq b$. If the superscripts *in* and *out* of the labels of the vertices are disregarded, the edge set of H_j^* is still E_j, for $1 \leq j \leq b$, and

therefore, the final *MBS* is not affected. Any cycle present in H_j^* must be an alternately oriented cycle. Furthermore, any cycle in H_j^* is also a cycle in G (again, if we disregard the superscripts of the labels). Since G is assumed to be in $\mathscr{E}$, H_j^* does not contain a cycle, that is, H_j^* underlies a tree. Therefore, the number of vertices in H_j^* is at least $|E_j|$ + 1. According to the construction, each vertex in H_j^* is incident with either incoming or outgoing edges (not both). Therefore, $|J(E_j)| \geq |E_j| + 1$, $1 \leq j \leq b$. Hence, by Lemma 3.1, $|E(\pi(G))| \geq \sum_{j=1}^{b} (|E_j| + 1) = \sum_{j=1}^{b} |E_j| + b = |E(G)| + b$. Also, by Lemma 3.2, $b \geq \beta(G)$. Hence the theorem follows. □

We must emphasize the fact that if an *IFG* G belongs to class $\mathscr{E}$, it does not necessarily mean that a partition π exists such that $|E(\pi(G))|$ is equal to its lower bound $|E(G)| + \beta(G)$. In other words, the lower bound for $|E(\pi(G))|$ stated in Theorem 4.3 is not always reachable. However, if the lower bound can be reached by color partition π', then the proof of Theorem 4.3 shows that $|Y(\pi'(G))| = \beta(G)$. Therefore, such a color partition minimizes both $|Y(\pi'(G))|$ and $|E(\pi'(G))|$. Therefore, irrespective of the values κ_3 and κ_3 used in the optimization function, π' is an optimal color partition. We state this in the following theorem.

Theorem 4.4: Let G be an *IFG* belonging to class $\mathscr{E}$. If color partition π' satisfies the condition $|E(\pi'(G))| = |E(G)| + \beta(G)$, then π' is an optimal color partition. □

4.2 Optimal Color Partitioning of a Cayley Color Graph

When the *IFG* is a Cayley color graph $C_\Delta(\Gamma)$, we can analytically find an optimal color partition of $C_\Delta(\Gamma)$ using group Γ and the generating set Δ. We first present some important characteristics of Cayley color graphs.

Lemma 4.3: $\beta(C_\Delta(\Gamma)) = |\Gamma|$.

Proof: Each vertex of $C_\Delta(\Gamma)$ has one edge from each color. Since there are $|\Gamma|$ vertices in the graph, there are $|\Gamma|$ edges from each color. Therefore, $\beta(C_\Delta(\Gamma)) = |\Gamma|$. □

Theorem 4.5: If Δ is a (non redundant) generating set of the group Γ, then the Cayley color graph $C_\Delta(\Gamma)$ belongs to class $\mathscr{E}$.

Proof: If $C_\Delta(\Gamma)$ contains an alternately oriented cycle with distinct color edges, then there must exist distinct generators $\delta_1, \delta_2, ..., \delta_{2k}$ such that $h_1h_2...h_{2k} = e$, where $h_i = \delta_i^{(-1)^i}$ for $1 \le i \le 2k \le |\Delta|$. Since the elements in Δ are non redundant, this is not possible. Hence $C_\Delta(\Gamma)$ belongs to class $\mathscr{E}$. □

Theorem 4.6: Let $E_1 \subset E(C_\Delta(\Gamma))$ be a subset containing exactly one edge from each color. Also, let H_1 be the subgraph induced by E_1. Then $E(C_\Delta(\Gamma))$ can be partitioned into subsets $E_1, E_2, ..., E_{|\Gamma|}$ such that subgraph H_j induced by E_j, $2 \le j \le |\Gamma|$, is isomorphic with H_1.

Proof: Define $\gamma_j(E_1) = \{(\gamma_j x, \gamma_j y) : (x, y) \in E_1\}$, $1 \le j \le |\Gamma|$. By the definition, if $(x, y) \in E_1$, then $y = x\delta$ for some generator $\delta \in \Delta$. Therefore, $(\gamma_j x, \gamma_j y) = (\gamma_j x, \gamma_j x\delta)$. Hence, edge (x, y) in E_1 and its image $(\gamma_j x, \gamma_j y)$ in $\gamma_j(E_1)$ are of the same color. Thus, $\gamma_j(E_1)$, $1 \le j \le |\Gamma|$, contains exactly one edge from each color. Now, we make the claim that, for $j \ne j'$, $\gamma_j(E_1) \cap \gamma_{j'}(E_1) = \{\}$. Suppose on the contrary that edge (x^*, y^*) is common to both $\gamma_j(E_1)$ and $\gamma_{j'}(E_1)$. For mapping γ_j, edge (x^*, y^*) in $\gamma_j(E_1)$ must be the image of a certain edge, say, (x, y), in E_1. Similarly, for mapping $\gamma_{j'}$, edge (x^*, y^*) in $\gamma_{j'}(E_1)$ must be the image of a certain edge $(x', y') \in E_1$. We have already shown that mappings γ_j and $\gamma_{j'}$ preserve the colors of the edges. Therefore, (x, y) and (x', y') must represent the same edge in E_1. Thus, $x^* = \gamma_j x = \gamma_{j'} x$. This is not possible since $j \ne j'$. Therefore, the claim is true, that is, $\gamma_j(E_1)$ and $\gamma_{j'}(E_1)$ are disjoint subsets. Now, without loss of generality, assume $\gamma_1 = e$.

Then we have disjoint subsets E_1, E_2, ..., $E_{|\Gamma|}$, where $E_j = \gamma_j(E_1)$, $1 \le j \le |\Gamma|$. Since the number of edges in the subsets are $|\Gamma|.|\Delta|$, they form a partition on $E(C_\Delta(\Gamma))$. It is clear from mapping γ_j that H_j is isomorphic with H_1. □

Therefore, for any subset $E_j \subset E(C_\Delta(\Gamma))$ containing one edge from each color, there exists a color partition such that the induced subgraphs are isomorphic to one another. According to Theorems 4.4 and 4.5, if we can find a color partition π such that $|E(\pi(C_\Delta(\Gamma)))| = |\Gamma|.|\Delta| + |\Gamma|$, then π must be optimal. Therefore, according to Lemma 3.1 and Theorem 4.6, what we need to find is only one subset $E_j \subset E(C_\Delta(\Gamma))$, containing exactly one edge from each color, such that $|J(E_j)| = |\Delta| + 1$. There are many different ways subset E_j can be formed such that $|J(E_j)| = |\Delta| + 1$. In this dissertation, we do not attempt to exhaust all possible partitioning methods nor do we try to compare their relative merits. We choose E_j as an alternately oriented path with one edge from each color. As we see later, the corresponding optimal color partition will produce an *MBS* with many attractive features. Next we formally define such a color partition.

Definition 4.5: Define $\phi_{\Gamma,\Delta} = \{E_j : 1 \le j \le |\Gamma|\}$, where

$E_j = \{(\gamma_j, \gamma_j h_1), (\gamma_j h_1, \gamma_j h_1 h_2), ..., (\gamma_j h_1 h_2 ... h_{c-1}, \gamma_j h_1 h_2 ... h_c)\}$, $h_i = (\delta_i)^{(-1)^{i-1}}$, for $1 \le i \le |\Delta|$.

From the above discussion, the following theorem is straightforward.

Theorem 4.7: Let $C_\Delta(\Gamma)$ be a Cayley color graph. Then $\phi_{\Gamma,\Delta}$ is an optimal color partition. Furthermore, $|E(\phi_{\Delta,\Gamma}(C_\Delta(\Gamma)))| = |\Gamma|(|\Delta| + 1)$. □

Let H_j be the subgraph induced by subset E_j of edges due to the optimal color partition $\phi_{\Gamma,\Delta}$. Then the vertices γ_j, $\gamma_j\delta_1$, $\gamma_j\delta_1\delta_2^{-1}$, ..., $\gamma_j\delta_1\delta_2^{-1}...\delta_{|\Delta|}^{x}$ ($x = -1$ when $|\Delta|$ is even, otherwise $x = 1$) of H_j will be called the 0th, the first, ... the $|\Delta|$th vertex of H_j, respectively. Also, the ith vertex of H_j will be denoted by $v_i(H_j)$ for $0 \le i \le |\Delta|$. Figure 4.2

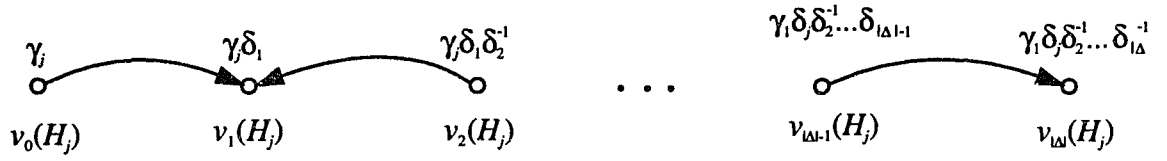


Figure 4.2: Induced subgraph H_j of partition $\phi_{\Gamma,\Delta}$ ($|\Delta|$ is even).

shows the induced subgraph H_j corresponding to color partition $\phi_{\Gamma,\Delta}$. The 0th vertex of H_j (that is, $v_0(H_j)$) is γ_j. For simplicity, the bus corresponding to subgraph H_j will also be denoted by γ_j. From the definition of color partition $\phi_{\Gamma,\Delta}$ of $C_\Delta(\Gamma)$ and from the notation used, the following observations can be made.

(1) There is a one to one correspondence between the elements in Γ and the elements in each of the sets $\{v_i(H_j) : 1 \le j \le |\Gamma|\}$, $0 \le i \le |\Delta|$.

(2) There is a one to one correspondence between the elements of Γ and the buses in $M(\Gamma, \Delta)$.

(3) If i is even (odd), then vertex $v_i(H_j)$ is incident with only outgoing (incoming) edges in subgraph H_j, $1 \le j \le |\Gamma|$.

(4) If i is even (odd), then the processor corresponding to vertex $v_i(H_j)$ is connected to bus γ_j via a driver (receiver), $1 \le j \le |\Gamma|$.

Example 4.3: Consider the *IFG* G_3 shown in Figure 2.14. Notice that G_3 is obtained by optimal processor assignment to the *CFG* C_4, which corresponds to the *Ascend/Descend* class of parallel algorithms. As we have already stated, that *IFG* is the Cayley color graph $G_{Q(3)}$. The group associated with $G_{Q(3)}$ is $\Gamma_{Q(3)} = \{000, 001, 010, 011, 100, 101, 110, 111\}$ and its generating set is $\Delta_{Q(3)} = \{001, 010, 100\}$. Edges corresponding to generators 001, 010, and 100 are represented as solid, broken, and dotted lines, respectively. Group

composition is the exclusive-or operation of binary strings. Figure 4.3 shows the subgraphs of $G_{Q(3)}$ induced by the subsets of partition $\phi_{\Gamma_{Q(3)},\Delta_{Q(3)}}$. Figure 4.4 shows the corresponding optimal *MBS*.

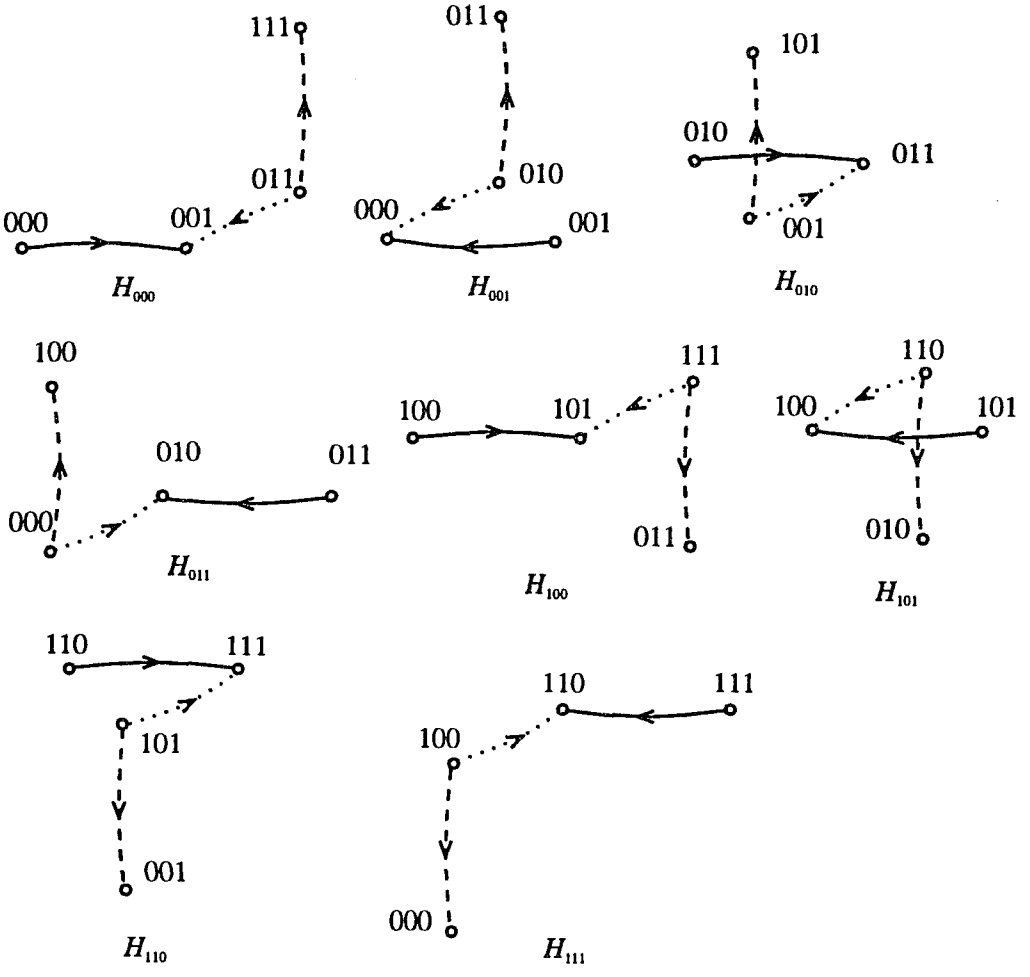


Figure 4.3: Induced subgraphs of $G_{Q(3)}$ by partition $\phi_{\Gamma_{Q(3)},\Delta_{Q(3)}}$.

4.3 Properties of *MBS*s Realizing Symmetric *IFG*s

In this section we will highlight some attractive features of an *MBS* which optimally realizes a symmetric *IFG*. From Theorem 4.2, a symmetric *IFG* can be

expressed as a Cayley color graph $C_\Delta(\Gamma)$. Therefore we will use the group Γ and its generating set Δ in obtaining the properties of an *MBS* realizing a symmetric *IFG*.

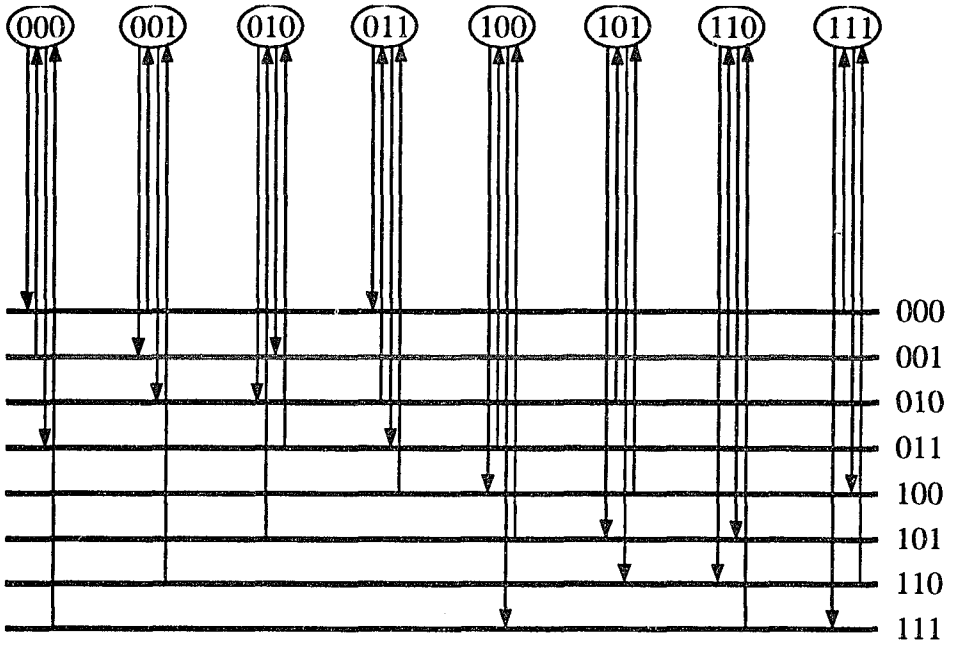


Figure 4.4: *MBS* corresponding to partition in Figure 4.3.

Definition 4.6: The *MBS* corresponding to the *MBG* $\phi_{\Gamma,\Delta}(C_\Delta(\Gamma))$ will be denoted by $M(\Gamma, \Delta)$, where $\phi_{\Gamma,\Delta}$ is the optimal color partition as given in Definition 4.5. The direct link interconnection network corresponding to the Cayley color graph $C_\Delta(\Gamma)$ will be denoted by $N(\Gamma, \Delta)$. Also, we use $N_{Q(n)}$ and $M_{Q(n)}$ to denote $N(\Gamma_{Q(n)}, \Delta_{Q(n)})$ and $M(\Gamma_{Q(n)}, \Delta_{Q(n)})$, respectively.

Other than the quite interesting properties inherent with any multiple bus system, properties of the *MBS* $M(\Gamma, \Delta)$ (to follow) will show how well an *MBS* can be used as an algorithm specific architecture. We will first show that the *MBS* $M(\Gamma, \Delta)$ is symmetric. Then we will show the superiority of the *MBS* $M(\Gamma, \Delta)$ over the counterpart direct link

interconnection network $N(\Gamma,\Delta)$, in terms of the number of ports per processor, the number of neighbors, and the diameter (to be defined).

4.3.1 Symmetry

To show that the *MBS* $M(\Gamma, \Delta)$ is symmetric, we introduce the following definition.

Definition 4.7: A directed bipartite graph (X, Y, E) is said to be *vertex symmetric* if for every pair of vertices u and v, both of them in X or both of them in Y, there exists an automorphism of the graph that maps u to v. An *MBS* is said to be *symmetric* if its corresponding bipartite graph is vertex symmetric.

Intuitively speaking, if we view a symmetric *MBS* from a processor, it appears the same irrespective of the processor used. Similarly, if we view the system from a bus, it appears the same irrespective of the bus used.

Theorem 4.8: The bipartite graph $\phi_{\Gamma\Delta}(C_\Delta(\Gamma))$ is vertex symmetric.

Proof: Let $\phi_{\Gamma\Delta}(C_\Delta(\Gamma)) = (X, Y, E)$. Consider a pair of vertices γ_1 and γ_2 in X. Let mapping ψ defined by $\psi(\gamma) = \gamma_2\gamma_1^{-1}\gamma$, where γ is an elements of Γ. Let (γ_y, γ_x) be an edge in E such that $\gamma_x \in X$ and $\gamma_y \in Y$. Then, according to our usual notation, γ_x is the r^{th} vertex of H_y for some odd integer r. Thus, $\gamma_x = \gamma_y\delta_1\delta_2^{-1}...\delta_r$, and therefore, $\gamma_y^{-1}\gamma_x = \delta_1\delta_2^{-1}...\delta_r$. But $(\psi(\gamma_y))^{-1}(\psi(\gamma_x)) = \gamma_y^{-1}(\gamma_1\gamma_2^{-1})(\gamma_2\gamma_1^{-1})\gamma_x = \gamma_y^{-1}\gamma_x$. Thus $\psi(\gamma_x) = \psi(\gamma_y)\delta_1\delta_2^{-1}...\delta_r$. Therefore, $\psi(\gamma_x)$ is the r^{th} vertex of H_z, where $\psi(\gamma_y) = \gamma_z$. Hence, there is an edge from γ_z to $\psi(\gamma_x)$, that is, $(\psi(\gamma_y), \psi(\gamma_x)) \in E$. Analogously, we can show that $(\psi(\gamma_y), \psi(\gamma_x)) \in E$ implies $(\gamma_y, \gamma_x) \in E$. Moreover, we can show that $(\gamma_x, \gamma_y) \in E$ if and only if $(\psi(\gamma_x), \psi(\gamma_y)) \in E$. Furthermore, $\psi(\gamma_1) = \gamma_2\gamma_1^{-1}\gamma_1 = \gamma_2$. Thus, for every pair of vertices γ_1 and γ_2 in X, there exists an automorphism of the bipartite graph $\phi_{\Gamma\Delta}(C_\Delta(\Gamma))$ which maps γ_1 to γ_2. By similar reasoning,

we can show that for every pair of vertices γ_3 and γ_4 in Y, there exists an automorphism of the graph which maps γ_3 to γ_4. Hence the bipartite graph $\phi_{\Gamma,\Delta}(C_\Delta(\Gamma))$ is vertex symmetric. □

The above result implies that the *MBS* $M(\Gamma, \Delta)$ is symmetric. This can be observed, for example, in Figure 4.4. Each processor (bus) is identical to every other processor (bus). This property suggests a simple and efficient *VLSI* implementation, due to the fact that symmetry leads to easy routing methods, convenient replacement of faulty components, and area efficient layout.

4.3.2 Number of Ports per Processor

The following result provides one of the favorable properties of an optimal *MBS* realizing a vertex symmetric *IFG*.

Theorem 4.9: The number of output and input communicating ports per processor in $M(\Gamma, \Delta)$ are $\lfloor |\Delta|/2 \rfloor + 1$ and $\lceil |\Delta|/2 \rceil$, respectively.

Proof: Subgraph H_j induced by the subset of edges E_j, $1 \le j \le |\Gamma|$, is an alternately oriented path. With our usual notation, only the vertices $v_0(H_j)$, $v_2(H_j)$, $v_4(H_j)$, ..., $v_{2\lfloor |\Delta|/2 \rfloor}(H_j)$ of H_j are incident with outgoing edges (see Figure 4.2). Let y_j be the vertex in $Y(\phi_{\Gamma,\Delta}(C_\Delta(\Gamma)))$ corresponding to subgraph H_j. Then there is an edge from each of the vertices x_0, x_2, x_4, ..., $x_{2\lfloor |\Delta|/2 \rfloor}$ to vertex y_j in $\phi_{\Gamma,\Delta}(C_\Delta(\Gamma))$, where $x_i \in X(\phi_{\Gamma,\Delta}(C_\Delta(\Gamma)))$ is the vertex corresponding to $v_i(H_j)$ in H_j. Thus the number of incoming edges incident with y_j is $\lfloor |\Delta|/2 \rfloor + 1$. That is, each bus in $M(\Gamma, \Delta)$ is connected to $\lfloor |\Delta|/2 \rfloor + 1$ drivers. Hence, there are $|\Gamma|(\lfloor |\Delta|/2 \rfloor + 1)$ drivers. Therefore, by symmetry, each processor is connected to $\lfloor |\Delta|/2 \rfloor + 1$ drivers. By similar reasoning we can show that each processor is connected to $\lceil |\Delta|/2 \rceil$ receivers. □

The above theorem establishes an important attraction for *MBS*s. It is obvious that in the direct link network $N(\Gamma, \Delta)$, the number of input and output ports is $|\Delta|$ each. Hence the optimal *MBS* realization of a set of interconnection functions requires approximately one half the number of ports of that used in a direct link network realizing the same set of interconnection functions. In fact, it is well known that the number of ports per processor is a very limiting factor in constructing large size multiprocessor systems, particularly in a *VLSI* context [107]. To demonstrate the advantage of *MBS* in this regard, we simply note that in implementing cube functions, an *MBS* with N^2 processors would have equal number of ports per processor as a conventional hypercube with only N processors. The following corollary is a direct consequence the Theorems 4.8 and 4.9.

Corollary: The number of drivers and receivers per bus in the *MBS* $M(\Gamma, \Delta)$ are $\lfloor |\Delta|/2 \rfloor + 1$ and $\lceil |\Delta|/2 \rceil$, respectively. □

4.3.3 Number of Neighbors per Processor

In this section we derive an expression for the number of neighbors per processor in $M(\Gamma, \Delta)$ and show that it is much larger than that for the direct link network $N(\Gamma, \Delta)$.

Definition 4.8: Let P_1 and P_2 be two processors in an *MBS* such that they are connected to bus B via a driver and a receiver, respectively. Then P_2 is said to be a *neighbor* of P_1 *via* bus B. Also, we say that there is a direct path from P_1 to P_2 in the *MBS*. If the identity of bus B is irrelevant, we simply say that P_2 is a neighbor of P_1.

Lemma 4.4: For two processors γ_x and γ_y in $M(\Gamma, \Delta)$, there exists *at most* one bus γ_z, such that, γ_y is a neighbor of γ_x via bus γ_z.

Proof: Suppose that γ_y is a neighbor of γ_x with respect to two distinct buses γ_z and γ_w in $M(\Gamma, \Delta)$. Then, with our usual notation, vertices γ_x and γ_y must be in both of the

subgraphs H_z and H_w. Furthermore, γ_x must be the a^{th} and b^{th} vertices of H_z and H_w, respectively, for some *even* integers $a, b \leq |\Delta|$. Therefore,

$$\gamma_x = \gamma_z\delta_1\delta_2^{-1}\delta_3...\delta_{a-1}\delta_a^{-1} = \gamma_w\delta_1\delta_2^{-1}\delta_3...\delta_{b-1}\delta_b^{-1}.$$

Similarly, γ_y must be the c^{th} and d^{th} vertices of H_z and H_w respectively, for some *odd* integers $c, d \leq |\Delta|$. Therefore,

$$\gamma_y = \gamma_z\delta_1\delta_2^{-1}\delta_3...\delta_{c-1}^{-1}\delta_c = \gamma_w\delta_1\delta_2^{-1}\delta_3...\delta_{d-1}^{-1}\delta_d.$$

Thus, we can write

$$\gamma_x^{-1}\gamma_y = (\gamma_z\delta_1\delta_2^{-1}\delta_3...\delta_{a-1}\delta_a^{-1})^{-1}(\gamma_z\delta_1\delta_2^{-1}\delta_3...\delta_{c-1}^{-1}\delta_c) = Z, \text{ and}$$

$$\gamma_x^{-1}\gamma_y = (\gamma_w\delta_1\delta_2^{-1}\delta_3...\delta_{b-1}\delta_b^{-1})^{-1}(\gamma_w\delta_1\delta_2^{-1}\delta_3...\delta_{d-1}^{-1}\delta_d) = W.$$

Suppose $a > c$. Then $Z = (\delta_a\delta_{a-1}^{-1}...\delta_3^{-1}\delta_2\delta_1^{-1}\gamma_z^{-1})(\gamma_z\delta_1\delta_2^{-1}\delta_3...\delta_{c-1}^{-1}\delta_c) = \delta_a\delta_{a-1}^{-1}\delta_{a-2}...\delta_{c+2}^{-1}\delta_{c+1}$. Now assume $b < d$. Then $W = (\delta_b\delta_{b-1}^{-1}...\delta_3^{-1}\delta_2\delta_1^{-1}\gamma_w^{-1})(\gamma_w\delta_1\delta_2^{-1}\delta_3...\delta_{d-1}^{-1}\delta_d) = \delta_{b+1}\delta_{b+2}^{-1}...\delta_{d-2}\delta_{d-1}^{-1}\delta_d$. The generators δ_a, δ_{a-1}, δ_{a-2}, ..., δ_{c+2}, and δ_{c+1} used in word Z are all distinct and so are the generators δ_{b+1}, δ_{b+2}, ..., δ_{d-2}, δ_{d-1}, and δ_d used in word W. Let $\Delta_Z = \{\delta_a, \delta_{a-1}, \delta_{a-2}, ..., \delta_{c+2}, \delta_{c+1}\}$, and $\Delta_W = \{\delta_{b+1}, \delta_{b+2}, ..., \delta_{d-2}, \delta_{d-1}, \delta_d\}$. Since the generators in Δ are non redundant, $Z = W$ implies that $\Delta_Z = \Delta_W$. Thus, $\delta_{b+1} = \delta_{c+1}$ (δ_{b+1} and δ_{c+1} are the two generators with the least indices in sets Δ_Z and Δ_W, respectively). Hence, $b = c$, which is impossible since b is even and c is odd. Therefore, b must be $> d$, when $a > c$. In this case $W = \delta_b\delta_{b-1}^{-1}\delta_{b-2}...\delta_{d+2}^{-1}\delta_{d+1}$. Now $Z = W$ implies $a = b$. But this is true only when $\gamma_z = \gamma_w$. The same is true when $a < c$. Therefore, γ_y can be a neighbor of γ_x via only one bus. □

Theorem 4.10: The number of neighbors per processor in $M(\Gamma, \Delta)$ is $(\lceil|\Delta|/2\rceil)(\lfloor|\Delta|/2\rfloor + 1)$.

Proof: From Theorem 4.9, each processor is connected to $\lfloor |\Delta|/2 \rfloor + 1$ distinct buses via drivers. Also from the corollary to Theorem 4.9, each bus is connected to $\lceil |\Delta|/2 \rceil$ distinct processors via receivers. Thus the total number of neighbors is $(\lceil |\Delta|/2 \rceil)(\lfloor |\Delta|/2 \rfloor + 1)$. According to Lemma 4.4, all of these neighbors are distinct. □

The number of neighbors per processor in the direct link network is clearly $|\Delta|$. The above theorem highlights another big advantage of an *MBS* over its direct link counterpart. An *MBS* has a larger number of neighbors (very useful for efficient communication and broadcasting) than a direct link interconnection network realizing the same set of interconnection functions, yet it uses roughly half the ports per processor. For $|\Delta| = 6$, $M(\Gamma, \Delta)$ would have 12 neighbors per processor which is twice as many as that in the direct link counterpart $N(\Gamma, \Delta)$. The advantage becomes even larger as the number of interconnection functions $|\Delta|$ increases.

4.3.4 Diameter

Another very useful parameter associated with any interconnection network is the diameter. For an *MBS*, we give the following definition.

Definition 4.9: The *distance* from processor P_1 to processors P_2 in an *MBS* is the minimum number of buses to be used to transfer data from P_1 to P_2. The *diameter* of an *MBS* is the maximum distance from one processor to another, taken over all pairs of processors.

For a direct link interconnection network, the diameter corresponds to the usual graph theoretic definition [30]. If there is an edge from vertex x to vertex y in an *IFG* G, then processor y is a neighbor of processors x in the *MBS* which realizes G. Therefore, the diameter of an *MBS* is no more than that of the direct link interconnection network

realizing the same *IFG*. This is true irrespective of the *IFG* used. Unfortunately, nothing more can be said about the relative magnitudes of the diameters even if the *IFG* is symmetric. There is no general formula for the diameter of an arbitrary Cayley color graph. However, for the important *IFG* $G_{Q(n)}$, we shall obtain a very attractive result, namely, the diameter of $M_{Q(n)}$ is $\lfloor n/2 \rfloor + 1$.

It is well known that the diameter of $N_{Q(n)}$ is n. Elements of group $\Gamma_{Q(n)}$ are the 2^n binary vectors of length n. Elements of the generating set $\Delta_{Q(n)}$ are the n unit vectors of length n. In order to find the diameter of $M_{Q(n)}$, we will adhere to the following notation. The exclusive-or operation is represented by $\oplus$. Notation 0^i (1^i) is used to represent the string of zeros (ones) of length i. Binary string w will be written as $w_n w_{n-1} \ldots w_2 w_1$, where w_i is the i^{th} bit of w, $1 \le i \le n$. The number of 1's in string w is denoted by $\mathcal{H}(w)$. We will denote the diameter of $M_{Q(n)}$ by $diam(M_{Q(n)})$. First we will obtain an upper bound on $diam(M_{Q(n)})$.

Lemma 4.5: $diam(M_{Q(n)}) \le \lfloor n/2 \rfloor + 1$.

Proof: Let γ_a and γ_b be two arbitrary processors in $M_{Q(n)}$. We can express $\gamma_b = \gamma_a \oplus x$, for some binary string x of length n. Then, in $M_{Q(n)}$, there is a path from γ_a to γ_b of length $\mathcal{H}(x)$. If $\mathcal{H}(x) \le \lfloor n/2 \rfloor + 1$, then we have a path from γ_a to γ_b of length $\le \lfloor n/2 \rfloor + 1$. Now suppose that $\mathcal{H}(x) > \lfloor n/2 \rfloor + 1$. Then, we will consider two cases. Let $\gamma_c = \gamma_a \oplus 1^n$.

Case 1: n is odd.

In this case, processor γ_c is connected to bus γ_a via a receiver. Therefore, there is a direct path from γ_a to γ_c in $M_{Q(n)}$. (Note that processor γ_a is connected to bus γ_a via a driver.) Also there is a path from γ_c to γ_b of length $n - \mathcal{H}(x)$. Therefore, there is a path from γ_a

to γ_b of length $n - \mathcal{H}(x) + 1$. Since $\mathcal{H}(x) > \lfloor n/2 \rfloor + 1$ and n is odd by the hypothesis, we have,

$$\begin{aligned} n - \mathcal{H}(x) + 1 &< n - (\lfloor n/2 \rfloor + 1) + 1, \text{ that is,} \\ n - \mathcal{H}(x) + 1 &\leq n - (\lfloor n/2 \rfloor + 1) \\ &= n - ((n-1)/2 + 1) \\ &= (n-1)/2 \\ &= \lfloor n/2 \rfloor. \end{aligned}$$

Case 2: n is even.

In this case processor γ_c is not connected to bus γ_a with a receiver. Let $\gamma_d = \gamma_a \oplus 01^{n-1}$. Then, since $n - 1$ is odd, there is a direct path from γ_a to γ_d in $M_{Q(n)}$. Since γ_c and γ_d are adjacent in $G_{Q(n)}$, there is a direct path from γ_d to γ_c. Therefore, there is a path of length $n - \mathcal{H}(x) + 2$ from γ_a to γ_b. Since n is even and $\mathcal{H}(x) > \lfloor n/2 \rfloor + 1$ by the hypothesis, we have,

$$\begin{aligned} n - \mathcal{H}(x) + 2 &< n - (\lfloor n/2 \rfloor + 1) + 2, \text{ that is,} \\ n - \mathcal{H}(x) + 2 &\leq n - (\lfloor n/2 \rfloor + 1) + 1 \\ &= n - (n/2 + 1) + 1 \\ &= n/2 \\ &= \lfloor n/2 \rfloor. \end{aligned}$$

Therefore, in $M_{Q(n)}$, there always exists a path of length less than or equal to $\lfloor n/2 \rfloor + 1$ from processor γ_a to processor γ_b. Since γ_a and γ_b are arbitrary processors of $M_{Q(n)}$, the lemma follows. □

Next we will obtain a lower bound on the diameter of $M_{Q(n)}$. For that purpose, some new notation will be introduced.

Definition 4.10: A binary string of the form $s = 0^i 1^{2k+1} 0^{n-i-2k-1}$ will be called an *SCOL* (String with Consecutive ones of Odd Length). Also define, $\eta(s) = \max_j \{j: s_j = 1\}$, $\mu(s) = \min_j \{j: s_j = 1\}$, and $\iota(s) = \eta(s) - \mu(s) + 1$.

Example 4.4: The strings 001000, 011100, 111000, and 000001 are *SCOL*s because all 1's in each string are consecutive and they form substrings of odd length. However, the strings 000011, 000000, and 100011 are not *SCOL*s. For the *SCOL* 011100, $\eta(011100) = 5$, $\mu(011100) = 3$, and $\iota(011100) = 3$. Notice that, $\iota(s)$ represents the length of the substring of s with consecutive 1's.

Lemma 4.6: Processor γ_b is a neighbor of processor γ_a in $M_{Q(n)}$ if and only if $\gamma_b = \gamma_a \oplus s$, for some *SCOL* s.

Proof: First suppose that processors γ_a and γ_b are connected to bus γ_c via a driver and a receiver, respectively. Then, vertices γ_a and γ_b must both belong to the same subgraph H_c. Furthermore, γ_a must be the g^{th} vertex of H_c for an even integer g, and γ_b must be the h^{th} vertex of H_c for an odd integer h. Therefore, $\gamma_a = \gamma_c \oplus 0^{n-g}1^g$ and $\gamma_b = \gamma_c \oplus 0^{n-h}1^h$. If $h > g$, then $\gamma_b = \gamma_a \oplus 0^{n-h}1^{h-g}0^g$. Since $h - g$ is odd, $0^{n-h}1^{h-g}0^g$ is an *SCOL*. The same result holds when $h < g$.

Now suppose that $\gamma_b = \gamma_a \oplus s$ for some *SCOL* s. Let $s = 0^{n-m-l}1^l0^m$, where l is an odd integer. We will consider two cases.

Case 1: m is even.

Let $\gamma_c = \gamma_a \oplus 0^{n-m}1^m$. Then, $\gamma_a = \gamma_c \oplus 0^{n-m}1^m$. Therefore, γ_a is the m^{th} vertex of the subgraph H_c. Since m is even by the hypothesis, processor γ_a is connected to bus γ_c via a driver in $M_{Q(n)}$. Now, $\gamma_b = \gamma_a \oplus s = \gamma_a \oplus 0^{n-m-l}1^l0^m = \gamma_c \oplus 0^{n-m}1^m \oplus 0^{n-m-l}1^l0^m = \gamma_c \oplus 0^{n-m-l}1^{m+l}$. Therefore γ_b is the $(m + l)^{\text{th}}$ vertex of subgraph H_c.

Since $m + l$ is odd (m is even and l is odd), processor γ_b is connected to bus γ_c via a receiver. Hence γ_b is a neighbor of γ_a via bus γ_c.

Case 2: m is odd.

Let $\gamma_c = \gamma_a \oplus 0^{n-m-l}1^{m+l}$. Since $m + l$ is even, processor γ_a is connected to bus γ_c via a driver. Now, $\gamma_b = \gamma_a \oplus s = \gamma_c \oplus 0^{n-m}1^m$, and therefore, processor γ_b is connected to bus γ_c via a receiver. Hence, γ_b is a neighbor of γ_b. □

Lemma 4.7: Let $w = s^1 \oplus s^2 \oplus ... \oplus s^\alpha$ where each s^i, $1 \le i \le \alpha$, is an *SCOL*. Let $m = \max\{ j : w_j = 1\}$. Then $w = t^1 \oplus t^2 \oplus ... \oplus t^\beta$, where t^i, $1 \le i \le \beta \le \alpha$, is an *SCOL* with the property $\eta(t^i) \le m$.

Proof: Let $S = \{s^1, s^2, ..., s^\alpha\}$. In the proof we will construct $T = \{t^1, t^2, ..., t^\beta\}$ from S. Divide S into three disjoint subsets S_1, S_2, and S_3 as follows.

$S_1 = \{s^i : \eta(s^i) \le m\}$

$S_2 = \{s^i : \eta(s^i) > m \ge \mu(s^i)\}$

$S_3 = \{s^i : \mu(s^i) > m\}$

In constructing T, we will retain all the elements in S_1, modify all the elements in S_2, and discard all the elements in S_3. Let $T_1 = S_1$. To modify the elements in S_2 and form T_2, we make the following claim.

Claim: If S_3 is empty, then the number of elements s^i in S_2 with the same value of $\eta(s^i)$ is even.

Proof: Let s^x be an element in S_2 such that $\eta(s^x)$ is the largest member in $\{\eta(s^i) : s^i \in S_2\}$. Since $w_j = 0$ for $m < j \le n$, there should be an even number of elements in $\{s^i : s^i \in S, \eta(s^i) = \eta(s^x)\}$. By the hypothesis of the claim, S_3 is empty. Therefore, there should be an even number of elements in $\{s^i : s^i \in S_2$,

$\eta(s^i) = \eta(s^x)\}$. The exclusive-or summation of all these *SCOL*s will yield zeros in bit positions $m + 1$ through n. Therefore, we can disregard all those *SCOL*s and similarly prove that there is an even number of elements in $\{s^i : s^i \in S, \eta(s^i) = \eta(s^y)\}$, where $\eta(s^y)$ is the second largest member in $\{\eta(s^i) : s^i \in S_2\}$. By repeated application of this argument, the claim follows.

For each $s^i \in S_2$ such that $(\eta(s^i) - m)$ is even, let $t^i = s^i \oplus 0^{n-\eta(s^i)}\, 1^{\eta(s^i)-m}\, 0^m$. Since $(\eta(s^i) - m)$ is even, t^i is also an *SCOL*. Furthermore, $\eta(t^i) = m$ and the m^{th} bits of s^i and t^i are the same. Similarly, for each $s^i \in S_2$ such that $(\eta(s^i) - m)$ is odd, let $t^i = s^i \oplus 0^{n-\eta(s^i)}\, 1^{\eta(s^i)-m+1}\, 0^{m-1}$. Since $(\eta(s^i) - m + 1)$ is even, t^i is also an *SCOL*. Furthermore, $\eta(t^i) = m - 1$ and the m^{th} bits of s^i and t^i are different. Now we construct T_2 by letting $T_2 = \{t^i : s^i \in S_2\}$.

Let v be the string obtained by exclusive-or summation of the elements in $T_1 \cup T_2$. If $v = w$, then we have the required set of *SCOL*s $T = T_1 \cup T_2$. Clearly, $\beta = |T| = |T_1| + |T_2| = |S_1| + |S_2| \leq |S| = \alpha$. If $v \neq w$, then they must differ only in bit position m. Therefore, we can make $v = w$ by letting $T = T_1 \cup T_2 \cup \{0^{n-m}10^{m-1}\}$. If that is the case, we need only to prove that $\beta = |T| \leq |S| = \alpha$. By constructing t^i from s^i, we altered the m^{th} bit of the *SCOL* s^i only when $(\eta(s^i) - m)$ is odd. Therefore, for the m^{th} bits of v and w to be different, the number of elements $s^i \in S_2$ with an odd value of $(\eta(s^i) - m)$ must be odd. But according to the previous claim, unless S_3 is non empty, the number of elements $s^i \in S_2$ with an odd value of $(\eta(s^i) - m)$ is even. Therefore, $v \neq w$ implies that S_3 is non empty. Therefore, $|S| \geq |S_1| + |S_2| + 1$, implying $|T| \leq |S|$. Hence, the lemma follows. □

Lemma 4.8: Let s be an *SCOL* of length n. Then $s \oplus 0^{n-\eta(s)}110^{\eta(s)-2}$ is also an *SCOL* whose $(\eta(s))^{th}$ bit is equal to zero.

Proof: For clarity denote $\eta(s)$ by η. First suppose that $\iota(s) = 1$. Then, $s = 0^{n-\eta}10^{\eta-1}$. Therefore,

$$s \oplus 0^{n-\eta}110^{\eta-2} = 0^{n-\eta}10^{\eta-1} \oplus 0^{n-\eta}110^{\eta-2}$$
$$= 0^{n-\eta+1}10^{\eta-2}.$$

Clearly, $0^{n-\eta+1}10^{\eta-2}$ is an *SCOL* and its η^{th} bit is equal to zero.

Now suppose that $\iota(s) = l > 1$. Then, $s = 0^{n-\eta}1^{l}0^{\eta-l}$. Therefore,

$$s \oplus 0^{n-\eta}110^{\eta-2} = 0^{n-\eta}1^{l}0^{\eta-l} \oplus 0^{n-\eta}110^{\eta-2}$$
$$= 0^{n-\eta+2}1^{l-2}0^{\eta-l}.$$

Clearly, $0^{n-\eta+2}1^{l-2}0^{\eta-l}$ is an *SCOL* and its η^{th} bit is equal to zero. □

Definition 4.11: Let w be a binary string of length n. Then $L(w)$ is the minimum number of *SCOL*s whose exclusive-or summation produces w.

Lemma 4.9: Let w be an arbitrary string of length n such that $m = max\{j : w_j = 1\} \leq n - 2$. Let $w' = w \oplus 0^{n-m-2}10^{m+1}$. Then $L(w') \geq L(w) + 1$.

Proof: Let S be a set of *SCOL*s whose exclusive-or summation yields w'. According to Lemma 4.7, without loss of generality, we can assume that $\eta(s^i) \leq m + 2$ for every element $s^i \in S$. Since $w'_{m+2} = 1$, at least one *SCOL* s^i in S must be such that $\eta(s^i) = m + 2$. Suppose that $0^{n-m-2}1^{x}0^{m-x+2}$ and $0^{n-m-2}1^{y}0^{m-y+2}$ are two such *SCOL*s. Now,

$$0^{n-m-2}1^{x}0^{m-x+2} \oplus 0^{n-m-2}1^{y}0^{m-y+2}$$
$$= 0^{n-m-2}1^{x}0^{m-x+2} \oplus 0^{n-m-2}1^{y}0^{m-y+2} \oplus (0^{n-m-2}110^{m} \oplus 0^{n-m-2}110^{m})$$
$$= (0^{n-m-2}1^{x}0^{m-x+2} \oplus 0^{n-m-2}110^{m}) \oplus (0^{n-m-2}1^{y}0^{m-y+2} \oplus 0^{n-m-2}110^{m})$$

Therefore, according to Lemma 4.8, $0^{n-m-2}1^x0^{m-x+2}$ and $0^{n-m-2}1^y0^{m-y+2}$ can be replaced by two *SCOL*s whose $(m + 2)^{nd}$ bits are equal to 0. Every pair of *SCOL*s in S with $(m + 2)^{nd}$ bits equal to 1 can be similarly replaced. Therefore, we can assume that S contains only one *SCOL* whose $(m + 2)^{nd}$ bit is equal to 1. Let that *SCOL* be

$$s^1 = 0^{n-m-2}1^x0^{m-x+2}.$$

Suppose that $x > 1$. Then the $(m + 1)^{st}$ bit of s^1 is also equal to 1. Since $w'_{m+1} = 0$, there should be an element of S whose $(m + 1)^{st}$ bit is equal to 1. Let that element be

$$s^2 = 0^{n-m-1}1^z0^{m-z+1}$$

Suppose that $z > x - 1$. Then,

$$s^1 \oplus s^2 = 0^{n-m-2}1^x0^{m-x+2} \oplus 0^{n-m-1}1^z0^{m-z+1}$$

$$= 0^{n-m-2}10^x1^{z-x+1}0^{m-z+1}$$

$$= 0^{n-m-2}10^{m+1} \oplus 0^{n-m+x-2}1^{z-x+1}0^{m-z+1}.$$

Since both x and z are odd, $z - x + 1$ is also odd. Therefore, $0^{n-m+x-2}1^{z-x+1}0^{m-z+1}$ is an *SCOL*. Thus, the two *SCOL*s s^1 and s^2 can be replaced by $0^{n-m-2}10^{m+1}$ and another *SCOL* with the $(m + 2)^{nd}$ bit equal to 0. The same result holds when $z < x - 1$. Therefore, we can assume that the only *SCOL* in S with bit $(m + 2)$ equal to 1 is the string $0^{n-m-2}10^{m+1}$. Now, since $w'_{m+1} = 0$, there should be an even number of elements of S whose $(m + 1)^{st}$ bits are equal to 1. Let s^3 and s^4 be such a pair of *SCOL*s. We have, $s^3 \oplus s^4 = (s^3 \oplus 0^{n-m-1}110^{m-1}) \oplus (s^4 \oplus 0^{n-m-1}110^{m-1})$. According to Lemma 4.8, $(s^3 \oplus 0^{n-m-1}110^{m-1})$ and $(s^4 \oplus 0^{n-m-1}110^{m-1})$ are *SCOL*s with their $(m + 1)^{st}$ bits equal to 0. Therefore, every pair of *SCOL*s in S with $(m + 1)^{st}$ bits equal to 1 can be replaced by another pair of *SCOL*s whose $(m + 1)^{st}$ bits are equal to 0. Therefore, we can safely assume that S consists of two disjoint subsets S_1 and S_2, where

every element of S_1 has its $(m + 2)^{nd}$ and $(m + 1)^{st}$ bits equal to zero and S_2 contains the single *SCOL* $0^{n-m-2}10^{m+1}$. Furthermore, $|S| = |S_1| + |S_2| = |S_1| + 1$.

Suppose that $|S| < L(w) + 1$. Then, $|S_1| < L(w)$. Since $w' = w \oplus 0^{n-m-2}10^{m+1}$, the exclusive-or summation of the elements of S_1 yields w. Therefore, we could construct a smaller set of *SCOL*s whose exclusive-or summation produces w. This violates the definition of $L(w)$. Therefore, $|S| \geq L(w) + 1$. That is, $L(w') \geq L(w) + 1$. □

Lemma 4.10: $diam(M_{Q(n)}) \geq \lfloor n/2 \rfloor + 1$.

Proof: When $n = 1$ or 2, the truth of the lemma can be easily verified. Let $n > 2$. Let γ_a be an arbitrary processor in the *MBS* $M_{Q(n)}$. Our proof consists of finding another processor γ_b which is at a distance of at least $\lfloor n/2 \rfloor + 1$ from γ_a. For that purpose, we consider two cases.

Case 1: n is even.

Let γ_b be the processor in $M_{Q(n)}$ such that $\gamma_b = \gamma_a \oplus (10)^{n/2-1}11$. Since $0^{n-2}11$ is not an *SCOL*, $L(0^{n-2}11) \geq 2$. Therefore, according to Lemma 4.9, $L(0^{n-4}1011) \geq 2 + 1$.

By repeatedly applying Lemma 4.9, we get

$L((10)^{n/2-1}11) \geq 2 + (n/2 - 1)$

$= n/2 + 1$

$= \lfloor n/2 \rfloor + 1$.

Case 2: n is odd.

Let γ_b be the processor in $M_{Q(n)}$ such that $\gamma_b = \gamma_a \oplus 0(10)^{(n-1)/2-1}11$. Again, by Lemma 4.9,

$0(10)^{(n-1)/2-1}11 \geq 2 + ((n - 1)/2 - 1)$

$= (n - 1)/2 + 1$

$= \lfloor n/2 \rfloor + 1$.

Therefore, in both cases, to construct γ_b from γ_a at least $\lfloor n/2 \rfloor + 1$ *SCOL*s are necessary. Hence, according to Lemma 4.6, the distance from γ_a to γ_b is at least $\lfloor n/2 \rfloor + 1$. Therefore, $diam(M_{Q(n)}) \geq \lfloor n/2 \rfloor + 1$. □

Theorem 4.11: $diam(M_{Q(n)}) = \lfloor n/2 \rfloor + 1$.

Proof: Directly follows from Lemma 4.5 and Lemma 4.10. □

This shows the existence of an *MBS* which emulates the *SIMD* hypercube with far superior features. The diameter of the *MBS* emulating the hypercube is half that of the hypercube while the number of ports per processor in the *MBS* is also half that of the hypercube.

So far we have shown how to perform an optimal color partition of a vertex symmetric *IFG* with non redundant generating sets. We have also shown some attractive features of the resulting *MBS*. Next we consider regular *IFG*s. As we show, not only vertex symmetric *IFG*s, but also regular *IFG*s can be optimally color partitioned in polynomial time if certain conditions are satisfied.

4.4 Optimal Color Partition of a Regular *IFG*

In Section 2.5, we introduced the notion of regular *IFG*s (see Definition 2.9). It is straightforward to see that, in a regular *IFG*, every vertex has exactly one incoming edge from each color (see the proof of Lemma 4.1). Obviously, every vertex symmetric *IFG* is regular. But the opposite is not true. If an *IFG* is regular and belongs to class $\mathscr{E}$, then we can find an optimal color partition in polynomial time. The following algorithm, which is based on the optimal color partition of a vertex symmetric *IFG*, serves that purpose.

Algorithm 4.1

INPUT : Regular *IFG* G, belonging to class $\mathscr{E}$, with c colors, where $V(G) = \{v_1, v_2, ..., v_n\}$.
OUTPUT: An optimal color partition $\{E_1, E_2, ..., E_n\}$

```
begin
    for j = 1 to n do
    begin
        E_j = { };
        w = v_j;
        for r = 1 to c do
        begin
            if r is odd then
            begin E_j = E_j ∪ {(w, w_r^in)}; w = w_r^in; end;
                /* w_r^in is the vertex which is incident with the
                edge of color r directed away from w. */
            if r is even then
            begin E_j = E_j ∪ {(w_r^out, w)}; w = w_r^out; end;
                /* w_r^out is the vertex which is incident with the
                edge of color r directed towards w. */
        end (r loop);
    end (j loop);
end.
```

Theorem 4.12: Algorithm 4.1 outputs an optimal color partition of G.

Proof: Since each vertex of G has exactly one outgoing (incoming) edge belonging to each color, at the end of the execution of the algorithm, each subset E_j would have c edges, one from each color. The rth edge in E_j is of color r, for $1 \le r \le c$. Furthermore, according to the algorithm, edges in the set E_j form an alternately oriented path (since G is assumed to belong to class $\mathscr{E}$, no alternately oriented cycles exist). Thus, E_j induces $c + 1$ vertices. Let those vertices be $u_1^j, u_2^j, ..., u_c^j$, and u_{c+1}^j. The edge with end vertices u_r^j and u_{r+1}^j would be of color r, for $1 \le r \le c$. The orientation of that edge will depend on whether r is even or odd. According to the algorithm, $u_1^j = v_j$, for $1 \le j \le n$. In other words, vertices $u_1^j, u_1^2, ..., u_1^n$ are distinct. Each vertex has exactly one incoming edge and one outgoing edge of color 1. Therefore, vertices $u_2^1, u_2^2, ..., u_2^n$ are all distinct.

Similarly, we can show that, for every r ($1 \le r \le c$), vertices u_r^1, u_r^2, ..., u_r^n are all distinct.

We need to prove that the subsets E_1 through E_n as outputted by the algorithm are edge disjoint. Suppose on the contrary that two (distinct) subsets E_x and E_y contain a common edge. Let $(u_r^x, u_{r+1}^x) \in E_x$ and $(u_k^y, u_{k+1}^y) \in E_y$ represent the same edge in G. Then they must be of the same color, and therefore, $k = r$. But as we have just shown, u_r^x and u_r^y are distinct vertices. Hence, (u_r^x, u_{r+1}^x) and (u_k^y, u_{k+1}^y) cannot be the same edge of G. Thus there are no common edges in E_x and E_y. Therefore, subsets $E_1, E_2, ..., E_n$ are disjoint. Clearly, edges in $E_1, E_2, ..., E_n$ cover all edges in G. Thus $\{E_1, E_2, ..., E_n\}$ is a color partition of G. Each subset E_j induces an alternately oriented path in G. Therefore, $|J(E_j)| = c + 1$, $1 \le j \le n$. Thus $\sum_{j=1}^{n} |J(E_j)| = n(c + 1) = nc + c = |E(G)| + \beta(G)$. Furthermore, G is in class $\mathscr{E}$. Therefore, by Theorem 4.4, $\{E_1, E_2, ..., E_n\}$ is an optimal color partition. □

Each subset E_j of the output of the algorithm induces an alternately oriented path of length c such that the first vertex is incident with an outgoing edge. Therefore, in the corresponding *MBS*, every bus is connected to the same number of drivers and the same number of buses. It is also easy to verify that every processor is connected to the same number of drivers and receivers. Thus the resulting *MBS* is regular. But other features of the *MBS* (such as symmetry and diameter) may not be as attractive as those obtained for vertex symmetric *IFG*s.

The time complexity of Algorithm 4.1 can be determined as follows. The outer loop of Algorithm 4.1 is executed n times, while the inner loop is executed c times.

Therefore, the time complexity of the above algorithm is $\Theta(n.c) = \Theta(|E(G)|)$. To demonstrate the operation of the algorithm, we provide the following example.

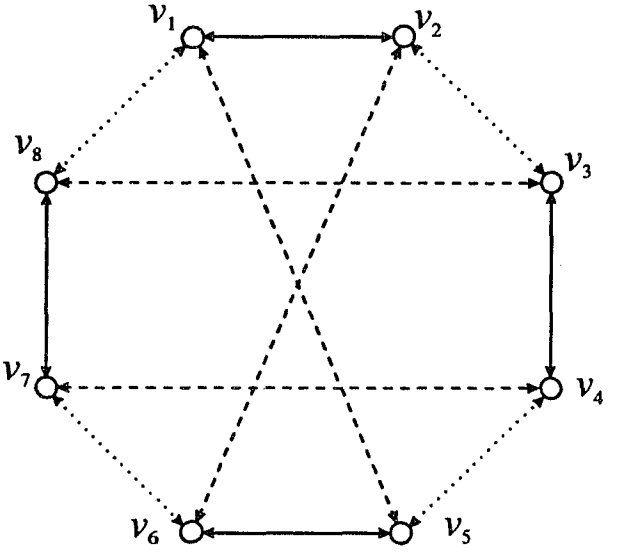


Figure 4.5: A regular *IFG* G_6.

Example 4.5: Consider the regular *IFG*, denoted G_6, shown in Figure 4.5 with eight vertices and three colors. Solid, dotted, and broken lines represent edges of colors 1, 2, and 3, respectively. It can be easily verified that G_6 is not vertex symmetric; therefore, it is not a Cayley color graph. Figure 4.6 shows the eight induced subgraphs corresponding to the optimal color partition obtained by using Algorithm 4.1. Figure 4.7 shows the corresponding *MBS*. Notice that it is regular.

4.5 Optimal Color Partition of $C_\Delta(\Gamma)$ when Δ is Redundant

So far we have studied Cayley color graphs with non redundant generating sets. Even though this is the case for almost all the situations of interest, we are obliged to consider how to perform an optimal color partition of $C_\Delta(\Gamma)$ when Δ is redundant for completeness. The purpose of this section is to briefly outline the difficulties faced when

we try to find an optimal color partition of a Cayley color graph associated with a group and a redundant generating set.

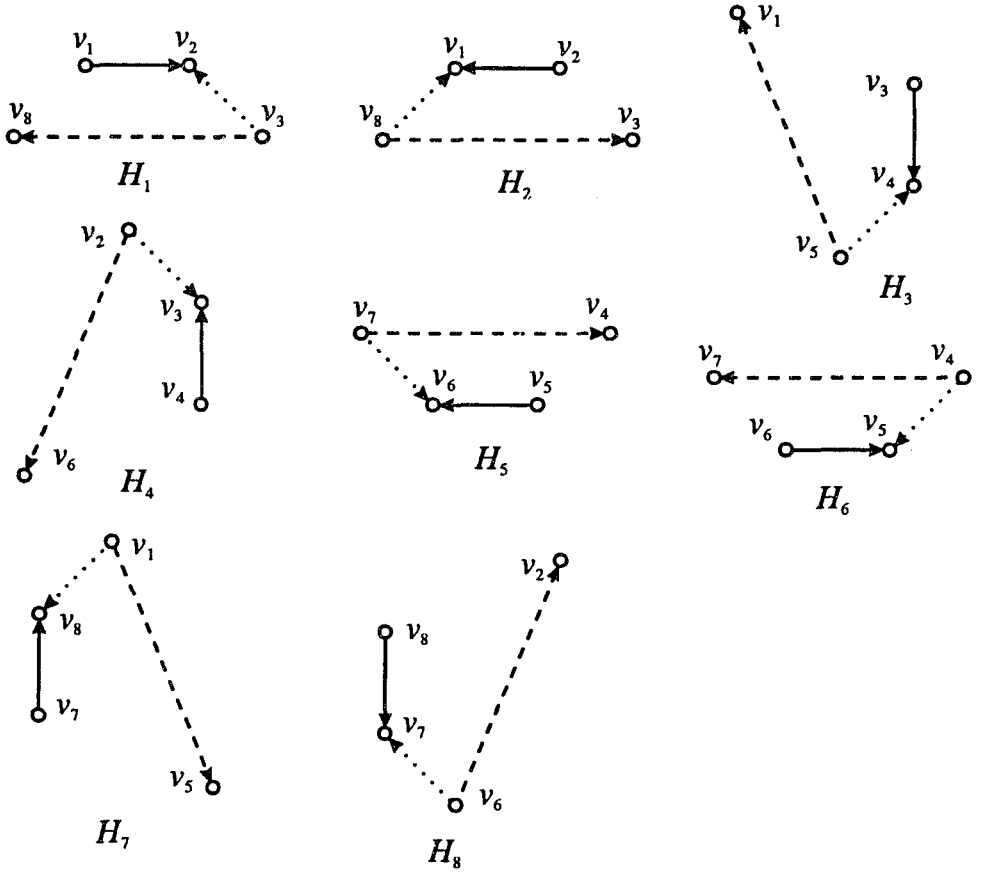


Figure 4.6: Color partition of G_6 by Algorithm 4.1.

In finding an optimal color partition for $C_\Delta(\Gamma)$, we used the non redundant nature of Δ to show that $C_\Delta(\Gamma)$ belongs to class $\mathscr{E}$. In fact, if $C_\Delta(\Gamma)$ belongs to class $\mathscr{E}$, then, even if Δ is redundant, obtaining an optimal color partition can be done exactly the same way as the non redundant case. All the results obtained in this chapter will still be valid. However, when Δ is redundant, $C_\Delta(\Gamma)$ may not belong to the class $\mathscr{E}$. In such a case, the color partition $\phi_{\Gamma\Delta}$ given in Definition 4.5 may not be optimal.

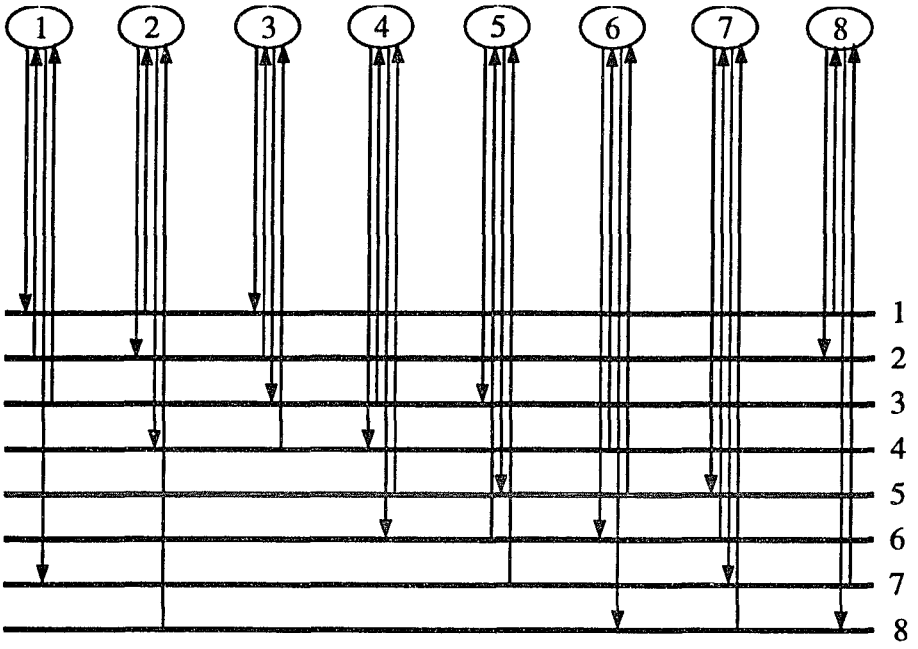


Figure 4.7: *MBS* corresponding to partition shown in Figure 4.6.

Theorem 4.6 is valid whether the generating set is redundant or not. Thus, what we need to do is to find only one subset E_1 of edges with distinct colors such that $|J(E_1)|$ is minimum. The remaining subsets can be easily determined by using the mapping given in the proof of Theorem 4.6. When $C_\Delta(\Gamma)$ belongs to class $\mathscr{E}$, the minimum number of vertices induced by any $|\Delta|$-element subset of distinct color edges is $|\Delta| + 1$. The color partition $\phi_{\Gamma,\Delta}$ divides the edge set of $C_\Delta(\Gamma)$ into such subsets E_j with the additional property that $|J(E_j)| = |\Delta| + 1$, $1 \leq j \leq |\Gamma|$. Recall that, for color partition $\phi_{\Gamma,\Delta}$, edges in each subset are selected such that the edge of color r and the edge of color $r + 1$ are adjacent (and compatible). This ordering of edges is not mandatory. Even if we change the ordering of the colors, the corresponding color partition would be still optimal. But these facts may not be true when $C_\Delta(\Gamma)$ does not belong to class $\mathscr{E}$. In the following two

examples, we show how the optimality is affected when the Cayley color graph does not belong to class $\mathscr{E}$.

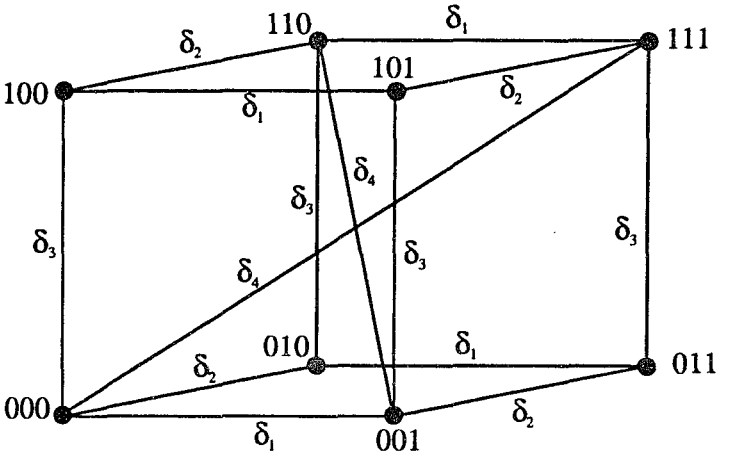


Figure 4.8: Cayley color graph G_7 with redundant Δ.

Example 4.6: Consider the *IFG*, denoted G_7, shown in Figure 4.8. It is the Cayley color graph associated with the group $\Gamma_{(Q(3))} = \{000, 001, 010, 011, 100, 101, 110, 111\}$ and its generating set $\Delta_1 = \{001, 010, 100, 111\}$. We will denote the generators 001, 010, 100, and 111 by δ_1, δ_2, δ_3, and δ_4, respectively. Not all edges associated with generator δ_4 are shown in Figure 4.8 for clarity. Since every element in Δ is its self inverse, we use undirected edges to represent bidirectional edges. Note that the *IFG* G_7 shown in Figure 4.8 corresponds to the folded hypercube of dimension three [91]. Figure 4.9 shows subgraph H_{000} of G_7 induced by subset E_{000} of color partition $\phi_{\Gamma_{Q(3)},\Delta_1}$. Clearly, H_{00} contains $4 = |\Delta|$ vertices instead of $|\Delta| + 1$ when Δ is non redundant. Furthermore, $|J(E_{000})| = 4$. One can be easily convinced that the minimum value of $|J(E')|$ for any 4-element subset E' of distinct color edges of G_7 is 4. Therefore, despite the fact that Cayley color graph G_7 does not belong to class $\mathscr{E}$, color partition $\phi_{\Gamma,\Delta}$ is optimal for this example.

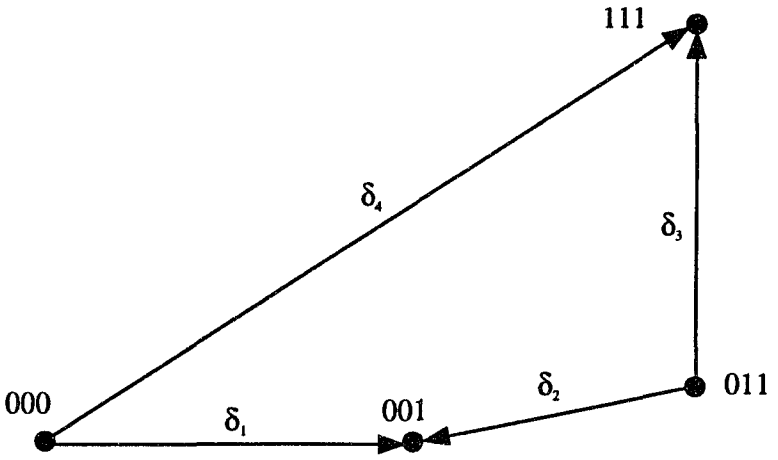


Figure 4.9: Induced subgraph H_{000} of G_7.

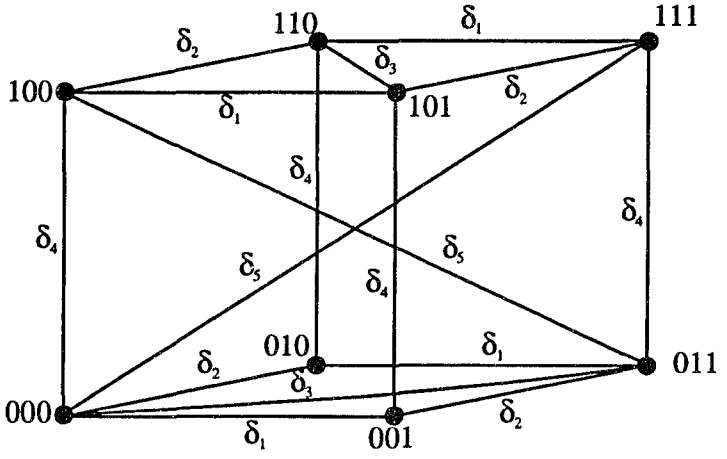


Figure 4.10: Cayley color graph G_8 with redundant Δ.

The above example shows us that even if $C_\Delta(\Gamma)$ does not belongs to class $\mathscr{E}$, the color partition $\phi_{\Gamma,\Delta}$ may be optimal. In the next example, we show a situation, where the optimality of color partition $\phi_{\Gamma,\Delta}$ depends on the order of the generators.

Example 4.7: Consider the generating set $\Delta_2 = \{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5\}$ for group $\Gamma_{Q(3)}$, where $\delta_1 = 001$, $\delta_2 = 010$, $\delta_3 = 011$, $\delta_4 = 100$, and $\delta_5 = 111$. Figure 4.10 shows the corresponding Cayley color graph, denoted by G_8. Not all edges associated with

generators δ_3 and δ_5 are shown for clarity. Figure 4.11 shows the subgraph H_{000} induced by subset E_{000} of color partition $\phi_{\Gamma_{Q(3)},\Delta_2}$. Clearly, $|J(E_{000})| = 6$. Now, consider the induced subgraph H'_{000} shown in Figure 4.12, which was obtained by reordering the generators as δ_1, δ_2, δ_4, δ_5, and δ_3. From Figure 4.12 it is clear that $|J(H'_{000})| = 5$. Thus, the value of $|J(H_j)|$ depends on the order of the generators used in the partition $\phi_{\Gamma_{Q(3)},\Delta_2}$. It is easy to verify that, for any 5-element subset E' of distinct color edges of an *IFG*, the minimum value of $|J(E')|$ is 5. Therefore, color partition $\phi_{\Gamma_{Q(3)},\Delta_2}$ with the generators taken in the order δ_1, δ_2, δ_4, δ_5, δ_3 is optimal, whereas that with the generators taken in the order δ_1, δ_2, δ_3, δ_4, δ_5 is not optimal.

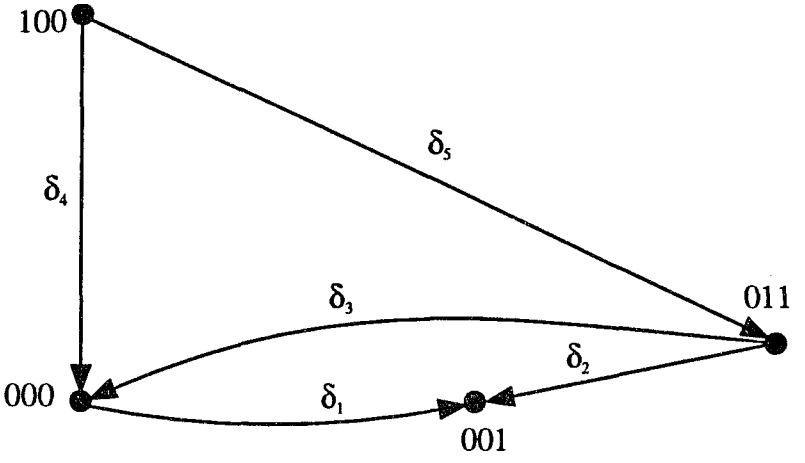


Figure 4.11: Induced subgraph H_{000} of G_8.

When $C_\Delta(\Gamma)$ does not belong to class $\mathscr{E}$, the order of the generators resulting in an optimal color partition $\phi_{\Gamma,\Delta}$ would not be obvious. In fact, a $|\Delta|$-element subset E_1 of distinct color edges with minimum $|J(E_1)|$ may not correspond to color partition $\phi_{\Gamma,\Delta}$ irrespective of the ordering of the generators used. When the size of $C_\Delta(\Gamma)$ is not very

large, the best E_1 can always be found by trial and error. Usually, $|\Delta|$ is much smaller than $|\Gamma|$. Therefore, utilizing an exhaustive search method to find E_1 with minimum $|J(E_1)|$ would not be time consuming. Finding of a subset E_1 with minimum $|J(E_1)|$ is beyond the scope of this dissertation and, hence, will not be addressed any further.

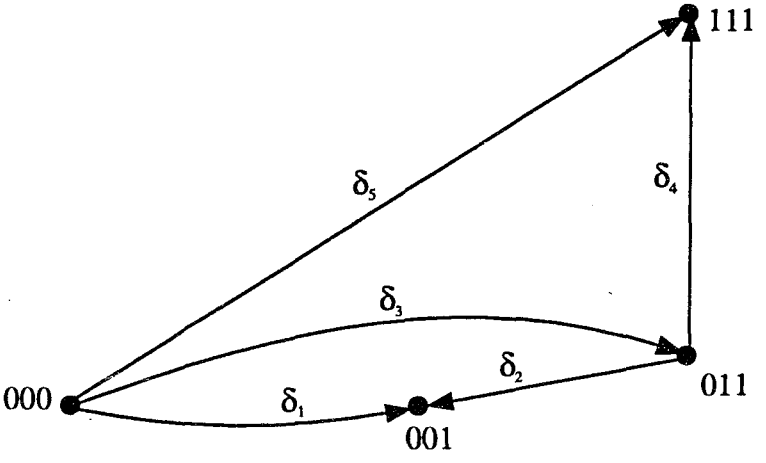


Figure 4.12: Induced subgraph H'_{000} of G_8.

CHAPTER 5

FAULT TOLERANCE

One of the key issues of a parallel processing system is its fault tolerance. Fault tolerance of a parallel processing system reflects its ability to function, possibly with degraded performance, under the failure of certain components. The likelihood of one or more components failing in a parallel processor system increases as the number of components increases. Therefore, fault tolerance is one of the richly addressed subjects in the area of parallel processing [4], [6], [7], [114], [117], [129]. Some systems such as hypercube [119], star [5], mesh [13], and conventional multiple bus systems [109] are inherently fault tolerant. They can continue functioning as smaller networks even when certain links and nodes are failed, provided that the network can detect and isolate the fault. Some other systems such as the linear array [140] and the generalized cube network [128] are not fault tolerant. However, they can be made fault tolerant by adding some redundant components. For example, the generalized cube network can be made fault tolerant by adding an extra stage [1], [4].

In this chapter, we study the fault tolerance of the *MBS*s corresponding to vertex symmetric *IFG*s. That is, we study the fault tolerance of $M(\Gamma, \Delta)$ for a group Γ and its generating set Δ, where Δ is assumed to be non redundant. We will analyze the behavior of $M(\Gamma, \Delta)$ in the case of single bus failure, single interface failure, and single processor failure. Even though the analysis carried out in this chapter can be extended to multiple component failures, we do not attempt to do so in this dissertation. If an *MBS* can execute its source algorithm (with performance degradation) although a fault exists, we say that the *MBS* sustains that fault. On the other hand, if the *MBS* cannot execute its source

algorithm in the case of a fault, then we say that the *MBS* does not sustain the fault. Our design criteria were directed towards constructing a minimal cost *MBS* realizing a given source algorithm. An *MBS* designed under such criteria will always suffer some performance degradation when a component fails, if it sustains the failure. Otherwise, one could have constructed the *MBS* without using the failed component.

We have already seen many attractive features of the *MBS* $M(\Gamma, \Delta)$. The fault tolerance capabilities of $M(\Gamma, \Delta)$ will further enhance the applicapability of the *MBS* as an algorithmically specialized parallel architecture. It is quite interesting to notice that although $M(\Gamma, \Delta)$ is the minimal component *MBS* which can realize the given *IFG*, it still has certain fault tolerant capabilities. In this chapter we analyze under what conditions the *MBS* $M(\Gamma, \Delta)$ sustains a single bus failure, single interface failure, or a single processor failure. We specifically show the following in this chapter.

(1) $M(\Gamma, \Delta)$ can sustain a single bus failure if and only if $|\Delta| > 2$.

(2) $M(\Gamma, \Delta)$ can sustain a single driver failure if and only if $|\Delta| > 1$.

(3) $M(\Gamma, \Delta)$ can sustain a single receiver failure if and only if $|\Delta| > 2$.

(4) $M(\Gamma, \Delta)$ can sustain a single processor failure if and only if $|\Delta| > 1$.

Clearly, for most practical cases, $|\Delta| \gg 1$ and thus the *MBS* will normally sustain any of the above failures. Also, when the *MBS* sustains the above faults, we address the issue of performance degradation under each fault condition. Furthermore, when $M(\Gamma, \Delta)$ does not sustain the above faults, we show how to add redundancy in order to improve its fault tolerance.

5.1 Preliminaries

We assume that a faulty component can be isolated, located, and disconnected. This implies that the effect of the fault can be isolated. Fault tolerance of a multiple bus system depends on the connectivity of the faulty *MBS* with respect to the non faulty *MBS*. Suppose that the original source algorithm requires direct data transfer from processor P_1 to processor P_2. Then the *MBS* realizing the algorithm has a direct path from P_1 to P_2. Suppose after a component failure, the *MBS* does not contain a direct path from P_1 to P_2 (assuming that P_1 and P_2 are not faulty) but there is an indirect path from P_1 to P_2. In that case data transfer from P_1 to P_2 may take several steps as opposed to the single step taken by the healthy *MBS*. This is an instance of sustaining the fault with performance degradation. But if the faulty *MBS* does not contain a path from P_1 to P_2, the communications required by the source algorithm cannot be performed on the faulty machine, even if performance degradation is acceptable. This is an instance where the *MBS* does not sustain a fault. As we see throughout the chapter, fault tolerance of $M(\Gamma, \Delta)$ depends on the connectivity property of the Cayley color graph $C_\Delta(\Gamma)$.

Definition 5.1: An *MBS* is *strongly connected* if there is a path from each processor to every other processor.

Lemma 5.1: Multiple bus system $M(\Gamma, \Delta)$ is strongly connected.

Proof: According to the construction of $M(\Gamma, \Delta)$, for every edge (u, v) in $C_\Delta(\Gamma)$ there is a direct path from processor u to processor v in $M(\Gamma, \Delta)$. According to Theorem 4.1, $C_\Delta(\Gamma)$ is strongly connected. Therefore, there is a path from every processor to every other processor in the *MBS* $M(\Gamma, \Delta)$. □

The fault tolerance of $M(\Gamma, \Delta)$ in front of an interface failure or a bus failure is based on the above lemma. We check the strong connectivity of the *MBS* after a component failure. If the *MBS* with a failed component is also strongly connected, then communications among processors are still possible but with some performance degradation. As we see later, fault tolerance of $M(\Gamma, \Delta)$ in case of a processor failure also depends on the strong connectivity of the faulty *MBS*. In the following sections we will discuss each component failure separately. Before studying fault tolerance of an $M(\Gamma, \Delta)$ we will present an important result concerning the connectivity of a Cayley color graph. To that end, a few concepts from group theory are necessary [131], [113]. For completeness, we state them next.

Definition 5.2: Let A be a subset of the elements of group Γ. Let γ be any element in Γ. Then γA is defined as the set $\{ \gamma a : a \in A \}$ [113].

Definition 5.3: Let Λ be a subgroup of Γ. Then $\gamma\Lambda$ is said to be the *left coset* of Γ generated by γ (or, containing γ) relative to Λ [113].

Similarly, we can define right coset $\Lambda\gamma$. However, in this dissertation, we only use left cosets. Therefore, unless otherwise stated, the word "coset" must be interpreted as a left coset.

Lemma 5.2: Any two cosets of a group are either identical or else have no element in common.

Proof: Let Λ be a subgroup of group Γ. Let $\gamma_1\Lambda$ and $\gamma_2\Lambda$ be two cosets of Γ. Suppose, there exists an element, say γ, common to both $\gamma_1\Lambda$ and $\gamma_2\Lambda$. Then there is an element $\lambda_1 \in \Lambda$ such that $\gamma = \gamma_1\lambda_1$. Also, there is an element $\lambda_2 \in \Lambda$ such that $\gamma = \gamma_2\lambda_2$. Hence, $\gamma_1\lambda_1 = \gamma_2\lambda_2$, that is, $\gamma_1 = \gamma_2\lambda_2\lambda_1^{-1}$. Therefore, $\gamma_1\Lambda = \gamma_2\lambda_2\lambda_1^{-1}\Lambda = \gamma_2\Lambda$. This proves the lemma. □

Corollary: A family of cosets of a group is a partition on the elements of the group.

Consequently, a family of cosets of a group Γ induce a partition on the vertex set of the Cayley color graph associated with Γ and its generating set. In general, partitions corresponding to right and left cosets are not the same.

Definition 5.4: For a subgroup Λ of Γ, if $\gamma\Lambda = \gamma\Lambda$ for every $\gamma \in \Gamma$, then Λ is a *normal subgroup* of Γ.

Definition 5.5: Let Γ be a finite group and Δ be a (non redundant) generating set for Γ. Let Λ be a subgroup of Γ. Then the subgraph of $C_\Delta(\Gamma)$ induced by a left coset of Γ relative to Λ will be called a *coset subgraph* of $C_\Delta(\Gamma)$ *relative to* Λ.

When there is no ambiguity as to the subgroup Λ under consideration, we will refer to a coset subgraph without referring to Λ. In such a case, the family of coset subgraphs of a Cayley color graph will be denoted by CS_1, CS_2, ..., etc. It is clear that every coset subgraph CS_i is a Cayley color graph by itself. Theorem 5.1 (to be introduced) provides an important result regarding the connectivity of $C_\Delta(\Gamma)$. To simplify its proof, we give the following algebraic result.

Lemma 5.3: Let x and k be integers such that $k > x > 0$. Then there exists an integer m such that $mx \bmod k \leq k - x + 1$.

Proof: Let $m = \lceil (k-1)/k - x) \rceil - 1$. Then, we have the following two inequalities.

$$(m+1)(k-x) \geq k-1 \tag{1}$$

$$m(k-x) < k-1 \tag{2}$$

From (2), we have, $mk - mx < k - 1$. Therefore,

$$mx - (m-1)k > 0 \tag{3}$$

Also, from (1), we have $mk - mx - x \geq -1$. Therefore, by combining (3), we have,

$$0 < mx - (m-1)k \leq k - x + 1 \qquad (4)$$

Inequality (3) implies that $mx \mod k = (mx - (m-1)k) \mod k$. Inequality (4) implies that $(mx - (m-1)k) \mod k \leq k - x + 1$. Therefore the required result follows. □

Theorem 5.1: The graph obtained from $C_\Delta(\Gamma)$ by deleting a single edge is strongly connected unless $|\Delta| = 1$.

Proof: If $|\Delta| = 1$, then $C_\Delta(\Gamma)$ is a (directed) cycle with $|\Gamma|$ vertices. Deletion of any edge breaks the cycle and the remaining graph is not therefore strongly connected. Now suppose $|\Delta| \geq 2$. Assume the deletion of edge (e, δ_1), where $\delta_1 \in \Delta$. We need only to prove that there exists a directed path from e to δ_1 in $C_\Delta(\Gamma)$ which does not use edge (e, δ_1). Let Λ be the subgroup of Γ generated by $\{\delta_1\}$. Let CS_1 be the coset subgraph of $C_\Delta(\Gamma)$ (relative to Λ and $\{\delta_1\}$) which contains vertex e. Since Γ is a finite group, δ_1 is of finite order, say, k. Therefore, CS_1 is a cycle of size k consisting of only color 1 edges. It is also clear that CS_1 contains edge (e, δ_1). First we prove the following claim.

> *Claim*: There exists a vertex γ in CS_1 which can be reached from e without using any edge from CS_1.
>
> *Proof*: Since $|\Delta| \geq 2$, there exists another generator, say $\delta_2 \in \Delta$. Let $a = \delta_1^{-1}$ and $b = \delta_2$. Then there is an edge from a to e of color 1 and an edge from e to b of color 2. See Figure 5.1. If we start from e and traverse along edges of colors, 2, 1, 2, 1, ..., in that order, we would come back to e. This is true because there is a finite integer g such that $(\delta_2\delta_1)^g = e$. Let γ be the first vertex in CS_1 encountered by the above traversal of edges. Then $\gamma \neq e$ since edge (a, e) must be included in the cycle. Thus there is a path from e to γ that does not use any edge from CS_1. Hence the claim.

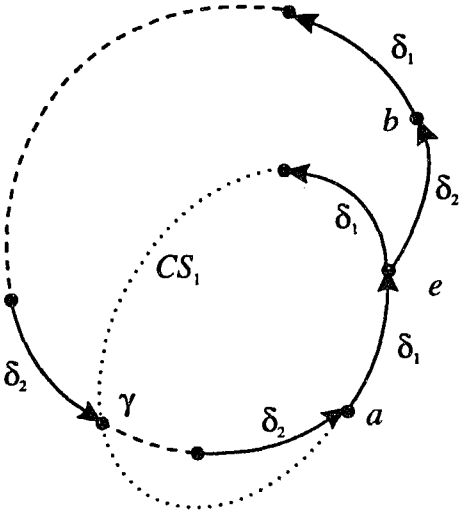


Figure 5.1: A path from e to γ which does not use any edge from CS_1.

Since γ is in CS_1, we have $\gamma = \delta_1^x$ for some integer $x < k$. Therefore, there is a *jump* of length x along CS_1 from e to γ. First suppose that there exists in integer l such that $lx = k + 1$. Therefore, we can move from e to γ by using l jumps. Thus, in that case, there exists a path from e to δ_1 that does not include edge (e, δ_1). Now suppose that such l does not exist. Then, let m be the least integer such that $mx \bmod k \leq k - x + 1$. By lemma 5.3, such an integer exists. Let $v = \delta_1^{mx}$ and $w = \delta_1^{k-x+1}$. Then v can be reached from e by m jumps. See Figure 5.2. Since our choice of m guarantees that $mx \bmod k \leq k - x + 1$, w can be reached from v along CS_1 without using edge (e, δ_1). Furthermore, δ_1 can be reached from w by a single jump. Therefore, there is a directed path from e to δ_1 which does not use edge (e, δ_1). □

5.2 Failure of a Bus

In this section we analyze the behavior of $M(\Gamma, \Delta)$ when a single bus fails. Failure of a bus will remove direct paths for some communicating pairs of processors. According

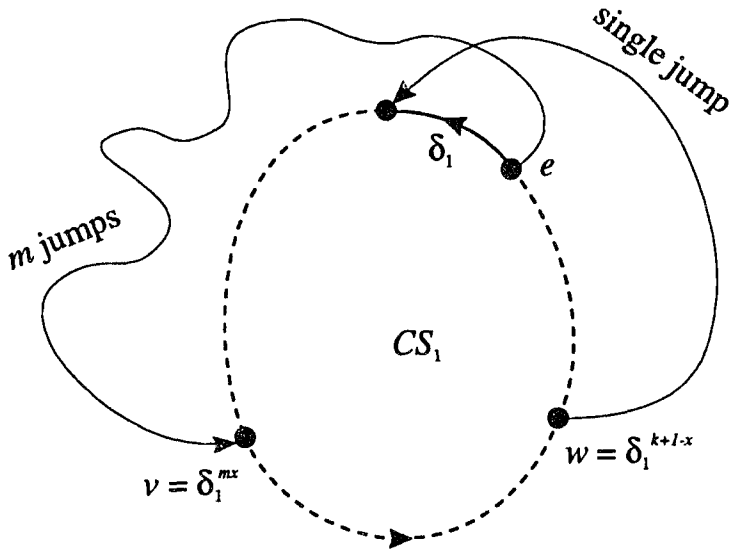


Figure 5.2: A path from e to δ_1 that does not include edge (e, δ_1).

to the way $M(\Gamma, \Delta)$ was constructed (see Section 4.2), bus B_j is assigned the data transfers corresponding to the set of edges E_j, $1 \le j \le |\Gamma|$, where

$E_j = \{(\gamma_j, \gamma_j h_1), (\gamma_j h_1, \gamma_j h_1 h_2), ..., (\gamma_j h_1 h_2 ... h_{c-1}, \gamma_j h_1 h_2 ... h_c)\}$; $h_i = (\delta_i)^{(-1)^{i-1}}$, $1 \le i \le |\Delta|$.

Figure 5.3 shows the set of processors connected to bus B_j, where $|\Delta|$ is assumed 5. Failure of bus B_j affects the direct communication among processors γ_j, $\gamma_j\delta_1$, $\gamma_j\delta_1\delta_2^{-1}$, ..., $\delta_1\delta_2^{-1}\delta_3\delta_4^{-1}\delta_5$. As for the Cayley color graph, failure of B_j corresponds to the removal of edges belonging to E_j from $C_\Delta(\Gamma)$. The following theorem provides yet another attractive feature of the *MBS* $M(\Gamma, \Delta)$.

Theorem 5.2: The *MBS* $M(\Gamma, \Delta)$ can sustain a single bus failure if and only if $|\Delta| > 2$.

Proof: First suppose $|\Delta| \le 2$. In this situation each processor P is connected with only one receiver (see Theorem 4.9). Failure of the bus connected to that receiver will destroy all input paths to processor P.

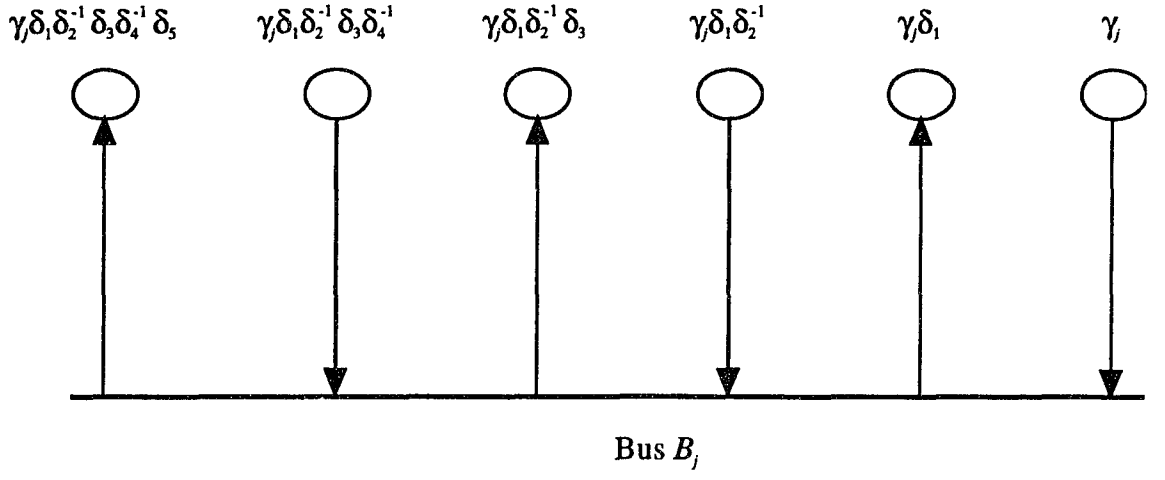


Figure 5.3: Processors connected to bus H_j in $M(\Gamma, \Delta)$.

Now suppose that $|\Delta| > 2$. Assume that bus B_j in $M(\Gamma, \Delta)$ fails. Figure 5.4 shows the induced subgraph H_j, where v_i denotes the i^{th} vertex of H_j. The set of processors among which communication may be in jeopardy due to the failure of bus B_j is represented by $V(H_j)$. To prove the theorem, it is sufficient to show the existence of a path in $C_\Delta(\Gamma)$ from v_i to $u_{i'}$ which does not use any of the edges in H_j. Here i is an even integer in the range $0 \le i \le |\Delta|$ and i' is an integer one less than or one more than i in the range $0 < i' \le |\Delta|$.

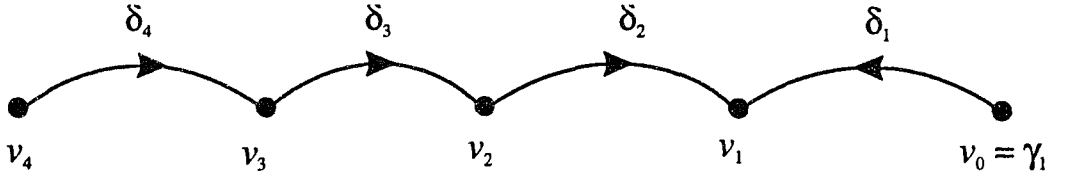


Figure 5.4: Induced subgraph H_j

We will use Λ_{rs} to denote the subgroup of Γ generated by $\{\delta_r, \delta_s\}$ for $\delta_r, \delta_s \in \Delta$. Denote by CS_{13} the coset subgraph of $C_\Delta(\Gamma)$ corresponding to the left coset of Γ

associated with subgroup Γ_{13} containing vertex $v_0 = \gamma_j$. It is clear that CS_{13} is a Cayley color graph with two colors. We claim that there is a path from v_0 to v_1 in $C_\Delta(\Gamma)$ which does not use any of the edges in H_j. It is clear that edge (v_0, v_1) of color 1 is in the subgraph CS_{13} (see Figure 5.4). Therefore, according to Theorem 5.1, there exists a path in CS_{13} which does not use edge (v_0, v_1). Suppose that the color 3 edge (v_2, v_3) is in that path. Then both vertices v_1 and v_2 are in CS_{13}. Therefore, since $v_1 = v_2\delta_2$, generator δ_2 could be expressed in terms of generators δ_1 and δ_3. This is not possible since we assume that Δ is a non redundant generating set. Therefore, there is a path from v_0 to v_1 in CS_{13} which does not use (v_0, v_1) or (v_2, v_3). Thus the claim is true and therefore, there exists a path from v_0 to v_1 which does not use any of the edges in H_j.

Now, let CS_{12} be the subgraph of $C_\Delta(\Gamma)$ corresponding to the left coset of Γ associated with subgroup Λ_{12} containing v_2. Then, according to Theorem 5.1, there is a path, say $\mathscr{P}$, from v_2 to v_1 in CS_{12} which does not use the color 2 edge (v_2, v_1). First suppose that $\mathscr{P}$ does not contain edge (v_0, v_1). Then $\mathscr{P}$ is a path from v_2 to v_1 which does not contain any edge from H_j. Next suppose $\mathscr{P}$ contains edge (v_0, v_1). We have already shown that there exists a path from v_0 to v_1 that does not use any edge from H_j. Hence, there exists a path from v_2 to v_1 that does not use any edge from H_j. Similar to the case for the path from v_0 to v_1, we can show that there exists a path from v_2 to v_3 that does not use any edge from H_j. We can continue this argument to prove that for every even i, $0 \leq i \leq |\Delta|$, there exist a path from v_i to v_{i+1} and a path from v_i to v_{i-1} that do not use any edge from H_j. Therefore, $M(\Gamma, \Delta)$ remains strongly connected even after the failure of bus B_j. □

Therefore, $M(\Gamma, \Delta)$ can function with some performance degradation in the case of a single bus failure, provided that $|\Delta| > 2$. Performance degradation of a faulty *MBS* depends on how many buses are needed to send data from processor P_1 to processor P_2, when the direct path from P_1 to P_2 is destroyed. Since $M(\Gamma, \Delta)$ is symmetric, performance degradation is the same irrespective of the identity of the failed bus. Unfortunately, there is no general formula to find the distance from P_1 to P_2 when the bus associated with the direct path from P_1 to P_2 is destroyed. However, we will obtain a measure of performance degradation when $M(\Gamma, \Delta)$ is the optimal *MBS* realizing cube interconnection functions.

5.3 Performance Degradation of $M(Q_n)$ due to a Bus Failure

Theorem 5.3: When a single bus fails in $M(Q_n)$ ($n > 2$), a processor connected to the faulty bus via a driver can send data to a processor connected to the same bus via a receiver, using three non faulty buses (*i.e.*, via a path of length 3).

Proof: As we have already shown, vertices of the *IFG* $G_{Q(n)}$ can be labeled by binary strings of length n. Furthermore, buses can also be assigned the same labels. Suppose that bus γ_c fails. Consider two processors γ_a and γ_b connected to bus γ_c via a driver and a receiver, respectively.

There is a direct path from processor γ_a to processor γ_b if and only if $\gamma_b = \gamma_a \oplus s$, for some *SCOL* s (see Lemma 4.6). By the definition, exclusive-or summation of two *SCOL*s cannot be an *SCOL*. Therefore, there cannot exist a path of length two from γ_a to γ_b. Also, according to Lemma 4.4, there can be at most one direct path in $M(\Gamma, \Delta)$ from one processor to another. Therefore, if bus γ_c fails, at least three buses are required to send data from γ_a to γ_b. We will next show that this is actually possible.

Since processor γ_a is connected to bus γ_c via a driver, $\gamma_a = \gamma_c \oplus 0^{n-\alpha}1^{\alpha}$, for an even integer α. Also, since processor γ_b is connected to bus γ_c via a receiver, $\gamma_b = \gamma_c \oplus 0^{n-\beta}1^{\beta}$, for an odd integer β. Therefore, $s = 0^{n-\alpha}1^{\alpha} \oplus 0^{n-\beta}1^{\beta}$. Without loss of generality, assume that $\alpha > \beta$. Then $\alpha \geq 2$. Consider bus γ_c^1 given by

$$\gamma_c^1 = \gamma_a \tag{1}$$

It is clear that processor γ_a is connected to bus γ_c^1 via a driver. Let

$$\gamma_b^1 = \gamma_c^1 \oplus 0^{n-1}1.$$

Now, processor γ_b^1 is connected to bus γ_c^1 via a receiver. Thus, there is a direct path from processor γ_a to processor γ_b^1 via bus γ_c^1. Next consider bus γ_c^2 given by

$$\gamma_c^2 = \gamma_b^1 \tag{2}$$

Thus, processor γ_b^1 is connected to bus γ_c^2 via a driver. Let

$$\gamma_b^2 = \gamma_c^2 \oplus 0^{n-\beta}1^{\beta}.$$

Since β is odd, processor γ_b^2 is connected to bus γ_c^2 via a receiver. Thus, there is a direct path from processor γ_b^1 to processor γ_b^2 via bus γ_c^2. Finally, consider bus γ_c^3 given by,

$$\gamma_c^3 = \gamma_b^2 \oplus 0^{n-\alpha}1^{\alpha}. \tag{3}$$

Since α is even, processor γ_b^2 is connected to bus γ_c^3 via a driver. Let

$$\gamma_b^3 = \gamma_c^3 \oplus 0^{n-1}1.$$

Then, processor γ_b^3 is connected to bus γ_c^3 via a receiver. Thus, there is a direct path from processor γ_b^2 to processor γ_b^3 via bus γ_c^3. Hence, there is a path of length 3 from processor γ_a to processor γ_b^3 via the three buses γ_c^1, γ_c^2, and γ_c^3. Now,

$$\begin{aligned}\gamma_b^3 &= \gamma_c^3 \oplus 0^{n-1}1 \\ &= \gamma_b^2 \oplus 0^{n-\alpha}1^{\alpha} \oplus 0^{n-1}1 \\ &= \gamma_c^2 \oplus 0^{n-\beta}1^{\beta} \oplus 0^{n-\alpha}1^{\alpha} \oplus 0^{n-1}1\end{aligned}$$

$$= \gamma_b^1 \oplus 0^{n-\beta}1^{\beta} \oplus 0^{n-\alpha}1^{\alpha} \oplus 0^{n-1}1$$

$$= \gamma_c^1 \oplus 0^{n-1}1 \oplus 0^{n-\beta}1^{\beta} \oplus 0^{n-\alpha}1^{\alpha} \oplus 0^{n-1}1$$

$$= \gamma_c^1 \oplus 0^{n-\beta}1^{\beta} \oplus 0^{n-\alpha}1^{\alpha}$$

$$= \gamma_a \oplus 0^{n-\beta}1^{\beta} \oplus 0^{n-\alpha}1^{\alpha}$$

$$= \gamma_b.$$

Therefore, there is a path of length 3 from γ_a to γ_b which uses the three buses γ_c^1, γ_c^2, and γ_c^3. It remains to show that none of those buses is γ_c (the faulty one). Since $\gamma_c = \gamma_a \oplus 0^{n-\alpha}1^{\alpha}$ (we assume $\alpha > 0$), according to (1) $\gamma_c^1 \neq \gamma_c$. Also, from (2), $\gamma_c^2 = \gamma_b^1 = \gamma_c^1 \oplus 0^{n-1}1 = \gamma_a \oplus 0^{n-1}1$. Clearly $\gamma_c^2 \neq \gamma_c$ since $\alpha \neq 1$. Furthermore, from (3),

$$\gamma_c^3 = \gamma_b^2 \oplus 0^{n-\alpha}1^{\alpha}$$

$$= \gamma_c^2 \oplus 0^{n-\beta}1^{\beta} \oplus 0^{n-\alpha}1^{\alpha}$$

$$= \gamma_a \oplus 0^{n-1}1 \oplus 0^{n-\beta}1^{\beta} \oplus 0^{n-\alpha}1^{\alpha}$$

Now, $0^{n-1}1 \oplus 0^{n-\beta}1^{\beta}$ cannot be 0^n since $\beta \neq 1$. Therefore, $\gamma_c^3 \neq \gamma_c$. So, there is a path of length 3 from γ_a to γ_b which uses three non faulty buses. The same result can be reached if we assume $\alpha < \beta$. □

To preserve possible precedence relations in the source algorithm, we should finish data transfers corresponding to a particular interconnection function before starting data transfers corresponding to another interconnection function. To perform the communication corresponding to an interconnection function, four steps are necessary for the faulty *MBS* as opposed to the single step necessary for the non faulty *MBS*. In the case of a bus failure, one step is required for source processors not using the faulty bus to communicate. Three more steps are necessary for the processor which could not communicate

because of the faulty bus to reroute the data to its target processor. Therefore, the communication time will increase by a factor of at most four due to a single bus failure.

5.4 Failure of an Interface

In this section we will analyze the fault tolerance of $M(\Gamma, \Delta)$ in the case of a single interface failure. Clearly, bus failure is more serious than an interface failure, in general. Failure of a bus is equivalent to the failure of all the interfaces connected to it, according to our fault model. Therefore, we would expect that $M(\Gamma, \Delta)$ is more tolerant to an interface failure than to a bus failure. But, as we shall see shortly, the above statement is only partially true. Due to the specific way $M(\Gamma, \Delta)$ was constructed, fault tolerance of $M(\Gamma, \Delta)$ with respect to a driver failure and a receiver failure are not identical, in general.

Theorem 5.4: The *MBS* $M(\Gamma, \Delta)$ sustains any single driver failure if and only if $|\Delta| > 1$. Furthermore, it sustains any single receiver failure if and only if $|\Delta| > 2$.

Proof: According to Theorem 4.9, the number of drivers and receivers connected to a processor in $M(\Gamma, \Delta)$ are $\lfloor |\Delta|/2 \rfloor + 1$ and $\lceil |\Delta|/2 \rceil$, respectively. When $|\Delta| = 1$, that is, when there is only one generator in the Cayley color graph, each processor has only one driver and one receiver. Thus $M(\Gamma, \Delta)$ does not sustain a driver failure or a receiver failure when $|\Delta| = 1$. When $|\Delta| = 2$, that is, when there are two generators, each processor is connected with 2 drivers and 1 receiver. Therefore, $M(\Gamma, \Delta)$ does not sustain a receiver failure when $|\Delta| = 2$. We will next show that $M(\Gamma, \Delta)$ sustains a driver failure when $|\Delta| = 2$.

When $|\Delta| = 2$, let δ_1 and δ_2 be the two generators in Δ. Then, bus γ_j is connected to processors γ_j and $\gamma_j\delta_1\delta_2^{-1}$ through drivers and to processor $\gamma_j\delta_1$ through a receiver (see

Figure 5.3). Suppose that the driver connecting processor γ_j to bus γ_j fails. The only direct path which uses the faulty driver is the one associated with edge $(\gamma_j, \gamma_j\delta_1)$. According to Theorem 5.1, there is a path from γ_j to $\gamma_j\delta_1$ in $C_\Delta(\Gamma)$ which does not use edge $(\gamma_j, \gamma_j\delta_1)$. Therefore, there is a path from processor γ_j to processor $\gamma_j\delta_1$ which does not require the faulty driver. Similar reasoning applies if the driver connecting bus γ_j to processor $\gamma_j\delta_1\delta_2^{-1}$ fails. It remains to show that, when $|\Delta| > 2$, $M(\Gamma, \Delta)$ sustains a driver or a receiver fault. This result directly follows from Theorem 5.2. □

Therefore, even though a bus failure appears to be more serious than an interface failure, the degree of fault tolerance of $M(\Gamma, \Delta)$ with respect to both is the same with only one minor exception. Even though $M(\Gamma, \Delta)$ cannot sustain a bus failure when $|\Delta| = 2$, it can sustain a driver failure. Similar to the faulty bus case, there is no general formula to find the distance from P_1 to P_2 when the driver connected to P_1 or the receiver connected to P_2 associated with the direct path from P_1 to P_2 is faulty. Therefore, a measure of performance degradation under an interface failure cannot be obtained for the general case. However, we can obtain a measure of performance degradation when the *MBS* is $M_{Q(n)}$.

5.5 Performance Degradation of $M(Q_n)$ due to an Interface Failure

Theorem 5.5: If $M(Q_n)$ sustains the failure of a driver or receiver, then any pair of processors in the faulty *MBS* can communicate using at most three buses.

Proof: First suppose that the *MBS* $M_{Q(n)}$ be such that $n > 2$. Let γ_a and γ_b be two processors such that there is a direct path from γ_a to γ_b via bus γ_c. In Theorem 5.3, we have already shown the existence of a path of length 3 from processor γ_a to processor γ_b

which does not use bus γ_c. The same three buses can be used if the driver connecting γ_a to γ_c or the receiver connecting γ_b to γ_c fails.

Now suppose that $n = 2$. In this case, $M_{Q(n)}$ does not tolerate a receiver failure (see Theorem 5.4). Suppose that the driver connecting γ_a to γ_c fails. As in the proof of Theorem 5.3, let $\gamma_a = \gamma_c \oplus 0^{n-\alpha}1^{\alpha}$, for an even integer α, and $\gamma_b = \gamma_c \oplus 0^{n-\beta}1^{\beta}$, for an odd integer β. Since $n = 2$, β must be equal to 1. Also, α must be either 0 or 2. We will assume $\alpha = 2$. The same result can be reached when $\alpha = 0$.

Since $\alpha = 2$, we have $\gamma_c = \gamma_a \oplus 0^{n-2}11$. Let the two buses γ_c^1 and γ_c^2 be defined by $\gamma_c^1 = \gamma_a$ and $\gamma_c^2 = \gamma_a \oplus 0^{n-2}10$. Then processor γ_a and processor $\gamma_a \oplus 0^{n-1}1$ are connected to bus γ_c^1 via a driver and a receiver, respectively. Therefore, there is a direct path from processor γ_a to processor $\gamma_a \oplus 0^{n-1}1$ via bus γ_c^1. Also, there is a direct path from processor $\gamma_a \oplus 0^{n-1}1$ to processor $\gamma_a \oplus 0^{n-2}11$ via bus γ_c^2. Furthermore, there is a direct path from processor $\gamma_a \oplus 0^{n-2}11$ to processor $\gamma_a \oplus 0^{n-1}1 = \gamma_b$ via bus γ_c. Hence there is a path of length 3 from γ_a to γ_b which does not use the driver connecting γ_a to γ_c. □

Notice that when $n = 2$, the indirect path from processor γ_a to processor γ_b uses bus γ_c. But it doesn't use the faulty receiver. While executing any interconnection function, the faulty $M_{Q(n)}$ requires three extra communication steps. Therefore, similar to the bus failure case, whenever $M_{Q(n)}$ sustains an interface failure, the communication time will be increased by a factor of at most four.

5.6 Failure of a Processor

In this section we will analyze the behavior of $M(\Gamma, \Delta)$ when a processor fails. Failure of a processor does not affect the direct paths among other processors. But that may affect indirect paths. Also, when a processor fails, the subtask assigned to the faulty

processor must be reassigned to one or more non faulty processors. We assume that the multiple bus system detects the faulty processor and assigns its subtask to one or more of its neighboring processors. We will show that unless $|\Delta| = 1$, $M(\Gamma, \Delta)$ is strongly connected after a processor failure. Unlike the previous two cases (interface failure and bus failure), more involved proofs are required to prove that $M(\Gamma, \Delta)$ remains strongly connected after the failure of a single bus.

Lemma 5.4: Let G be a Cayley color graph associated with group Γ and its generating set Δ. Let Λ be the subgroup of Γ generated by $\Delta_1 \subset \Delta$. All edges of color r, $\delta_r \in \Delta - \Delta_1$, originating from one coset subgraph relative to Λ will terminate at another coset subgraph (relative to Λ) if and only if Λ is a normal subgroup of Γ.

Proof: First suppose that Λ is a normal subgroup of Γ. Then $\gamma\Lambda\delta_r = \gamma\delta_r\Lambda$. Thus all edges of color r originating from coset subgraph $\gamma\Lambda$ terminate at coset subgraph $\gamma\delta_r\Lambda$.

Now suppose that, for every coset $\gamma_1\Lambda$, there exists another coset $\gamma_2\Lambda$ such that $\gamma_2\Lambda = \gamma_1\Lambda\delta_r$. Clearly, $\gamma_1\delta_r \in \gamma_1\Lambda\delta_r$. So, $\gamma_1\delta_r$ belongs to the left coset $\gamma_2\Lambda$. Furthermore, $\gamma_1\delta_r$ belongs to the left coset $\gamma_1\delta_r\Lambda$. Therefore, by Lemma 5.2, $\gamma_1\Lambda\delta_r = \gamma_1\delta_r\Lambda$. So, $\Lambda\delta_r = \delta_r\Lambda$. This is true for every $\delta_r \in \Delta - \Delta_1$. Furthermore, for every $\delta_s \in \Delta_1$, we have $\Lambda\delta_s = \delta_s\Lambda$. Let γ be an arbitrary element in Γ. Since Δ is a generating set for Γ, γ is a word in Δ. Hence, $\gamma\Lambda = \Lambda\gamma$, and therefore, Λ is a normal subgroup of Γ. □

Lemma 5.5: Let $G = (V, E)$ be the Cayley color graph associated with finite group Γ and its generating set Δ. Let $A \subseteq V$ be a subset of vertices. Then the number of edges from A to $V - A$ of color r is equal to the number of edges from $V - A$ to A of color r, for every color r in G.

Proof: Each vertex in A has one outgoing edge of color r. Therefore, there are $|A|$ edges of color r whose tails are in A. From the heads of those edges, let q be in $V - A$. The remaining $|A| - q$ heads are in A itself. Furthermore, there are $|A|$ edges of color r whose heads are in A. From those heads, $|A| - q$ are heads which originate in A itself. Therefore, there are q edges from $V - A$ to A. □

The next lemma, which requires a lengthy proof, provides an important characterization of the connectivity among coset subgraphs of a Cayley color graph.

Lemma 5.6: Let $G = (V, E)$ be the Cayley color graph associated with finite group Γ and its generating set $\Delta = \{\delta_1, \delta_2, ..., \delta_k\}$. Let Λ be the subgroup of Γ generated by $\Delta_1 \subset \Delta$ and S be the family of coset subgraphs relative to Λ and Δ_1. Then S can be divided into two disjoint subsets A and B satisfying

(a) $|A| \geq 1$,

(b) $|B| \geq 2$, and

(c) All edges in $C_\Delta(\Gamma)$ from A to B terminate at a single coset subgraph in B.

only if Λ is a normal subgroup of Γ.

Proof: Suppose that the statements (a), (b), and (c) of the lemma are true. Also suppose that all edges from A to B terminate at coset subgraph CS_1. First, assume that A contains only one coset subgraph, say CS_a. Then, all edges originating from CS_a terminate in CS_1. So, by Lemma 5.4, Λ is a normal subgroup of Γ.

Next, assume that A contains more than one coset subgraph. In that case, we claim that there exists a subset $A' \subset S$, $|A'| < |A|$, such that all edges from A' to $S - A' = B'$ terminate at a single coset subgraph in B'. If the claim is true, we can ultimately find a coset subgraph CS_a such that all edges originating from CS_a terminate at a single coset

subgraph. This will then imply that Λ is a normal subgroup of Γ. We now prove the correctness of the above claim.

We first observe that, by symmetry of $C_{\Delta}(\Gamma)$ and by Statement (c) of the lemma, for every coset subgraph CS_i, there exists a subset $\xi(CS_i) \subset S$ such that all edges from $\xi(CS_i)$ to $S - \xi(CS_i)$ terminate at CS_i. Clearly, $\xi(CS_1) = A$. It is also clear that CS_i is not contained in $\xi(CS_i)$, for $CS_i \in S$. If CS_i and CS_j are distinct coset subgraphs, then $\xi(CS_i) \neq \xi(CS_j)$, for otherwise, edges originating from $\xi(CS_i)$ should terminate at CS_i as well as at CS_j.

To obtain the subset $A' \subset S$ stated in the claim, we first choose a coset subgraph CS_x in A such that $\xi(CS_x) \cap A$ is not empty. Suppose that $\xi(CS_x) \cap A$ is empty for selected CS_x. Then, $\xi(C_x) \subseteq B$. Let there be q edges from A to B. Then, there are q edges from A to CS_1. Therefore, by symmetry, there are q edges from $\xi(CS_x)$ to CS_x. From Lemma 5.5, the number of edges from B to A is q. Since $\xi(CS_x) \subseteq B$, all edges from B to A terminate at CS_x. Let $CS_{x'}$ be another coset subgraph other than CS_x in A (since A is assumed to contain more than one coset subgraph, such a coset subgraph exists). Then there are no edges from B to $CS_{x'}$. Therefore, $\xi(CS_{x'})$ cannot be completely contained in B. Hence $\xi(CS_{x'}) \cap A$ is not empty. Thus, without loss of generality, we will assume that $\xi(CS_x) \cap A$ is not empty. It should be noticed that $\xi(CS_x) \cap B$ cannot be empty since $\xi(CS_x) \neq A$. Denote $\xi(CS_x) \cap A$ and $\xi(CS_x) \cap B$ by A_x and B_x, respectively (see Figure 5.5(a)). The construction of A' will be different depending on whether CS_1 is contained in B_x or not.

Case 1: CS_1 is not contained in B_x.

Since all edges from A to B terminate at CS_1, there are no edges from A_x to B_x. Also, all edges originating from $A_x \cup B_x$ ($= \xi(CS_x)$) terminate at CS_x. Hence all edges originating from A_x should terminate at CS_x. Clearly, $|A_x| < |A|$. By letting $A' = A_x$, we have the proof for claim 1.

Case 2: CS_1 is contained in B_x.

In this case, there are no edges from A to $B - B_x$, because, by Statement (c) of the lemma, all edges from A to B terminate at CS_1. Also, there are no edges from B_x to $B - B_x$ because, edges originating from B_x terminate either in CS_x or in A_x. Therefore, there are no edges terminating at $B - B_x$. Since this violates the strong connectivity of $C_\Delta(\Gamma)$, $B - B_x$ must be empty. Therefore, $B_x = B$. Let CS_y be any coset subgraph in $A - A_x - CS_x$. By the hypothesis of the lemma (Statement (b) stipulates that $S - \xi(CS_i)$ contains at least two coset subgraphs, for $CS_i \in S$), such a coset subgraph exists. Since $|\xi(CS_y)| = |\xi(CS_x)| = |A_x| + |B| > |B|$, $\xi(CS_y) \cap A = A_y$ cannot be empty. Let $B_y = \xi(CS_y) \cap B$. If CS_1 is not contained in B_y, then the claim holds by Case 1. If CS_1 is contained in B_y, then $B - B_y$ must be empty. Therefore, $B_y = B$. Let $A_{xy} = A_x \cap A_y$ and $R = A_{xy} \cup B$. We will now prove that CS_x is contained in A_y. Suppose on the contrary that CS_x is not contained in A_y (see Figure 5.5(b)). Since $R \subset \xi(CS_x)$, the edges originating from R would not terminate in $(A - CS_x - A_x)$. Similarly, since $R \subset \xi(CS_y)$, the edges originating from R would not terminate in $(A - CS_y - A_y)$. But $(A - CS_x - A_x) \cup (A - CS_y - A_y) = A - A_{xy} = S - R$. So, there are no edges from R to $S - R$. But this violates the strong connectivity of $C_\Delta(\Gamma)$, and so CS_x must be contained in A_y. In that case (see Figure 5.5(c)), all edges originating from R will terminate at CS_x. Since $A_x \neq A_y$, we have $|A_{xy}| < |A_x|$.

Thus $|R| = |A_{xy} \cup B| = |A_{xy}| + |B| < |A_x| + |B| = |\xi(CS_x)| = |A|$. We obtain A' by letting $A' = R$. The proof of the claim for Case 2 is thereby complete. □

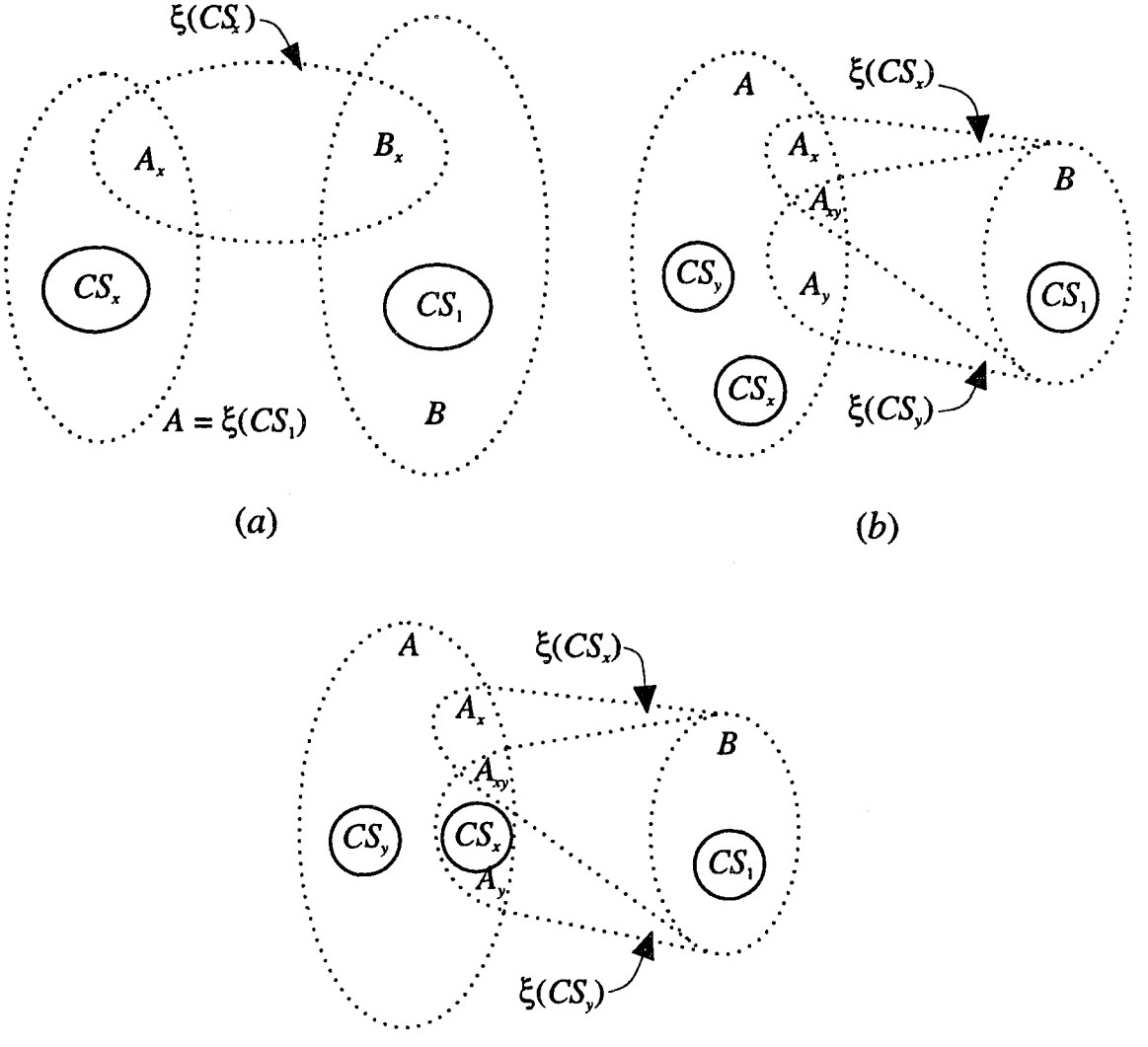


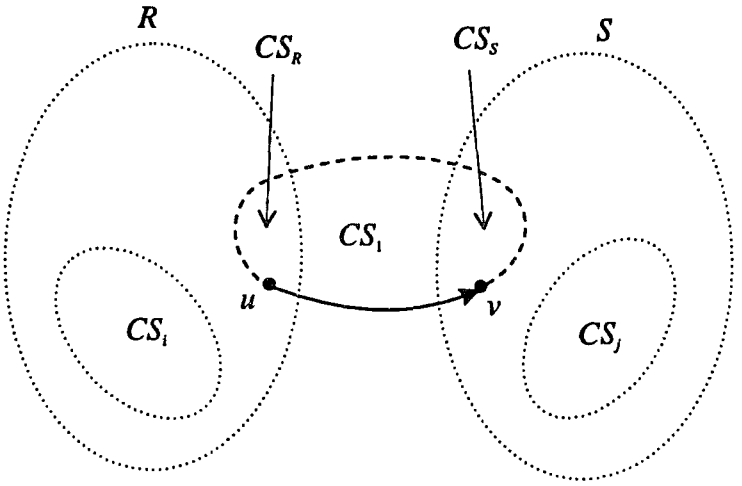
Figure 5.5: Illustration for Lemma 5.6.

Lemma 5.7: Let $C_\Delta(\Gamma)$ be such that $|\Delta| > 1$. Then $V(C_\Delta(\Gamma))$ cannot be divided into two disjoint subsets R and S, $|R| \geq 1$, $|S| \geq 2$, such that all edges from R to S terminate at a single vertex in S.

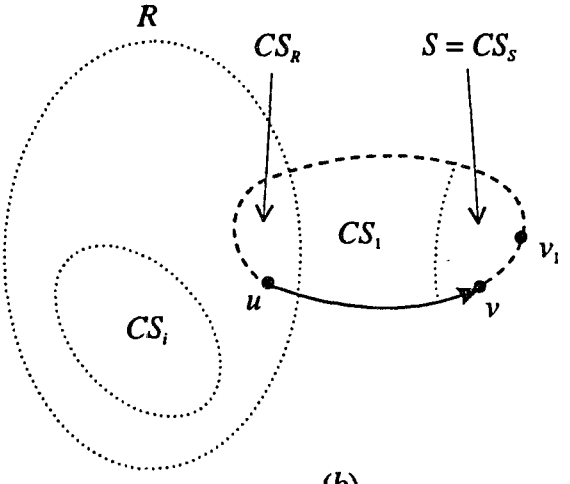
Proof: Suppose that there exist two subsets of vertices R and S such that all edges from R to S terminate at $v \in S$. Due to the strong connectivity of $C_\Delta(\Gamma)$, there should be at least one edge from R to S. Let (u, v) be such an edge whose color is, say, 1. Let Λ_1 be the subgroup of Γ generated by $\{\delta_1\}$. Denote by CS_i the family of coset subgraphs relative to Λ_1 and $\{\delta_1\}$. It is clear that CS_i is a directed cycle consisting of color 1 edges. Let the coset subgraph containing edge (u, v) be CS_1. Clearly, CS_1 is common to subsets R and S (see Figure 5.6(a)). Suppose that there is a coset subgraph CS', other than CS_1, which is also common to both R and S. Then, there must be another edge of color 1 from R to S. Since this is not possible by the hypothesis of the lemma, such a coset subgraph CS' does not exist. Therefore, CS_1 is the only coset subgraph common to A and B. Let $CS_1 \cap R$ and $CS_1 \cap S$ be denoted by CS_R and CS_S, respectively (see Figure 5.6(a)).

Now, we claim that $S - CS_s$ is empty. Suppose that it is not empty. Since all edges from R to S terminate at v, it is true that all edges from $R - CS_1$ to $S \cup CS_1$ terminate at the single coset subgraph CS_1. Therefore, according to Lemma 5.6, Λ must be a normal subgroup of Γ. There are no edges from CS_R to $S - CS_S$ by the hypothesis of the lemma. Therefore, since Λ is a normal subgroup, there are no edges from CS_R to $S - CS_R$. Also, by the hypothesis of the lemma, there are no edges from $R - CS_R$ to $S - SC_S$. This implies that CS_S has no incoming edges. Therefore, due to the strong connectivity of $C_\Delta(\Gamma)$, $S - CS_S$ must be empty. Therefore, $S = CS_S$. Figure 5.6(b) shows the new configuration. Let v_1 be a vertex in S other than v. Then incoming edges to v_1

of colors other than 1 should originate from R. Since this not possible by the hypothesis of the lemma, the only vertex in S must be v. This also violates the condition given in the lemma ($|S| \geq 2$). This completes the proof of the lemma. □



(*a*)



(*b*)

Figure 5.6: Illustration for Lemma 5.7.

Lemma 5.8: Let $C_\Delta(\Gamma)$ be such that $|\Delta| > 1$. Then $C_\Delta(\Gamma)$ remains strongly connected after the removal of a single vertex.

Proof: Let v be an arbitrary vertex of $C_\Delta(\Gamma)$. If $C_\Delta(\Gamma)$ is not strongly connected, then there exists two vertices v_1 and v_2 such that all paths from v_1 to v_2 pass through v. Let R be the set of vertices of $C_\Delta(\Gamma)$ (including v_1 but excluding v) which can be reached from v_1 without passing through v. Let $S = V(C_\Delta(\Gamma)) - R$. Since v and v_2 are in S, we have $|S| \geq 2$. Furthermore, every edge from R to S terminates at v. This is not possible according to Lemma 5.7. Hence, there is a path from v_1 to v_2 which does not pass through v. □

As a consequence of Lemma 5.8, we have the following theorem.

Theorem 5.6: If $|\Delta| > 1$, then the *MBS* $M(\Gamma, \Delta)$ remains strongly connected after the failure of a single processor. □

Thus $M(\Gamma, \Delta)$ sustains a single processor failure provided that $|\Delta| > 1$. Similar to the previous two cases, there is no general formulation for the performance degradation of $M(\Gamma, \Delta)$ with respect to a processor fault. As before, we will obtain a measure of performance degradation in the special case when *MBS* is $M(Q_n)$.

5.7 Performance Degradation of $M(Q_n)$ due to a Processor Failure

As mentioned before, failure of a processor does not affect the communication among other neighboring processors. But extra data transfers are required when the subtask originally assigned to the faulty processor is reassigned to one or more non-faulty processors. Let P_f be the faulty processor. Let P_i be the processor whose corresponding vertex in the *IFG* $G_{Q(n)}$ is adjacent with the vertex corresponding to processor P_f along dimension i, $1 \leq i \leq n$. For the analysis in this section, we assume that the subtask of the faulty processor is assigned to a single neighboring processor, say P_1. See Figure 5.7.

Consider the situation when $M(Q_n)$ is performing the interconnection function associated with generator δ_2, that is, when the *MBS* is performing the $cube_2$ interconnection function [125]. The subtasks assigned to processors P_f and P_2 must communicate with each other. In the non-faulty $M(Q_n)$, processor P_f can communicate with processor P_2 in one step. To accomplish this in the single processor fault situation, processor P_1 should communicate (simultaneously, if possible) with processor P_2.

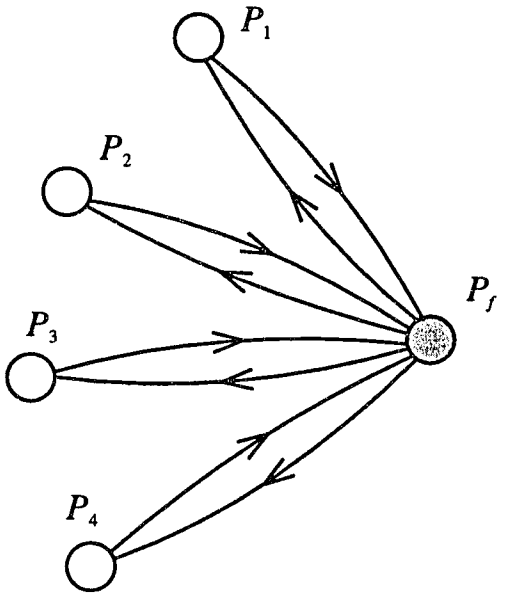


Figure 5.7: Communication among neighboring processors of a faulty processor in $M(Q_n)$.

Let γ_f be the vertex of the *IFG* $G_{Q(n)}$ corresponding to processor P_f in the *MBS* $M_{Q(n)}$. Then, $\gamma_f\delta_i$ is the vertex corresponding to processor P_i, $1 \leq i \leq n$. Now, there is a direct path from $\gamma_f\delta_1$ to $\gamma_f\delta_1\delta_2$ via bus $\gamma_f\delta_2$. Also, there is a direct path from $\gamma_f\delta_1\delta_2$ to $\gamma_f\delta_2$ via bus $\gamma_f\delta_1\delta_2$. Thus, there is a path from $\gamma_f\delta_1$ to $\gamma_f\delta_2$ via the two buses $\gamma_f\delta_2$ and $\gamma_f\delta_1\delta_2$. Similarly, there is a path from $\gamma_f\delta_2$ to $\gamma_f\delta_1$ via the two buses $\gamma_f\delta_1\delta_2$ and γ_f, in that order. Notice that both paths contain the bus $\gamma_f\delta_1\delta_2$. When the data transfer from P_1 to P_2 is

using bus $\gamma_f\delta_1\delta_2$, the data transfer from P_2 to P_1 will be using bus γ_f. Also, when the data transfer from P_2 to P_1 is using bus $\gamma_f\delta_1\delta_2$, the data transfer from P_1 to P_2 will be using bus $\gamma_f\delta_2$. Therefore, no bus conflict will occur. Hence, processors $\gamma_f\delta_1$ (= P_1) and $\gamma_f\delta_2$ (= P_2) can communicate with each other using only two steps. Thus, two extra communication steps are required in order to perform the communications required by the faulty processor corresponding to the interconnection function δ_2. This is true for every other interconnection function except δ_1. Therefore, communication time will be increased by at most three times because of a processor failure.

5.8 Inclusion of Redundancy

When the *MBS* does not sustain the failure of a certain component, we can make it fault tolerant by adding some redundant components. In all three cases of component failures we have addressed (bus, interface, and processor), whether $M(\Gamma, \Delta)$ is fault tolerant or not was decided by the size of $|\Delta|$. In $M(\Gamma, \Delta)$, only the number of interfaces is dependent on $|\Delta|$. Therefore, to make $M(\Gamma, \Delta)$ fault tolerant, whenever necessary, only interfaces need to be added.

If $\Delta = \{\delta_1\}$, that is, if $|\Delta| = 1$, then $M(\Gamma, \Delta)$ is not fault tolerant in the face of any component failure. Note that $\Delta = \{\delta_1\}$ corresponds to an *IFG* with unidirectional ring topology. In such a case, we can use generating set $\Delta' = \{\delta_1, \delta_1^{-1}\}$ of group Γ instead of Δ. Even though Δ' is a redundant generating set, color partition $\phi_{\Gamma,\Delta'}$ is optimal (see Section 4.2). From Theorems 5.4 and 5.6, $M(\Gamma, \Delta')$ is fault tolerant against a single processor failure or a single driver failure. According to Theorem 4.9, the number of interfaces in $M(\Gamma, \Delta)$ and $M(\Gamma, \Delta')$ are $2|\Gamma|$ and $3|\Gamma|$, respectively. Therefore, an *MBS* $M(\Gamma, \Delta)$ which is not fault tolerant against a single processor or a single driver failure can

be converted to a fault-tolerant one by adding 50% more interfaces. Note that, the insertion of generator δ^2 will fail when $\delta^2 = e$. But this corresponds to the two processor case which can easily be handled. For $M(\Gamma, \Delta)$ to tolerate a bus failure or a receiver failure, Δ must have at least three generators. We may have to add one or two extra interfaces per processor depending on whether $|\Delta| = 2$ or 1.

CHAPTER 6

CONSTRUCTION WITH GIVEN NUMBER OF PROCESSORS AND BUSES

An optimal *MBS* realizing a given *IFG* may require a large number of processors and buses. When the *IFG* is vertex symmetric or regular, the number of processors, as well as the number of buses, of the optimal *MBS* is equal to the number of vertices of the *IFG*. Implementing an *MBS* with as many processors and buses as what is needed for optimality may be infeasible due to practical considerations. Therefore, it is imperative to study how well an *MBS* with a given number of buses and/or processors can run the source algorithm(s). Such a system will inevitably suffer some performance penalties.

In this chapter we study how to design an optimal multiple bus system realizing a given *IFG*, when the number of processors and buses in the target *MBS* are specified. In this case, the optimality of the target *MBS* depends on the number of interfaces used. Therefore, in this chapter an *optimal MBS* means an *MBS* with minimum number of interfaces. This is our secondary objective of the dissertation and it is an extension to the work presented in Chapters 2, 3, and 4. Note that the problem at hand differs from mapping a given *IFG* onto an existing *MBS*. In the mapping problem, how processors and buses are interconnected together is specified while in our design problem it is not specified. We set up the interconnection between the given set of processors and/or the buses such that the target *MBS* can realize the original source algorithm at maximum possible speed and the number of interfaces used is minimum.

First we consider the case with given number of processors. We show that the number of processors p must be a factor of the optimal number of processors in order to

utilize them efficiently. We will partition the vertex set of the given *IFG* into p subsets. The optimality criterion is to minimize the number of edges induced by the partition. Since the general partition problem is *NP*-Hard, we consider the case when the *IFG* is vertex symmetric. By partitioning the vertex set of the given *IFG* $C_{\Delta}(\Gamma)$, we will construct a new *IFG* which has p vertices. We state some specific problems associated with the construction of the new *IFG*. We show that unless the vertex partition satisfies certain conditions, those questions cannot be answered properly. Using cosets of group Γ, we will show how such partition can be made. We also show that the resulting *MBS* is regular. Furthermore, we will provide a proper scheduling of the processors which guarantees maximum speed.

Then, we will consider the case with a given number of buses. Here also, for efficient utilization of buses, we show that the specified number of buses b must be a factor of the optimal number of buses. We will show that even when the given *IFG* is vertex symmetric, the optimal *MBS* with a given number of buses may be heterogeneous. Therefore, we may sacrifice certain amount of optimality in order to make the target *MBS* homogeneous. Using cosets of Γ, we will show how to combine buses of the optimal *MBS* $M(\Gamma, \Delta)$ to obtain a new *MBS* which is symmetric and contains b buses.

6.1 *MBS* with Given Number of Processors

In order to construct an *MBS* with p processors realizing a given *IFG* G, we partition the vertex set of G into p subsets such that the subtasks associated with each subset are to be performed by a single processor in the target *MBS*. (Note that each vertex of the *IFG* corresponds to a single subtask.) Consider subset V_1 of vertices of G assigned to processor P_1. The edges induced by subset V_1 will not impose any communications

since both the source and destination subtasks involved in the communication are assigned to the same processor. These edges represent the communication *hidden* by processor P_1. Therefore, the number of edges induced by V_1 is a measure of the amount of communication hidden by processor P_1. Thus, we define an optimal vertex partition as follows.

Definition 6.1: A partition of the vertex set of a given *IFG* is optimal if the total number of edges induced by the partition is maximum.

The optimal partition of a general *IFG* into a given number of subsets is equivalent to the general graph partition problem, and is, therefore, NP-Hard [24], [53], [60], [81]. Besides, we have already demonstrated the importance of symmetric *IFG*s. Therefore, we will only consider the partition of vertex symmetric *IFG*s. Unfortunately, as we demonstrate later, even when the *IFG* is vertex symmetric, finding an optimal vertex partition is not a trivial problem for a given value of p.

We will now show an interesting relation to be held between the given number of processors p and the optimal number of processors $|V(G)|$. Let C_A be the (regular) *CFG* from which the *IFG* G was constructed by an optimal level-disjoint partition. Then $|V(G)| = \alpha(C_A)$. Suppose that C_A consists of n levels. Momentarily assume that $p = |V(G)| - 1$. The source algorithm requires n parallel steps to be executed, where each parallel step consists of $\alpha(C_A) = |V(G)|$ single steps. Since there are only p $(= |V(G)| - 1)$ processors available, it takes two steps for the processors to execute one parallel computation step of the source algorithm. This is true because we assume that all the computational steps take the same amount of time. Also we assume that computations from different levels cannot be interleaved together because of possible data dependencies. Therefore, it takes $2n$ time steps to complete all the computations in the algorithm. Even if we use

$\lceil |V(G)|/2 \rceil$ processors, then the computation time needed is $2n$. Thus, if the specified number of processors is only $|V(G)| - 1$, then the actual number of processors needed for optimal execution is $\lceil |V(G)|/2 \rceil$. This idea is generalized in the following theorem.

Theorem 6.1: If the specified number of processors is p, then only $\lceil |V(G)|/k \rceil$ processors are needed without affecting the performance, where $k = \lceil |V(G)|/p \rceil$. □

Therefore, without loss of generality, we assume that p is a factor of $|V(G)|$. Each subset of the partition is assigned to a unique processor. The size of each subset will determine the number of subtasks allocated to the corresponding processor.

Definition 6.2: A partition of the vertex set of an *IFG* is said to be an *isomorphic vertex partition* if the induced subgraphs of the subsets are isomorphic to one another.

If a vertex partition is not isomorphic, different processors may be assigned different number of subtasks and thus processors may behave differently. Also, the number of outgoing edges of color r from a subset of vertices in G may vary from subset to subset. Suppose all the processors in the new *MBS* are performing computation r. Then the number of times a processor will perform an external communication may be different from processor to processor. Then not only processor execution times are different but buses will also be used inefficiently. Therefore, we are interested in finding an isomorphic vertex partition even at the expense of optimality.

By partitioning the vertex set of the *IFG* G into p subsets of size k, we construct another *IFG* G^k. As one would have expected, the vertices of G^k are the subsets of the partition. We use notation v_i to represent the vertex in G^k corresponding the subset V_i of G, according to a given vertex partition. When we want to make a distinction between the two *IFG*s G and G^k, we will refer to the former as the *principal IFG* and to the latter

as the *derived IFG*. To determine the edge set of G^k, closer attention must be paid to the concurrent data transfers dictated by the principal *IFG G*.

If there exists an edge in $E(G)$ such that its initial vertex is in V_i and its terminal vertex is in V_j, we say that an edge exists from V_i to V_j in G. Suppose that there exists an edge from V_i to V_j in G. Then the processor corresponding to v_i (or, processor v_i), at a certain time during its operation, should send data to the processor corresponding to v_j (or, processor v_j). Conversely, if there are no edges in G from V_i to V_j, then processor v_i does not send data to processor v_j at all. Therefore, we stipulate that $(v_i, v_j) \in E(G^k)$ if and only if there is an edge from V_i to V_j in G. Obviously, there could be more than one edge from V_i to V_j. Those edges are called parallel edges. Interpretation of those parallel edges in G^k involves two important issues, *merging* and *secondary coloring*.

First we address the issue of merging. Suppose there are more than one edge in G from V_i to V_j of the same color, say r. Since these data transfers must be performed sequentially by processors v_i and v_j, they can be represented by a single edge of color r from v_i to v_j in G^k. This is called *merging* of parallel edges.

Definition 6.3: Let there be w edges of color r from V_i to V_j in G. Then the representative edge of color r from v_i to v_j in G^k has *weight w*.

Example 6.1: Suppose there are 2 edges from subset V_1 to subset V_3 and 3 edges from subset V_2 to subset V_4 of color r in G. Assume that the subsets V_1, V_2, V_3, and V_4 in G are all distinct. Representative edges (v_1, v_3) and (v_2, v_4) in G^k are of weights 2 and 3, respectively. See Figure 6.1.

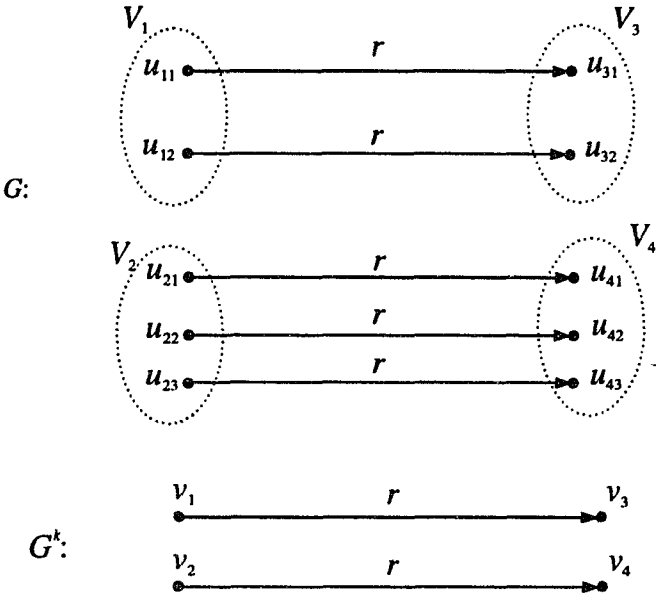


Figure 6.1: Two edges of G^k with different weights.

If the weights of the edges of G^k are different, then bus utilization will be poor. Some buses will stay idle while some others are in operation. This is undesirable since we are interested in an *MBS* with regular features.

Definition 6.4: A *uniform merging* allocates the same weight to every edge in G^k.

With uniform merging, every bus carries the same amount of data concurrently. Therefore, bus utilization is uniform. We are interested in finding a partition on $V(G)$ which allows for uniform merging of parallel edges while constructing the edge set of G^k.

Now we consider the issue of secondary coloring. Let u_1, u_2, u_3, and u_4 be four vertices in G such that (u_1, u_3) and (u_2, u_4) are edges of the same color r. Assume that V_1, V_2, and V_3 are distinct subsets of the partition such that $u_1, u_2 \in V_1$; $u_3 \in V_2$; and $u_4 \in V_3$. Then we have color r edges (v_1, v_2) and (v_1, v_3) in G^k. See Figure 6.2. Should we retain the colors of those two edges? If so, two edges (v_1, v_2) and (v_1, v_3) in G^k demand two distinct buses for the corresponding data transfers. Processor v_1 will perform the subtasks

corresponding to u_1 and u_2 sequentially. Therefore, the two data transfers corresponding to the edges (u_1, u_3) and (u_2, u_4) of G will not occur concurrently and can be carried out via the same bus. Hence, for the optimality of the target *MBS*, different colors must be assigned to edges (v_1, v_2) and (v_1, v_3) in G^k.

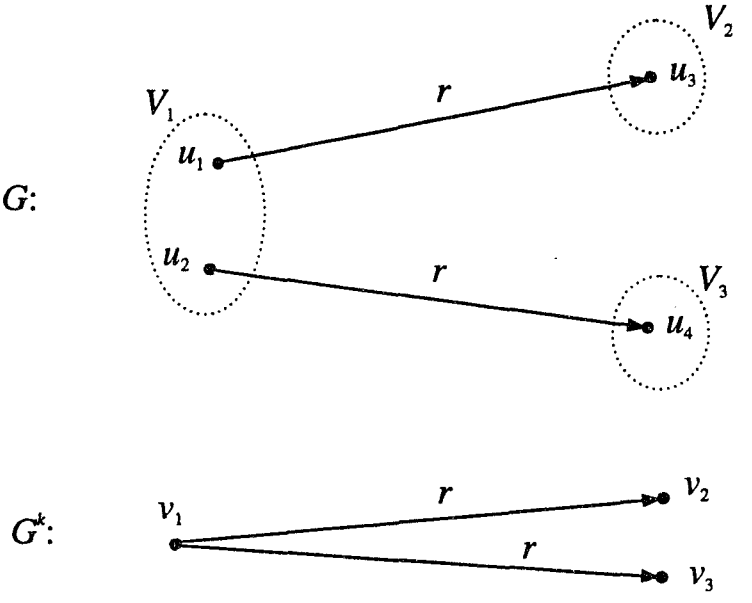


Figure 6.2: Two edges of color r in G^k with the same initial vertex.

We can assign two new colors to edges (v_1, v_2) and (v_1, v_3). Suppose we do this for all edges originating from a vertex for the same color, for every color and every vertex. Then there is no way of finding edges corresponding to potential concurrent data transfers. Therefore, instead of assigning totally new colors to edges (v_1, v_2) and (v_1, v_3) we will assign two distinct *secondary colors* to those edges and retain the primary color r[†]. Two edges having the same primary color and the same secondary color correspond to concurrent data transfers.

[†]Note that this coloring scheme is different from that used for broadcasting in Section 3.3.

Now the question is: what criteria should be used to assign secondary colors to the edges with the same primary color? Different secondary colorings will result in different target *MBS*s. An arbitrary secondary coloring may result in an *MBS* which is heterogeneous.

Definition 6.5: A (secondary) coloring of the edges of G^k is said to be a *regular coloring* if each vertex has distinct colored incoming edges as well as distinct colored outgoing edges according to that coloring.

If possible, we need to find a regular secondary coloring. If the secondary coloring is regular, then we would have made the derived *IFG* regular. In that case, Algorithm 4.1 can be used to find an optimal color partition of the *IFG* G^k. Furthermore, the resulting *MBS* will also be regular. Now, our aim is to partition the vertex set of the principal *IFG* G into k subsets satisfying the following three conditions.

(*a*) Subsets are isomorphic

(*b*) Parallel edge merging is uniform

(*c*) There exists a regular secondary coloring for edges in G^k.

We will now show how to obtain a vertex partitioning of the symmetric *IFG* $C_\Delta(\Gamma)$ satisfying the above three requirements. The partition may not be optimal in general. But it will be optimal in many cases of interest. We assume that there exists a subgroup of Γ of cardinality $k = |\Gamma|/p$. If there are more than one such subgroup, we will select the one containing the largest number of generators from Δ. By selecting the subgroup containing the largest number of generators from Δ, we actually maximize the communication hidden by each processor. We will denote the selected subgroup of Γ by Λ.

Remark: In general, there may not exist any subgroup of Γ whose cardinality is k. If such a subgroup does not exist, we will use the largest subgroup whose size is less than $|\Gamma|/p$. But many of the Cayley color graphs, which are of interest for our purpose, have subgroups of orders equal to almost every factor of $|\Gamma|$.

We partition the vertex set of $C_\Delta(\Gamma)$ into coset subgraphs relative to Λ. This is called *coset partitioning*. First we will show that the coset partition is an isomorphic vertex partition. Second, we will show that coset partitioning yields uniform merging of parallel edges. Third, we will show that there exists a regular secondary coloring corresponding to the coset partitioning.

6.1.1 Isomorphism

Theorem 6.2: Coset partition of $C_\Delta(\Gamma)$ is an isomorphic vertex partition.

Proof: Let Λ be a subgroup of Γ. Consider the coset subgraphs CS_i and CS_j corresponding to the two distinct cosets $\gamma_i\Lambda$ and $\gamma_j\Lambda$, respectively. Let the mapping ψ: $V(CS_i) \rightarrow V(CS_j)$ be defined by $\psi(\gamma) = \gamma_j\gamma_i^{-1}\gamma$. Let (γ_1, γ_2) be an edge of CS_i whose color is, say, r. Then $\gamma_2 = \gamma_1\delta_r$. So, $\psi(\gamma_2) = \gamma_j\gamma_i^{-1}\gamma_2 = \gamma_j\gamma_i^{-1}\gamma_1\delta_r = \psi(\gamma_1)\delta_r$. Since $\gamma_1 \in \gamma_i\Lambda$, it follows that $\psi(\gamma_1) = \gamma_j\gamma_i^{-1}\gamma_1 \in \gamma_j\gamma_i^{-1}\gamma_i\Lambda = \gamma_j\Lambda$. Similarly, $\psi(\gamma_2) \in \gamma_j\Lambda$. Therefore, $(\psi(\gamma_1), \psi(\gamma_2))$ is a color r edge of CS_j. Thus mapping ψ preserves adjacency and colors. Hence, CS_i and CS_j are isomorphic. □

6.1.2 Uniform Merging

Here we show that merging of parallel edges corresponding to coset partitioning is uniform. For that purpose, some properties of cosets and their subgraphs are presented next.

Lemma 6.1: For every pair of coset subgraphs in $C_\Delta(\Gamma)$, there exists a color preserving automorphism which maps the first coset subgraph to the second.

Proof: Let $\gamma_1\Lambda$ and $\gamma_2\Lambda$ be two cosets of Γ. Consider the mapping ψ defined by $\psi(\gamma) = \gamma_1^{-1}\gamma_2\gamma$, where γ is any element in Γ. Clearly, ψ corresponds to a color preserving automorphism which maps left coset $\gamma_1\Lambda$ to left coset $\gamma_2\Lambda$. □

Lemma 6.2: Let Γ be a group and let Λ be a subgroup of Γ. Then two elements γ_1 and γ_2 of Γ belong to the same left coset relative to Λ if and only if $\gamma_1^{-1}\gamma_2 \in \Lambda$.

Proof: First suppose that $\gamma_1^{-1}\gamma_2 \in \Lambda$. Then $\gamma_1^{-1}\gamma_2 = \lambda$ for some element $\lambda \in \Lambda$. It is clear that γ_1 and γ_2 are in the left cosets $\gamma_1\Lambda$ and $\gamma_2\Lambda$, respectively. But $\gamma_2\Lambda = \gamma_1\lambda\Lambda = \gamma_1(\lambda\Lambda) = \gamma_1\Lambda$. Thus γ_1 and γ_2 belong to the same left coset. Next suppose that $\gamma_2 \in \gamma_1\Lambda$. Then, there exists an element $\lambda \in \Lambda$ such that $\gamma_2 = \gamma_1\lambda$. That is, $\gamma_1^{-1}\gamma_2 = \lambda \in \Lambda$. □

Next we present a very interesting relationship among the vertices of $C_\Delta(\Gamma)$ which correspond to parallel, similar color edges between coset subgraphs. Once that relationship is obtained, merging of similar color edges can be done in a straightforward manner.

Theorem 6.3: Let Δ be a generating set for group Γ and Λ be a subgroup of Γ generated by $\Delta_1 \subset \Delta$. Let δ_r be a generator in Δ but not in Δ_1. For an arbitrary element γ_1 of Γ, let $\mathscr{A}_r \subseteq \gamma_1\Lambda$ be the largest cardinality subset of the coset $\gamma_1\Lambda$, which satisfies the condition that all the elements in the set $\mathscr{A}_r\delta_r$ belong to the same left coset of Γ relative to Λ. Then $\mathscr{A}_r$ is a left coset of Γ relative to a subgroup of Λ. Furthermore that subgroup is unique.

Proof: Since $\mathscr{A}_r$ is a subset of $\gamma_1\Lambda$, we can write $\mathscr{A}_r = \gamma_1\mathscr{B}_r$, where $\mathscr{B}_r$ is a subset of Λ. Let θ be an arbitrary element of $\mathscr{B}_r$. Let $\mathscr{C}_r = \theta^{-1}\mathscr{B}_r$, that is, $\mathscr{A}_r = \gamma_1\theta\mathscr{C}_r$. According to the hypothesis of the theorem, all elements in set $\mathscr{A}_r\delta_r$ belong to the same left coset of Γ relative to Λ. Let that left coset be $\gamma_2\Lambda$. Therefore, $\gamma_1\theta\mathscr{C}_r\delta_r \subseteq \gamma_2\Lambda$. Now we make the following claim.

Claim: $\mathscr{C}_r$ is a group.

Proof: Since $\mathscr{C}_r = \theta^{-1}\mathscr{B}_r$ and $\theta \in \mathscr{B}_r$, $\mathscr{C}_r$ contains the identity element e. Also, as a result, $\gamma_1\theta\delta_r \in \gamma_2\Lambda$. Let ς_1 and ς_2 be two arbitrary elements in $\mathscr{C}_r$. Since $\gamma_1\theta\mathscr{C}_r\delta_r \subseteq \gamma_2\Lambda$, we have $\gamma_1\theta\varsigma_1\delta_r \in \gamma_2\Lambda$. Hence, the elements $\gamma_1\theta\delta_r$ and $\gamma_1\theta\varsigma_1\delta_r$ belong to the same left coset and therefore (from Lemma 6.2), $(\gamma_1\theta\delta_r)^{-1}(\gamma_1\theta\varsigma_1\delta_r) = \delta_r^{-1}\varsigma_1\delta_r \in \Lambda$. Similarly, $\delta_r^{-1}\varsigma_2\delta_r \in \Lambda$. Therefore,

$$(\delta_r^{-1}\varsigma_1\delta_r)(\delta_r^{-1}\varsigma_2\delta_r) = \delta_r^{-1}\varsigma_1\varsigma_2\delta_r \in \Lambda.$$

That is,

$$(\gamma_1\theta\delta_r)^{-1}(\gamma_1\theta\varsigma_1\varsigma_2\delta_r) \in \Lambda.$$

Therefore, from Lemma 6.2, both $\gamma_1\theta\delta_r$ and $\gamma_1\theta\varsigma_1\varsigma_2\delta_r$ belong to the same left coset of Γ. But $\gamma_1\theta\delta_r$ belongs to left coset $\gamma_2\Lambda$. Therefore, $\gamma_1\theta\varsigma_1\varsigma_2\delta_r$ also belongs to left coset $\gamma_2\Lambda$. From the definition, we have $\mathscr{C}_r = \theta^{-1}\mathscr{B}_r$ and $\theta \in \mathscr{B}_r$. Therefore, since $\mathscr{B}_r$ is a subset of Λ, $\mathscr{C}_r$ is also a subset of Λ. Now, $\theta \in \mathscr{B}_r$ and $\varsigma_1, \varsigma_2 \in \mathscr{C}_r$. Hence, $\theta\varsigma_1\varsigma_2 \in \Lambda$, and therefore, $\gamma_1\theta\varsigma_1\varsigma_2 \in \gamma_1\Lambda$. We have already shown that $\gamma_1\theta\varsigma_1\varsigma_2\delta_r \in \gamma_2\Lambda$. Since $\mathscr{A}_r$ is assumed to be the largest subset of $\gamma_1\Lambda$ with the required property, $\gamma_1\theta\varsigma_1\varsigma_2$ must belong to $\mathscr{A}_r$. That is, $\gamma_1\theta\varsigma_1\varsigma_2 \in \gamma_1\theta\mathscr{C}_r$. Therefore, $\varsigma_1\varsigma_2 \in \mathscr{C}_r$. Let ς be any element of $\mathscr{C}_r$. Similar to the above reasoning, we can show $\varsigma^{-1} \in \mathscr{C}_r$. Hence $\mathscr{C}_r$ is a group.

Now, $\mathscr{A}_r = \gamma_1\theta\mathscr{C}_r$. Therefore, $\mathscr{A}_r$ is a left coset of Γ relative to $\mathscr{C}_r$. Since $\mathscr{A}_r$ is the largest subset of $\gamma_1\Lambda$ with the required property, group $\mathscr{C}_r$ is unique. □

Definition 6.6: The unique subgroup stated in Theorem 6.3 is denoted by Λ_r.

The above theorem states the existence of a unique subgroup Λ_r of Λ for every color r. If for a certain color r there are no parallel edges, that is, if there exists at most

one edge of color r from one coset subgraph (relative to Λ) to another in $C_\Delta(\Gamma)$, then Λ_r is the trivial group e.

Now we will show that merging of parallel edges is uniform for the coset partitioning. Suppose there are m_r parallel edges of color r from one coset subgraph to another in $C_\Delta(\Gamma)$. Then, from Theorem 6.3, there is a subgroup Λ_r of Λ of order m_r. Since Λ_r is unique, by symmetry (Lemma 6.1), for every edge of color r in $C_\Delta(\Gamma)$, there are m_r parallel edges. We can merge parallel edges of color m_r to produce a single edge of color r and weight m_r in the derived *IFG*. Thus the merging is uniform. Note that edges belonging to different colors may have different number of occurrences of parallel edges. This doesn't create problems since data transfers corresponding different color edges are carried out sequentially.

Definition 6.7: The derived *IFG* obtained by coset partitioning $C_\Delta(\Gamma)$ relative subgroup Λ is denoted by $C_\Delta(\Gamma, \Lambda)$.

An edge of color r in $C_\Delta(\Gamma, \Lambda)$ corresponds to $|\Lambda_r|$ color r edges of $C_\Delta(\Gamma)$, where Λ_r is as given in Definition 6.6. From Lagrange's Theorem [113], $|\Lambda_r|$ evenly divides $|\Lambda|$ (recall that $|\Lambda| = k$). Thus, there are $|\Lambda|/\Lambda_r|$ r-neighbors for every vertex of $C_\Delta(\Gamma)$. When we need to emphasize the correspondence between $C_\Delta(\Gamma)$ and $C_\Delta(\Gamma, \Lambda)$, we will sometimes denote the vertices of $C_\Delta(\Gamma, \Lambda)$ by cosets $\gamma\Lambda$.

Definition 6.8: Let v_1 and v_2 be two vertices of $C_\Delta(\Gamma, \Lambda)$ such that there is an edge of color r from v_1 to v_2. Then v_2 is said to be an *r-neighbor* of v_1.

In $C_\Delta(\Gamma, \Lambda)$, there may still exist parallel edges of different colors. This occurs when there are edges from one coset subgraph to another (relative to a subgroup Λ) in

$C_\Delta(\Gamma)$ belonging to two different colors. We will show how these situation can be easily tackled. The following lemma from basic group theory is now in order [113].

Lemma 6.3: Let Λ be a subgroup of the group Γ. Let γ be an arbitrary element of Γ. Then $\gamma^{-1}\Lambda\gamma$ is also a subgroup of Γ.

Theorem 6.4: Suppose that there is an edge of color r in parallel with an edge of color s in $C_\Delta(\Gamma)$ from coset subgraph $\gamma_i\Lambda$ to coset subgraph $\gamma_j\Lambda$ whose initial vertices are $\gamma_i\lambda_r$ and $\gamma_i\lambda_s$ (not necessarily distinct), respectively. Then $\Lambda_s = \lambda_s^{-1}\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s$, where Λ_r and Λ_s are as defined in Definition 6.6.

Proof: Since $\gamma_i\lambda_r, \gamma_i\lambda_s \in \gamma_i\Lambda$, it is clear that $\lambda_r, \lambda_s \in \Lambda$. Furthermore, since both edges terminate in $\gamma_j\Lambda$, we get $\gamma_i\lambda_r\delta_r, \gamma_i\lambda_s\delta_s \in \gamma_j\Lambda$. In fact, there are $|\Lambda_r|$ parallel edges of color r from coset subgraph $\gamma_i\Lambda$ to coset subgraph $\gamma_j\Lambda$ of $C_\Delta(\Gamma)$. Furthermore, subgroup Λ_r of Λ satisfies the condition that, all the elements in $\Lambda_r\delta_r$ belong to the same coset of Γ relative to Λ. Since $\gamma_i\lambda_r\delta_r \in \gamma_j\Lambda$, it follows that $\delta_r \in \lambda_r^{-1}\gamma_i\text{-}1\gamma_j\Lambda$. But $\delta_r \in \Lambda_r\delta_r$ (because $e \in \Lambda_r$). Therefore, one element of $\Lambda_r\delta_r$ is in the coset $\lambda_r^{-1}\gamma_i\text{-}1\gamma_j\Lambda$. Since all elements of $\Lambda_r\delta_r$ should belong to the same coset, we get $\Lambda_r\delta_r \subseteq \lambda_r^{-1}\gamma_i\text{-}1\gamma_j\Lambda$. Hence, $\gamma_i\lambda_r\Lambda_r\delta_r \subseteq \gamma_j\Lambda$. Therefore, $\gamma_i\lambda_r t\delta_r \in \gamma_j\Lambda$, for any arbitrary element $t \in \Lambda_r$. Hence, both $\gamma_i\lambda_r\delta_r$ and $\gamma_i\lambda_r t\delta_r$ belong to the same coset $\gamma_j\Lambda$, and therefore, from Lemma 6.2,

$$(\gamma_i\lambda_r\delta_r)^{-1}(\gamma_i\lambda_r t\delta_r) = \delta_r^{-1}t\delta_r \in \Lambda.$$

Furthermore, both $\gamma_i\lambda_r\delta_r$ and $\gamma_i\lambda_s\delta_s$ also belong to the same coset $\gamma_j\Lambda$. Hence,

$$(\gamma_i\lambda_r\delta_r)^{-1}(\gamma_i\lambda_s\delta_s) = \delta_r^{-1}\lambda_r^{-1}\lambda_s\delta_s \in \Lambda.$$

We have just shown that both $\delta_r^{-1}t\delta_r$ and $\delta_r^{-1}\lambda_r^{-1}\lambda_s\delta_s$ are elements of group Λ. Therefore, their product $(\delta_r^{-1}t\delta_r)(\delta_r^{-1}\lambda_r^{-1}\lambda_s\delta_s) = \delta_r^{-1}t\lambda_r^{-1}\lambda_s\delta_s$ is also an element of Λ. But

$$(\gamma_i\lambda_r\delta_r)^{-1}(\gamma_i\lambda_r t\lambda_r^{-1}\lambda_s\delta_s) = (\delta_r^{-1}\lambda_r^{-1}\gamma_i\text{-}1)(\gamma_i\lambda_r t\lambda_s^{-1}\lambda_s\delta_s) = \delta_r^{-1}t\lambda_r^{-1}\lambda_s\delta_s.$$

Therefore, from Lemma 6.2, both $\gamma_i\lambda_r\delta_r$ and $\gamma_i\lambda_r t\lambda_r^{-1}\lambda_s\delta_s$ should belong to the same coset. Therefore, since $\gamma_i\lambda_r\delta_r$ belongs to coset $\gamma_j\Lambda$, $\gamma_i\lambda_r t\lambda_s^{-1}\lambda_s\delta_s$ should also belong to coset $\gamma_j\Lambda$. Since t is an arbitrary element of Λ_r, we can deduce that $\gamma_i\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s\delta_s \subseteq \gamma_j\Lambda$. We have already seen that $\lambda_r, \lambda_s \in \Lambda$. Since Λ_r is a subgroup of Λ, every element of Λ_r must also be an element of Λ. Therefore, $\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s \subseteq \Lambda$. In other words, $\gamma_i\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s \subseteq \gamma_j\Lambda$. Therefore edges of color s originating from vertices in the set $\gamma_i\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s$ will all terminate at vertices in the coset $\gamma_j\Lambda$. Clearly, $\gamma_i\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s$ is the largest subset of $\gamma_i\Lambda$ which satisfies the relation $\gamma_i\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s\delta_s \subseteq \gamma_j\Lambda$. According to Theorem 6.3, $\gamma_i\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s$ must be a (left) coset of the subgroup Λ_s of Λ. We can write $\gamma_i\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s = \gamma_i\lambda_s(\lambda_s^{-1}\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s)$. From Lemma 6.3, $\lambda_s^{-1}\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s = (\lambda_r^{-1}\lambda_s)^{-1}\Lambda_r(\lambda_r^{-1}\lambda_s)$ is a subgroup of Λ. Obviously,

$\Lambda_s = \lambda_s^{-1}\lambda_r\Lambda_r\lambda_r^{-1}\lambda_s$. □

Corollary: Suppose that there is an edge of color r in parallel with an edge of color s in $C_\Delta(\Gamma)$ from coset subgraph $\gamma_i\Lambda$ to coset subgraph $\gamma_j\Lambda$. Then $\Lambda_r = \Lambda_s$.

Therefore, if there are two parallel edges of colors r and s from vertex v_1 to vertex v_2 in $C_\Delta(\Gamma, \Lambda)$, then all edges of colors r and s of $C_\Delta(\Gamma, \Lambda)$ will be of the same weight. Furthermore, it is clear that for every vertex in $C_\Delta(\Gamma, \Lambda)$, the number of r-neighbors is equal to the number of s-neighbors. The following theorem states a much stronger result.

Theorem 6.5: Let a certain edge of color r be parallel with another edge of color s in $C_\Delta(\Gamma, \Lambda)$. Then for every vertex in $C_\Delta(\Gamma, \Lambda)$, an r-neighbor is also an s-neighbor, and vice versa.

Proof: Suppose that there is an edge of color r in parallel with an edge of color s from vertex $\gamma_1\Lambda$ to vertex $\gamma_2\Lambda$ in $C_\Delta(\Gamma, \Lambda)$. Let the initial vertices of those edges of color r and s in $C_\Delta(\Gamma)$ be $\gamma_1\lambda_r$ and $\gamma_1\lambda_s$, respectively. Let $\gamma_i\Lambda$ and $\gamma_j\Lambda$ be two arbitrary vertices in

$C_\Delta(\Gamma, \Lambda)$. We need to prove that $\gamma_j\Lambda$ is an r-neighbor of $\gamma_j\Lambda$ if and only if $\gamma_j\Lambda$ is an s-neighbor of $\gamma_j\Lambda$. Suppose that $\gamma_j\Lambda$ is an r-neighbor of $\gamma_i\Lambda$. Then there exists an element $\lambda \in \Lambda$ such that $\gamma_i\lambda\delta_r \in \gamma_j\Lambda$. From the hypothesis, both $\gamma_1\lambda_r\delta_r$ and $\gamma_1\lambda_s\delta_s$ belong to the same left coset $\gamma_2\Lambda$. Therefore, from Lemma 6.2, $(\gamma_1\lambda_r\delta_r)^{-1}(\gamma_1\lambda_s\delta_s) = \delta_r^{-1}\lambda_r^{-1}\lambda_s\delta_s \in \Lambda$. But,

$$(\gamma_i\lambda\delta_r)^{-1}(\gamma_i\lambda\lambda_r^{-1}\lambda_s\delta_s) = \delta_r^{-1}\lambda_r^{-1}\lambda_s\delta_s.$$

Therefore (from Lemma 6.2), $\gamma_i\lambda\delta_r$ and $\gamma_i\lambda\lambda_r^{-1}\lambda_s\delta_s$ belong to the same left coset. Since $\gamma_i\lambda\delta_r \in \gamma_j\Lambda$, we have, $\gamma_i\lambda\lambda_r^{-1}\lambda_s\delta_s \in \gamma_j\Lambda$. The three elements λ, λ_r, and λ_s all belong to subgroup Λ. Therefore $\lambda\lambda_r^{-1}\lambda_s$ also belongs to Λ. Hence, $\gamma_i\lambda\lambda_r^{-1}\lambda_s \in \gamma_i\Lambda$. So, there exists an edge from $\gamma_i\Lambda$ to $\gamma_j\Lambda$ of color s. Hence $\gamma_j\Lambda$ is an s-neighbor of $\gamma_i\Lambda$. Similarly, we can show that $\gamma_j\Lambda$ is an r-neighbor of $\gamma_i\Lambda$ if $\gamma_j\Lambda$ is an s-neighbor of $\gamma_i\Lambda$. □

The following corollary is a direct consequence of the above two theorems.

Corollary: Let a certain edge of color r be parallel with an edge of color s in $C_\Delta(\Gamma, \Lambda)$. Then, for every edge of color r in $C_\Delta(\Gamma, \Lambda)$, there exists a distinct parallel edge of color s.

Suppose an edge of color r is parallel with an edge of color s in $C_\Delta(\Gamma, \Lambda)$. Data transfers corresponding to edges of color s can be carried out using paths allocated for data transfers corresponding to edges of color r. We will say that edges of color s can be *masked* by edges of color r. Therefore, all edges of color s can be removed from $C_\Delta(\Gamma, \Lambda)$ without affecting the target *MBS*. However, the data transfers corresponding to edges of color s are still required by the original algorithm. The removal of color s edges does not mean that those data transfers are non existent. What we actually do is to let the data transfers corresponding to color s edges use the connections established for the edges of color r.

6.1.3 Regular Secondary Coloring

Now we show that a regular secondary coloring of the edges of $C_\Delta(\Gamma, \Lambda)$ can be performed. We will consider two cases. The first case is when Λ is a normal subgroup of Γ (see Definition 5.4). In that case no secondary coloring is necessary. The second case is when Λ is not a normal subgroup of Γ. In that case, secondary coloring may be necessary.

6.1.3.1 Λ is a Normal Subgroup of Γ

If Λ is a normal subgroup of Γ, then $\Lambda\delta_r = \delta_r\Lambda$ for any generator δ_r. Therefore, $\gamma\Lambda\delta_r = \delta_r\gamma\Lambda$, $\gamma \in \Gamma$. Therefore, in $C_\Delta(\Gamma)$, all edges of color r originating from one coset subgraph terminate at the same coset subgraph, for $\delta_r \in \Delta$. In other words, $\Lambda_r = \Lambda$. Therefore, every vertex of $C_\Delta(\Gamma, \Lambda)$ has only one outgoing edge of color r. Thus, a secondary coloring is not necessary. The following is a well known theorem in group theory [113].

Theorem 6.6: Let Γ be group and Λ be a normal subgroup of it. Then left (right) cosets of Γ relative to Λ forms a group denoted by Γ/Λ. □

The group Γ/Λ is called the *quotient group*. Let $\Delta_1 \subset \Delta$ be the subset of generators for Λ. Then, the derived *IFG* $C_\Delta(\Gamma, \Lambda)$ is the Cayley color graph associated with group Γ/Λ and the generating set $\Delta - \Delta_1$.

Lemma 6.4: The elements in the generating set $\Delta - \Delta_1$ corresponding to the quotient group Γ/Λ are non-redundant.

Proof: Suppose that there is a redundant generator $\delta \in \Delta - \Delta_1$ associated with group Γ/Λ. Also suppose that generator δ transforms an element in coset $\gamma_i\Lambda$ to an element in coset $\gamma_j\Lambda$. Then, $\gamma_i\Lambda\delta = \gamma_j\Lambda$. Therefore, element $\gamma_j\Lambda \in \Gamma/\Lambda$ can be obtained from element $\gamma_i\Lambda$

$\in \Gamma$ by right multiplication by δ. Since δ is assumed to be a redundant generator for Γ/Λ, $\gamma_j\Lambda \in \Gamma/\Lambda$ can be obtained from $\gamma_i\Lambda \in \Gamma/\Lambda$ by right multiplying it by $h_1h_2...h_x$, where h_i $(1 \le i \le x)$ is an element selected from set $\Delta - \Delta_1$ and their inverses; that is, $\gamma_j\Lambda = (\gamma_i\Lambda)h_1h_2...h_x$. Let γ_1 be an element in coset $\gamma_i\Lambda$. Then $\gamma_1\delta = \gamma_2$ is an element in coset $\gamma_j\Lambda$. Furthermore, $\gamma_1h_1h_2...h_x$ is in coset $\gamma_i\Lambda h_1h_2...h_x = \gamma_j\Lambda$. Since both γ_2 and $\gamma_1h_1h_2...h_x$ belong to the same coset $\gamma_j\Lambda$ and subgraphs corresponding to cosets are assumed to be connected, it follows that $\gamma_2 = \gamma_1h_1h_2...h_xg_1g_2...g_y$, where g_i (for $1 < i \le y$) is an element selected from Δ_1 and their inverses. Therefore, $\delta = h_1h_2...h_xg_1g_2...g_y$, that is, δ can be expressed as the product of the remaining generators in Δ. This is not possible since δ is not redundant in Δ with respect to group Γ. Hence elements in $\Delta - \Delta_1$ are non redundant with respect to the quotient group Γ/Λ. □

Therefore, if Λ is a normal subgroup of Γ, we can find an optimal color partitioning of the *IFG* $C_\Delta(\Gamma, \Lambda)$ using the method described in Section 4.2.

Example 6.2: Consider the 16-node Cayley color graph $G_{Q(4)}$ shown in Figure 6.3. The generators are $\delta_1 = 0001$, $\delta_2 = 0010$, $\delta_3 = 0100$, and $\delta_4 = 1000$. Different line styles are used for different generators. Also since $\delta_i = \delta_i^{-1}$, $1 \le i \le 4$, undirected edges are used. The number of processors needed for the optimal *MBS* $M_{Q(4)}$ is 16. Let the specified number of processors be $p = 4$. Then $k = 16/4 = 4$. Let Λ' be the 4-element subgroup of $\Gamma_{Q(4)}$ generated by $\{\delta_1, \delta_2\}$. It is clear that Λ' is a normal subgroup of $\Gamma_{Q(4)}$. Figure 6.4 shows the corresponding coset subgraphs (enclosed in dotted curves). Figure 6.5 shows the derived *IFG* and Figure 6.6 shows the corresponding *MBS* containing four processors and four buses.

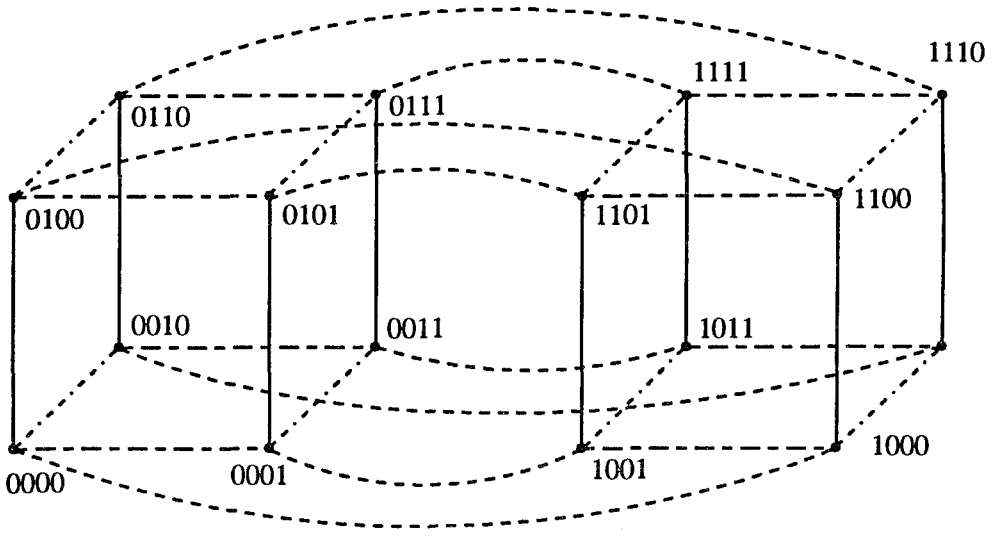


Figure 6.3: Principal *IFG* $G_{Q(4)}$.

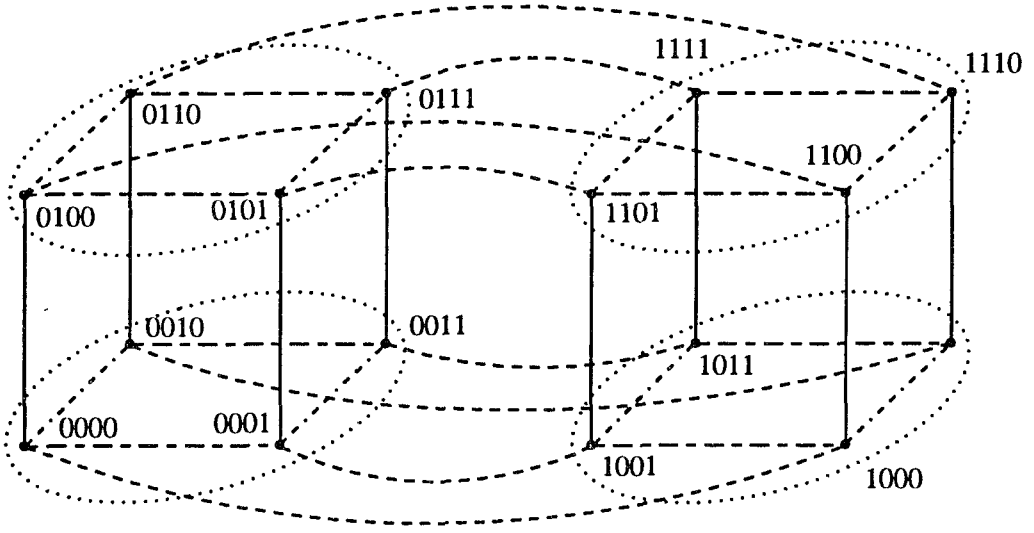


Figure 6.4: Vertex partition of the principal *IFG* $G_{Q(4)}$.

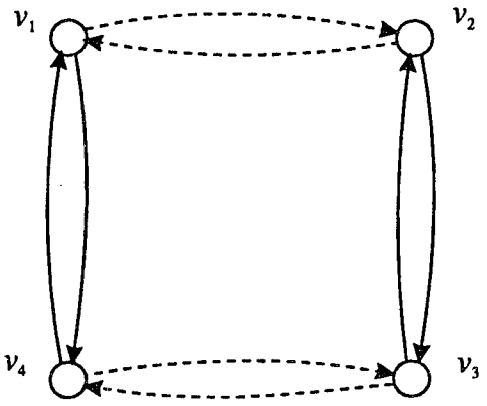


Figure 6.5: The *IFG* derived from Figure 6.4.

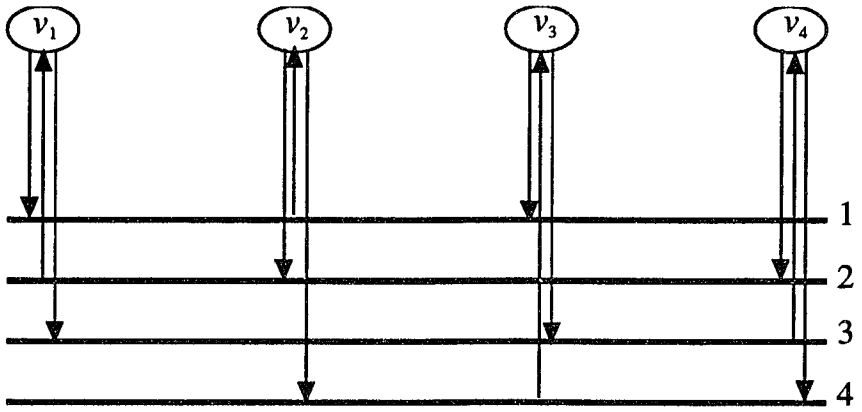


Figure 6.6: The optimal *MBS* corresponding to the derived *IFG* in Figure 6.5.

6.1.3.2 Λ is not a Normal Subgroup of Γ

When Λ is not a normal subgroup of Γ, Λ_r can be a proper subgroup of Λ. In that case, $C_\Delta(\Gamma, \Lambda)$ has more than one edge of the same color originating from the same vertex. Thus secondary coloring is necessary. Can we find a regular secondary coloring for $C_\Delta(\Gamma, \Lambda)$?

Each vertex has $k/|\Lambda_r|$ outgoing edges of (primary) color r. We can arbitrarily assign $k/|\Lambda_r|$ secondary colors to those edges. We can repeat this for every vertex. This

will guarantee that no vertex has more than one outgoing edge of a given primary and a given secondary color. So, this method guarantees that every vertex has outgoing edges of distinct colors. But this method does not, in general, guarantee that each vertex has incoming edges of distinct colors. In the following, we prove that a regular secondary coloring always exists for $C_\Delta(\Gamma, \Lambda)$.

Theorem 6.7: There exists a regular coloring for $G_\Delta(\Gamma, \Lambda)$.

Proof: We will provide a constructive proof. We will only show how to insert secondary colors to edges with primary color r. The same method can be used with every primary color. Construct the bipartite graph G_T^r as follows. For every vertex v_i of $C_\Delta(\Gamma, \Lambda)$, there exist two vertices v_i^{in} and v_i^{out} in G_T^r, and vice versa. For every edge (v_i, v_j) of color r in $C_\Delta(\Gamma, \Lambda)$, there exists an undirected edge (v_i^{out}, v_i^{in}) in G_T^r. It is clear that G_T^r is a bipartite graph whose first partite set is $\{v_i^{in} : 1 \le i \le |\Gamma|\}$ and the second partite set is $\{v_i^{out} : 1 \le i \le |\Gamma|\}$. Furthermore, since each vertex of $C_\Delta(\Gamma, \Lambda)$ has $|\Lambda_r|$ incoming edges and $|\Lambda_r|$ outgoing edges of color r, G_T^r is a regular bipartite graph. Therefore, G_T^r has a perfect matching [30].

Let M_1 be a perfect matching in G_T^r. For every element $(v_i^{out}, v_i^{in}) \in M_1$, assign secondary color 1 to edge (v_i, v_j) of $C_\Delta(\Gamma, \Lambda)$. Now remove all edges of G_T^r belonging to M_1. Bipartite graph G_T^r remains regular since the degree of each vertex is one less than before. Let M_2 be a perfect matching of G_T^r after the removal of the edges. Assign secondary color 2 to every edge of $C_\Delta(\Gamma, \Lambda)$ corresponding to M_2. Repeat this procedure until all edges of G_T^r are exhausted.

We will now show that the assigned secondary coloring is regular. There is exactly one edge in matching M_l, $l = 1, 2, \ldots$, which is incident with vertex v_i^{out} of G_T^r.

Therefore, there is exactly one edge of primary color r and secondary color l directed away from every vertex of $C_\Delta(\Gamma, \Lambda)$. Similarly, there is exactly one edge of primary color r and secondary color l directed towards every vertex of $C_\Delta(\Gamma, \Lambda)$. Thus the secondary coloring is regular. □

The proof of the above theorem also describes a method to find a regular coloring for $C_\Delta(\Gamma, \Lambda)$. Next we provide an example for the case when Λ is not a normal subgroup of Γ.

Example 6.3: Figure 6.7 shows the Cayley color graph associated with the symmetric group with four symbols, denoted by $\Gamma_{P(4)}$, and three generators. The four symbols are *a*, *b*, *c*, and *d*. The three generators are $\delta_1 = bacd$, $\delta_2 = cbad$, and $\delta_3 = dcba$. Let $\Delta_{P(4)} = \{\delta_1, \delta_2, \delta_3\}$. Since each generator is its self inverse, we represent bidirectional edges by undirected edges for convenience. Solid, broken. and dotted lines correspond to the generators δ_1, δ_2, and δ_3 respectively. In fact, the graph in Figure 6.7 is the *pancake graph* associated with four symbols (if we ignore the edge colors). Since $C_{\Delta_{P(4)}}(\Gamma_{P(4)})$ has 24 vertices, the optimal multiple bus system $M(\Gamma_{P(4)}, \Delta_{P(4)})$ requires 24 processors and 24 buses. Let us construct an *MBS* with only four processors, that is, $p = 4$. Therefore, we need to partition the vertex set of $C_{\Delta_{P(4)}}(\Gamma_{P(4)})$ into four subsets such that each subset has 6 vertices. Clearly, $\{abcd, bacd, cabd, acbd, bcad, cbad\}$ is a six element subgroup of Γ. Let that subgroup be Λ. The subgraph corresponding to Λ has 6 vertices and six edges. Coset subgraphs of $C_{\Delta_{P(4)}}(\Gamma_{P(4)})$ relative to Λ are the hexagons in Figure 6.7 connected by solid and broken lines. There are two dotted line edges (edges corresponding to generator δ_3) from each coset subgraph to every other coset subgraph. Figure 6.8 shows $C_{\Delta_{P(4)}}(\Gamma_{P(4)}, \Lambda)$ after merging the parallel edges. Even though we could have used

undirected edges to represent bidirectional edges, we need to explicitly represent directed edges in order to assign secondary colors.

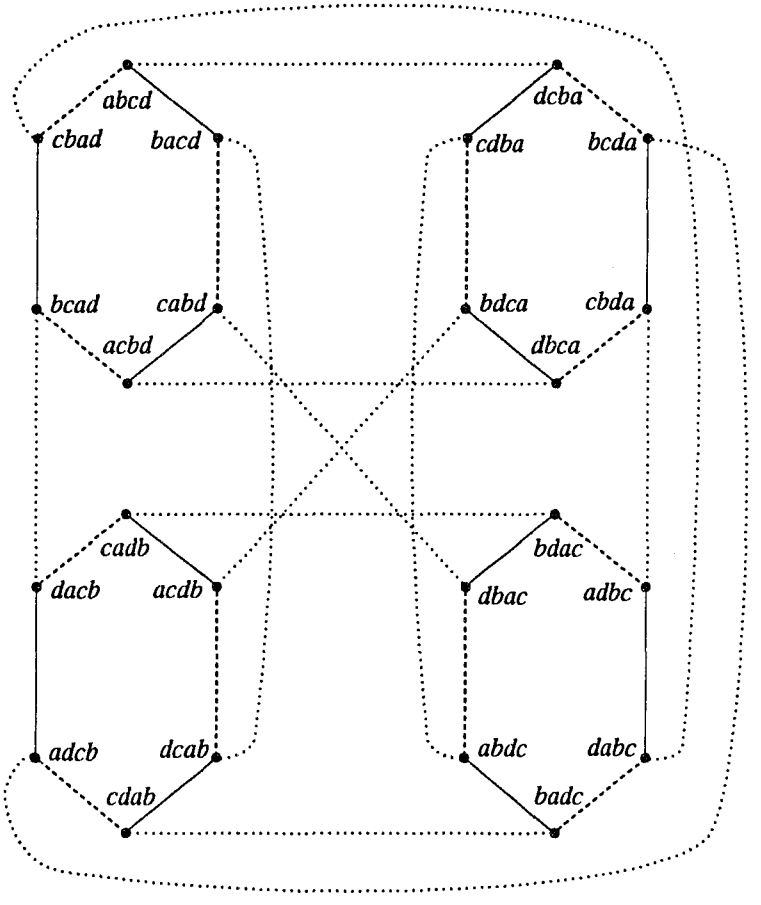


Figure 6.7: The principal *IFG* $C_{\Delta_{P(4)}}(\Gamma_{P(4)})$.

Each directed edge in Figure 6.8 has a weight of 2. This is because one edge of $C_{\Delta_{P(4)}}(\Gamma_{P(4)}, \Lambda)$ is obtained by merging two edges of $C_{\Delta_{P(4)}}(\Gamma_{P(4)})$. In Figure 6.8, each vertex has 3 outgoing edges of the same (primary) color. Figure 6.9 shows the corresponding $C_{\Delta_{P(4)}}(\Gamma_{P(4)}, \Lambda)$ after assigning a regular secondary coloring. Solid, dotted, and broken lines correspond to the three secondary colors. It can be noticed that the derived *IFG* in Figure 6.9 is vertex symmetric. Therefore, according to Theorem 4.2, the

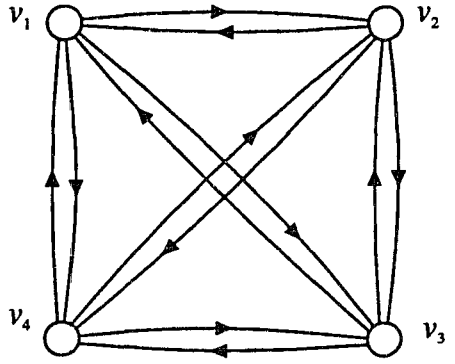


Figure 6.8: Derived *IFG* $C_{\Delta_{P(4)}}(\Gamma_{P(4)}, \Lambda)$ without secondary colors.

graph in Figure 6.9 is the Cayley color graph associated with a group and its generating set. Since in general $C_\Delta(\Gamma, \Lambda)$ may not be vertex symmetric, we should not pay attention to the group and its generating set for the *IFG* in Figure 6.9. What we are concerned with is its regularity. Clearly, the derived *IFG* in Figure 6.9 belongs to class $\mathscr{E}$ (see Definition 4.4). Therefore, an optimal color partition can be found using Algorithm 4.1. The subgraphs induced by the subsets of an optimal color partition of $C_{\Delta_{P(4)}}(\Gamma_{P(4)}, \Lambda)$ are shown in Figure 6.10. Figure 6.11 shows the corresponding multiple bus system. It has 4 processors and 4 buses.

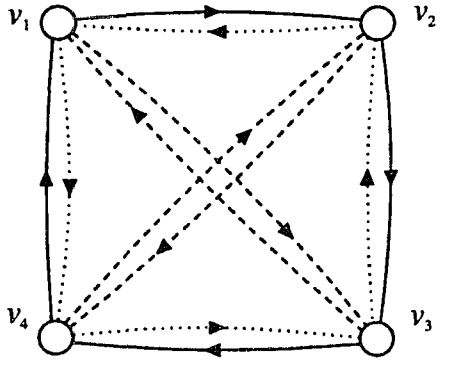


Figure 6.9: The *IFG* $C_{\Delta_{P(4)}}(\Gamma_{P(4)}, \Lambda)$ with secondary colors.

It is to be noted that, irrespective of whether or not Λ is a normal subgroup of Γ, the optimal *MBS* realizing $C_{\Delta}(\Gamma, \Lambda)$ has the same number of processors and buses. Therefore, when we decrease the number processors, the number of buses will automatically decrease by the same factor.

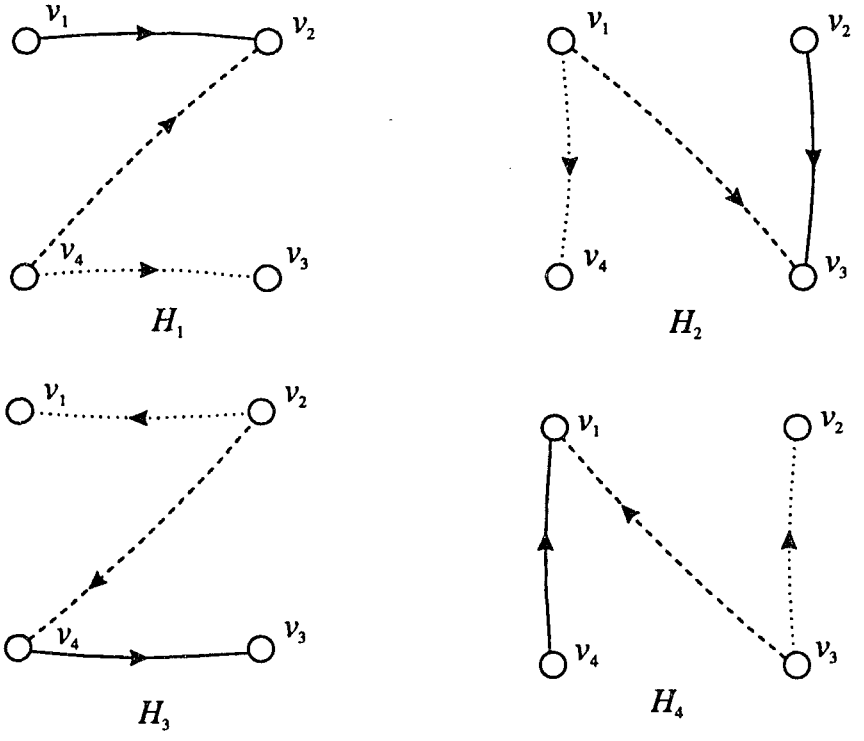


Figure 6.10: Optimal color partition of the derived *IFG* in Figure 6.9.

6.1.4 Scheduling

Once the *MBS* has been constructed, processors must be properly scheduled in order to run the source algorithm(s) at the maximum speed. Computations executed by a processor correspond to vertices in the original *CFG*. The set of computations (one

computation from each level) represented by a single vertex in the principal *IFG* $C_\Delta(\Gamma)$ is a subtask (see Definition 2.2). Since one vertex of $C_\Delta(\Gamma, \Lambda)$ corresponds to a set of vertices of $C_\Delta(\Gamma)$, each processor in the target *MBS* is assigned a set of subtasks. A certain processor *P* will first perform computation 1 (computation at level 1) of each of its subtasks. We will say that *P* is executing its subtasks at level 1. Concurrently, every other processor must be executing computations at level 1 of all of its subtasks. After every processor has finished executing its subtasks at level 1, they must execute all their subtasks at level 2 (computation 2 of each of its subtasks), and so on. At a certain level, what is the order of execution of the subtasks by a processor? In the optimal *MBS* $M(\Gamma, \Delta)$, each processor was assigned a single subtask. Therefore, the problem of the order of execution of the subtasks did not arise there.

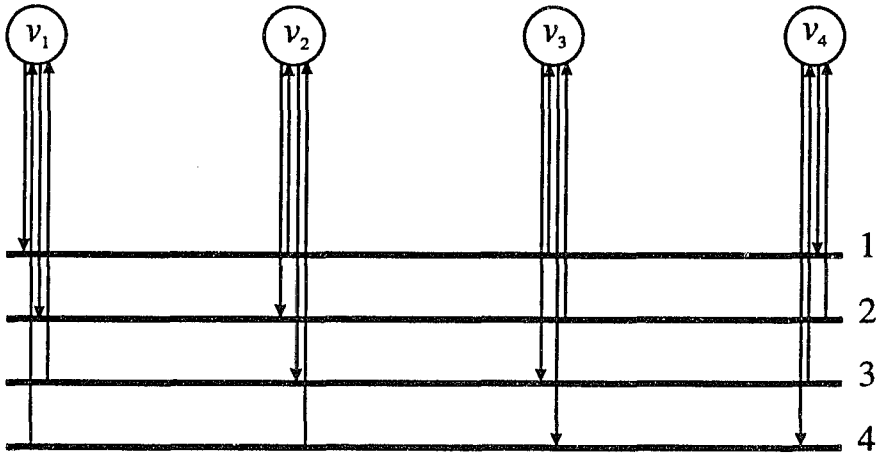


Figure 6.11: The *MBS* corresponding to the color partition in Figure 6.10.

If the communications are ignored, the order of execution of the subtasks is immaterial provided that every processor is executing at the same level concurrently. The governing factor for proper ordering of the subtasks is the communication. Therefore, we need to order the subtasks according to the way communications are performed. We need

only to consider processors executing at a particular level r. The same scheduling policy can be applied to every level. Let P_i be the processor assigned to vertex v_i of $C_\Delta(\Gamma, \Lambda)$, $1 \le i \le p$. Each vertex v_i of $C_\Delta(\Gamma, \Lambda)$ corresponds to a coset $\gamma_i\Lambda$ of group Γ. There are $|\Lambda_r|$ parallel edges originating from coset subgraph $\gamma_i\Lambda$ of $C_\Delta(\Gamma)$. All these edges are represented by a single edge of $C_\Delta(\Gamma, \Lambda)$ with primary color r and a certain secondary color. Therefore, a proper scheduling should be such that the tails of the aforementioned $|\Lambda_r|$ parallel edges correspond to consecutive subtasks executed by processor P_i. Towards that end, we give the following definition.

Definition 6.9: Let $(\psi^*(x), \psi^*(y))$ be the edge in the derived *IFG* $C_\Delta(\Gamma, \Lambda)$ corresponding to edge (x, y) in the principal *IFG* $C_\Delta(\Gamma)$. Then, we define

$$TS(i, l, r) = \{\gamma : \gamma \in \gamma_i\Lambda, \text{ edge } (\psi^*(\gamma), \psi^*(\gamma\delta_r)) \text{ is of secondary color } l\}$$

With the above definition, we can describe processor scheduling as follows. Consider the processors executing at level r. Each processor P_i will execute the subtasks belonging to set $TS(i, 1, r)$ sequentially. Suppose that the two processors P_{i_1} and P_{i_2} are executing the subtasks belonging to $TS(i_1, 1, r)$ and $TS(i_2, 1, r)$, respectively, at level r. When communicating at level r, these two processors should not be allowed to use the same bus simultaneously. The bus processor P_{i_1} is using is the one assigned to the edge of primary color r and secondary color 1 originating from vertex v_{i_1}. Also, the bus processor P_{i_2} is using is the one assigned to the edge of primary color r and secondary color 1 originating from vertex v_{i_2}. Color partitioning guarantees that those two buses are distinct. Therefore, no bus conflicts will occur. This scheduling can be used for every level r for which there is a primary color. For levels, where there is no primary color (this happens when that color was masked by another primary color), follow the same

scheduling, all processors will perform computations concurrently and communications concurrently. There will be no bus conflicts. Furthermore, bus utilization will be uniform.

6.2 *MBS* with Given Number of Buses

If the optimal number of buses is large, it may be very costly and impractical. If the source algorithm requires more computation time and less communication time, then reduction of communication hardware at the cost of some speed will be well justified. For example, suppose that halving the number of buses results in 5 percent decrease in speed. Then it may be quite acceptable to reduce the number of buses by half at the cost of little extra time. Notice that, as shown in Section 6.1, reducing the number of processors automatically reduces the number of buses by the same factor. The purpose of this section is to analyze how to reduce the number of buses without altering the number of processors.

We will denote the specified number of buses by b. Also, we assume that $b < \beta(G)$. The construction of an *MBS* with b buses realizing a given *IFG* G is equivalent to the partition of the edge set of G into b subsets such that data transfers corresponding to the edges in each subset is carried out by a single bus. Since we assume that $b < \beta(G)$, a subset of the partition may contain more than one edge from the same color. Thus the partition does not correspond to a color partition (see Definition 3.4). According to our model, all data transfers associated with similar color edges of the *IFG* can be performed concurrently without affecting the integrity of the source algorithm. Therefore, when $b < \beta(G)$, some of the concurrent data transfers implied by the source algorithm must be carried out sequentially by the target *MBS*.

We have already proved that the problem of designing an optimal *MBS* with $\beta(G)$ buses to realize an arbitrary *IFG* G is NP-Hard. Therefore, the problem of designing an *MBS* with b number of buses is also NP-Hard. If one needs to solve the general problem, the heuristic algorithm given in Section 3.7 can be easily modified. As we have already mentioned, the general problem has no practical interest. Therefore, we do not attempt to solve the general color partition problem here. We assume that the *IFG* is vertex symmetric.

In Section 6.1, we showed that, in order to use processors efficiently, the number of processors used must be a factor of the optimal number of processors. Here, we will establish a similar result for the number of buses. When one processor is transferring data corresponding to color r, every processor is transferring data corresponding to color r. Suppose that the number of buses b in the target *MBS* is one less than the optimal number of buses $\beta(G)$. Then only b processors can simultaneously send data to their r-neighbors. The remaining processor, say P_1, must send its data to its r-neighbor after all other processors have finished their data transfers corresponding to the rth interconnection function. In order to preserve the integrity of the algorithm, while P_1 is transferring data to its r-neighbor, other processors cannot involve in communication transactions. Thus, during that period, $b - 1$ buses must stay idle. We could have used $\lceil \beta(G)/2 \rceil$ buses and obtained the same communication overhead. The following theorem generalizes this idea.

Theorem 6.8: If the specified number of buses is b, then only $\lceil \beta(G)/k \rceil$ buses can be used without speed penalty, where $k = \lceil \beta(G)/b \rceil$.

Proof: If we partition the edge set of $C_\Delta(\Gamma)$ into b subsets, at least one subset would contain at least $\lceil \beta(G)/b \rceil = k$ similar color edges. Therefore, to perform data transfers

corresponding to a single interconnection function, at least k communication steps are necessary. Therefore, the actual number of buses needed is the minimum number of buses required to perform a single interconnection function in k steps. The minimum number of disjoint subsets which can be formed without any subset exceeding k edges from a single color is $\lceil \beta(G)/k \rceil$. □

For example, suppose that $\beta(G) = 9$ and $b = 4$. Then $k = \lceil 9/4 \rceil = 3$. Therefore, $\lceil 9/3 \rceil = 3$ buses are actually needed. Communications which can be performed by 4 buses can also be performed by 3 buses without loss of speed; and fewer than 3 buses cannot perform the communication without loss of speed. If $\beta(G)/k$ is not an integer, all subsets in the partition will not have the same number of edges and the target *MBS* will be heterogeneous. We have already mentioned many advantages of regularity and symmetry of an *MBS*. Therefore, in order to maintain those regular properties of the target *MBS*, we always assume that b is a factor of $\beta(G)$.

By decreasing the number of buses by a factor of k, we reduce the cost of the buses by the same factor. Decreasing of the number of buses would have no effect on the cost of the processors. Note that, by decreasing the number of interfaces, the cost of a single processor would decrease due to the reduced number of ports. But we do not consider this effect in this research. Reduction of buses may automatically make some of the interfaces redundant. However, decreasing the number of buses by a certain factor will not necessarily decrease the number of interfaces also by the same factor. Our optimality criterion for partitioning the edge set of the *IFG* into b subsets would be to minimize the number of interfaces. In other words, we need to find a partition π of the edge set of

$C_\Delta(\Gamma)$ of cardinality b such that each subset contains k (= $|\Gamma|/b$) edges from each color and $|E(\pi(C_\Delta(\Gamma)))|$ is minimum.

Unlike the case for $b = \beta(C_\Delta(\Gamma)) = |\Gamma|$, there is no straightforward method to perform an optimal partition of $C_\Delta(\Gamma)$ for an arbitrary b. This rather unexpected nature of Cayley color graphs will be clarified using a very simple example.

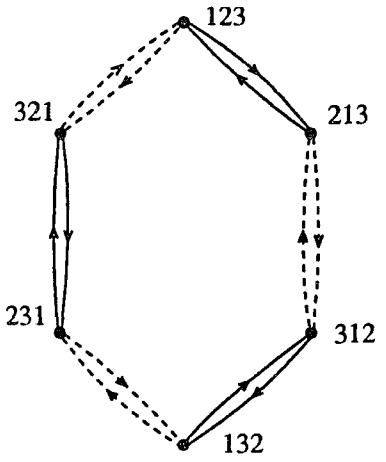


Figure 6.12: Principal *IFG* $C_{\Delta_{P(3)}}(\Gamma_{P(3)}\Lambda)$.

Example 6.4: Figure 6.12 shows the *IFG* $C_{\Delta_{P(3)}}(\Gamma_{P(3)})$ which is the Cayley color graph associated with the symmetric group $\Gamma_{P(3)}$ = {123, 213, 132, 312, 231, 321} and its generating set $\Delta_{P(3)}$ = {213, 321}. We will denote 213 and 321 by δ_1 and δ_2, respectively. Solid lines are used for generator δ_1 (color 1) and broken lines are used for generator δ_2 (color 2). If we replace bidirectional edges in Figure 6.12 with undirected edges and ignore colors, what we get is the pancake (or star) graph of three symbols. The optimal color partition of $C_{\Delta_{P(3)}}(\Gamma_{P(3)})$ contains six 2-element subsets such that for each subset E_i, $|J(E_i)|$ = 3. Now consider the problem of partitioning the edge set of $C_{\Delta_{P(3)}}(\Gamma_{P(3)})$ into three subsets instead of six. We need to find three edge disjoint subsets E_1, E_2, and E_3 such that $|J(E_1)| + |J(E_2)| + |J(E_3)|$ is minimum. One can be easily convinced that for any

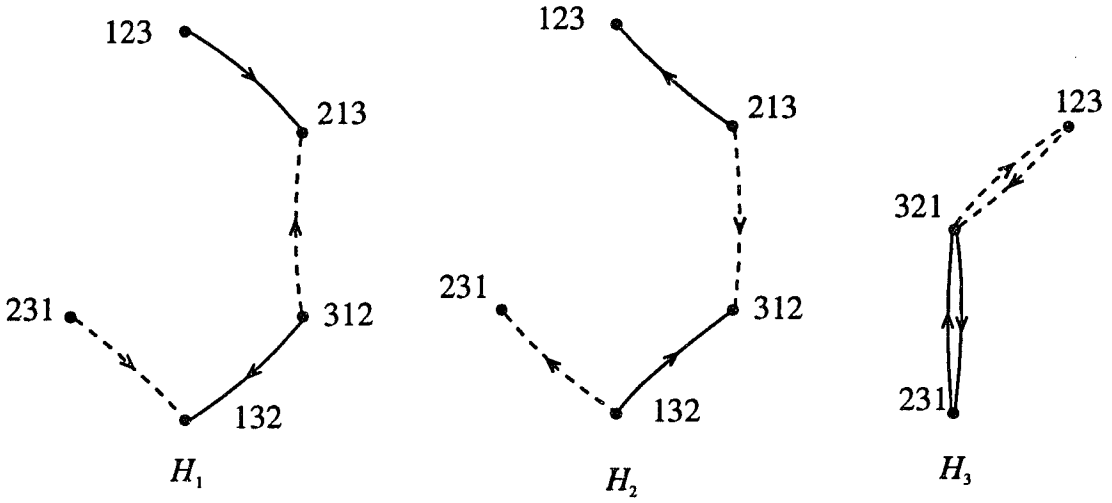


Figure 6.13: Optimal partition of $C_{\Delta_{P(3)}}(\Gamma_{P(3)}\Lambda)$ into three subsets.

subset E' of $E(C_{\Delta_{P(3)}}(\Gamma_{P(3)}))$ containing 2 edges from each color, the minimum value of $|J(E')|$ is 5. One can also be convinced that (possibly by exhaustive search) $E(C_{\Delta_{P(3)}}(\Gamma_{P(3)}))$ cannot be partitioned into three subsets E_1, E_2, and E_3 such that $|J(E_1)| = |J(E_2)| = |J(E_3)| = 5$. Figure 6.13 shows an optimal partition, where $|J(E_1)| = |J(E_2)| = 5$, and $|J(E_3)| = 6$. Therefore, the optimal partition of $C_{\Delta_{P(3)}}(\Gamma_{P(3)})$ results in three induced subgraphs which are not isomorphic to one another (recall that, for the optimal color partition $\phi_{\Gamma\Delta}$, induced subgraphs are isomorphic to one another). Figure 6.14 shows the *MBS* corresponding to the optimal partition shown in Figure 6.13. In the *MBS* shown in Figure 6.14, bus 1 is connected to 5 processors, whereas bus 3 is connected to only 3 processors. Furthermore, processor 123 has two drivers and two receivers, whereas, processor 321 has one driver and one receiver. Therefore, although the *IFG* is vertex symmetric, the optimal *MBS* is not even regular. This outcome is rather unexpected.

According to the above example, the optimal *MBS* with b buses corresponding to a vertex symmetric *IFG* can be heterogeneous. This undermines the whole purpose of

analyzing vertex symmetric *IFGs*. Besides, when the Cayley color graph is complex, it is unlikely to find an optimal partition using group properties. Therefore, we would seek to find the best partition under the requirement of symmetry (even at the expense of optimality). We will utilize the symmetric properties of $M(\Gamma, \Delta)$ to construct an *MBS* with b buses.

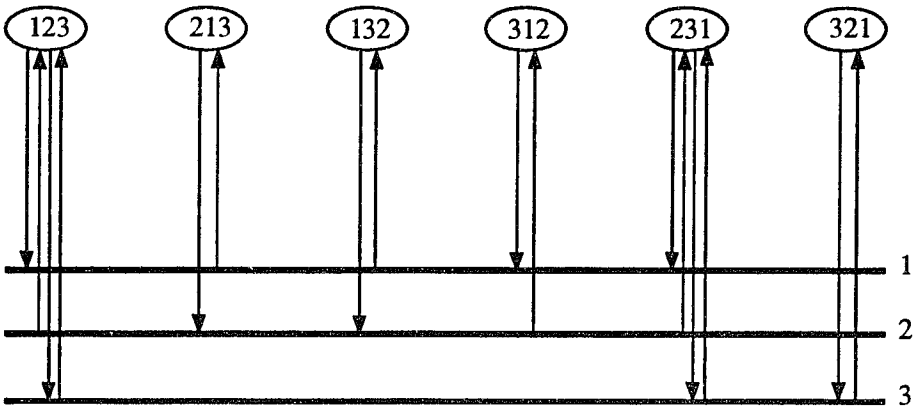


Figure 6.14: The *MBS* corresponding to the optimal partition in Figure 6.13.

Our approach is to combine the buses in $M(\Gamma, \Delta)$ in order to convert it into an *MBS* with b buses. This is equivalent to the partition of the buses in $M(\Gamma, \Delta)$ into k-element subsets. In doing so, we will reduce the number of buses from $|\Gamma|$ to b. Let processor P be connected to buses B_1 and B_2 via two drivers. If we combine the two buses together, only one driver is necessary. In other words, by combining the two buses B_1 and B_2, we save a driver.

As was done in Section 6.1, we will use subgroups of Γ to combine buses. Let Λ be a subgroup of Γ whose size is $|\Gamma|/b = k$. We will combine the buses of $M(\Gamma, \Delta)$ to form the *MBS* $M(\Gamma, \Lambda, \Delta)$ such that all buses of $M(\Gamma, \Delta)$ belonging to set $\{B_{ij} : \gamma_{ij} \in \gamma_i\Lambda\}$ are replaced by the single bus B_i in $M(\Gamma, \Lambda, \Delta)$. If there are more than one subgroup of size $|\Gamma|/b$, then the one which saves the maximum number of interfaces must be chosen.

In the following, we show how to determine the number of interfaces saved by the *MBS* $M(\Gamma, \Lambda, \Delta)$.

Let $E_1, E_2, ..., E_{|\Gamma|}$ be the subsets of edges produced by the optimal color partition $\phi_{\Gamma,\Delta}$ (see Definition 4.5). Also, let H_j be the subgraph induced by E_j, $1 \le j \le |\Gamma|$. Furthermore, let $\gamma_1\Lambda, \gamma_2\Lambda, ..., \gamma_b\Lambda$ be the distinct left cosets of Γ relative to Λ. Consider the set of edges $\bar{E}_i$ defined by, $\bar{E}_i = \bigcup_{\gamma_j \in \gamma_i\Lambda} E_i$. Clearly, $\bar{E}_i$ contains the edges of $C_\Delta(\Gamma)$ assigned to bus B_i. As we have observed in Section 4.2, the t^{th} vertex of H_j corresponds to a driver (receiver) if t is even (odd). Now, $\gamma_i\Lambda$ represents the set of 0^{th} vertices of the induced subgraphs H_j, $1 \le j \le |\Gamma|$. Also, $\gamma_i\Lambda\delta_1\delta_2^{-1}$ represents the set of 2^{nd} vertices of the induced subgraphs H_j, $1 \le j \le |\Gamma|$, and so on. Therefore, the number of drivers connected to bus B_i of $M(\Gamma, \Lambda, \Delta)$ is equal to the cardinality of set D_i, where

$$D_i = (\gamma_i\Lambda) \cup (\gamma_i\Lambda\delta_1\delta_2^{-1}) \cup (\gamma_i\Lambda\delta_1\delta_2^{-1}\delta_3\delta_4^{-1}) \cup ... \cup (\gamma_i\Lambda\delta_1\delta_2^{-1}...\delta_{2\lfloor|\Delta|/2\rfloor-1}\delta_{2\lfloor|\Delta|/2\rfloor})$$

Similarly, the number of receivers connected to bus B_i is equal to the cardinality of set R_i, where

$$R_i = (\gamma_i\Lambda\delta_1) \cup (\gamma_i\Lambda\delta_1\delta_2^{-1}\delta_3) \cup ... \cup (\gamma_i\Lambda\delta_1\delta_2^{-1}\delta_3...\delta_{2\lfloor(|\Delta|-1)/2\rfloor}\delta_{2\lfloor(|\Delta|-1)/2\rfloor+1})$$

Thus, the total number of interfaces in the *MBS* $M(\Gamma, \Lambda, \Delta)$ is

$$\sum_{i=1}^{b} (|D_i| + |R_i|) = b(|D_i| + |R_i|).$$

With the help of Lemma 6.1, it is easy to verify that the *MBS* $M(\Gamma, \Lambda, \Delta)$ is symmetric. Thus, we have constructed a symmetric *MBS* with $b = |\Gamma|/|\Lambda|$ buses and $|\Gamma|$ processors. That *MBS* has $b(|D_i| + |R_i|) - |\Gamma|(|\Delta| + 1)$ fewer interfaces compared with the optimal *MBS* $M(\Gamma, \Delta)$. The following example illustrates the analysis done in this section.

Example 6.5: We will revisit the *IFG* $C_{\Delta_{P(3)}}(\Gamma_{P(3)})$ shown in Figure 6.12. Figure 6.15 shows the six induced subgraphs of the optimal color partition $\phi_{\Gamma_{P(3)},\Delta_{P(3)}}$. Let Λ =

{123, 213}, that is, $\Lambda = \{e, \delta_1\}$. Induced subgraphs of the three subsets $\bar{E}_1$, $\bar{E}_2$, and $\bar{E}_3$, denoted by $\bar{H}_1$, $\bar{H}_2$, and $\bar{H}_3$, respectively, are shown in Figure 6.16. Notice that, $\bar{E}_1 = E_1 \cup E_2$, $\bar{E}_2 = E_3 \cup E_4$, and $\bar{E}_3 = E_5 \cup E_6$. Isomorphism of the induced subgraphs can be easily observed. Figure 6.17 shows the corresponding *MBS* $M(\Gamma, \Lambda, \Delta)$. Clearly, it has 18 interfaces (12 drivers and 6 receivers), in contrast to the 16 interfaces required by the optimal *MBS* shown in Figure 6.14. But as a compensation, we have symmetry. Notice that bus loading and the maximum number of ports per processor are also better in the system of Figure 6.17.

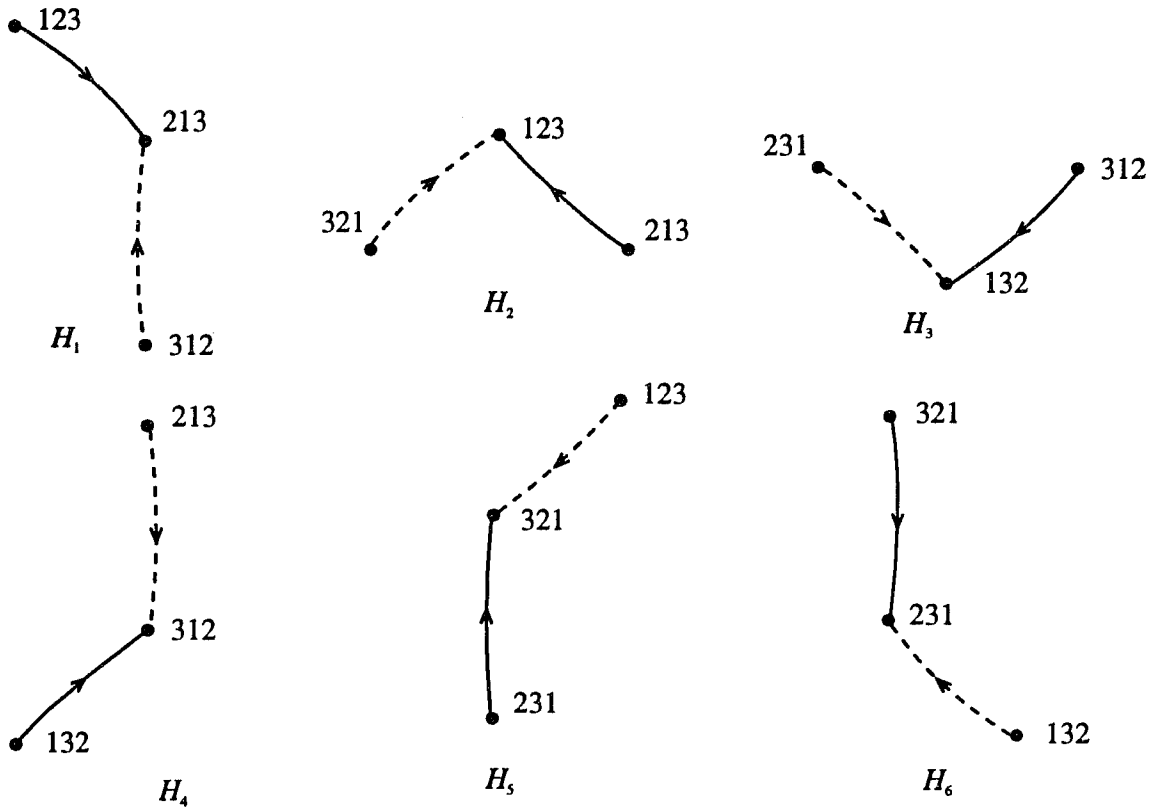


Figure 6.15: Optimal color partition of the *IFG* in Figure 6.12.

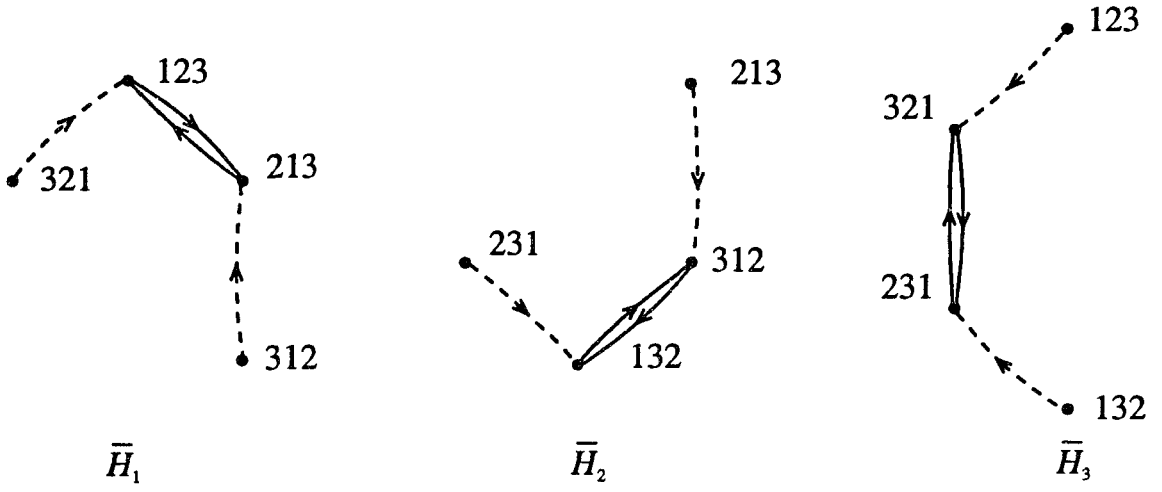


Figure 6.16: Induced subgraphs obtained by combining those in Figure 6.15.

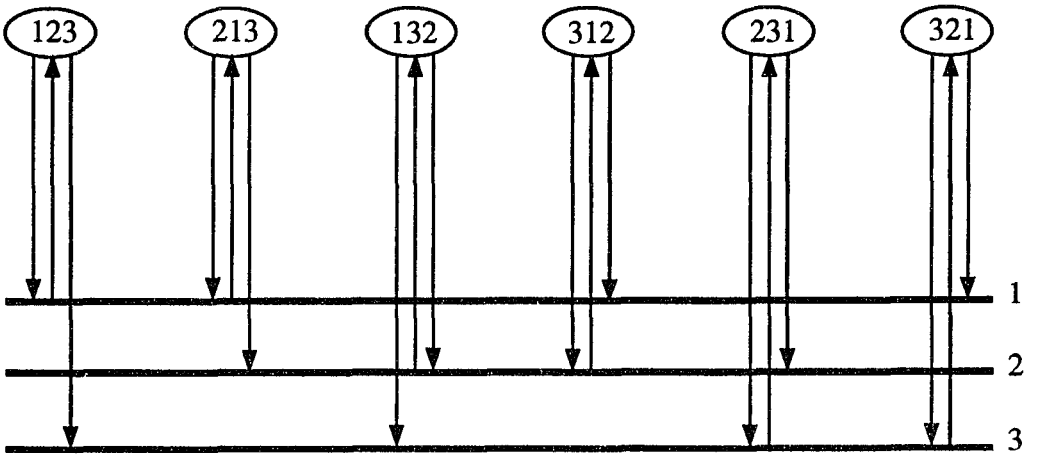


Figure 6.17: The *MBS* corresponding to the induced subgraphs in Figure 6.16.

CHAPTER 7

SUMMARY AND CONCLUSIONS

Applicability of multiple bus systems as special purpose architectures is relatively unexplored in the literature. In this dissertation, we developed a methodology to design multiple bus architecture which favors a given class of parallel algorithms. The algorithmic class we considered here reflects the communication pattern of the member algorithms. We represented the source algorithm (or the algorithmic class) as a labeled, edge colored, directed graph called *CFG*. The target *MBS* was represented as a directed bipartite graph. According to the model we assumed, the target *MBS* operated in the *SIMD* mode and used message passing model for interprocessor communication.

We performed the construction of an optimal *MBS* from the given *CFG* in two stages. In the first stage, which was addressed in Chapter 2, from the given *CFG*, we constructed another graph *IFG*. The *IFG* reflects the communication pattern among processors in the target architecture that can efficiently execute any algorithm belonging to the given class. We proved that the construction of an optimal *IFG* from the given *CFG* is an *NP*-Hard problem. Nevertheless, we showed that certain regularities present in almost every, well-behaved, parallel algorithm can be exploited to obtain a polynomial time solution. We also showed that when the *CFG* is regular, the resulting *IFG* is also regular.

In the second stage, starting from the *IFG* constructed in the first stage (or, from any other arbitrary *IFG*), we constructed an optimal *MBS*. The construction of an *MBS* optimally realizing a given *IFG* naturally answers two other important issues relating to *MBS*s. The first is the design of an optimal *MBS* emulating an existing, *SIMD*, static

interconnection network. The second is the design of an optimal *MBS* realizing a given set of interconnection functions. Due to this wide applicability, we devoted a major part of the dissertation to the construction of an optimal *MBS* realizing a given *IFG*. We showed that the design of an *MBS* from a given *IFG* is equivalent to the partition of the edge set of the *IFG* with certain constraints. This is called color partition.

In Chapter 3, we proved that the optimal color partition of an arbitrary *IFG* is an *NP*-Hard problem. We also proved that the optimal color partition problem is solvable in polynomial time when the *IFG* has only two colors. Therefore, an optimal *MBS* realizing two interconnection functions (such as, shuffle and exchange) can be constructed in polynomial time. Taking the solution method to the two color case as a guideline, we developed a heuristic algorithm for solving the general color partition problem in polynomial time.

The analysis in Chapter 3 naturally opens some avenues for future research work. We did not consider other features (except when the *IFG* is vertex symmetric or regular) of an *IFG* which will guarantee a polynomial time solution. For example, when the *IFG* is a tree, the problem may be solvable in polynomial time. Thus, the investigation of attractive properties of algorithms which can be used to solve the color partition problem in polynomial time is a possible extension to this work. The heuristic algorithm we presented was obviously not the best to solve the general color partition problem. The development of a better algorithm would be another extension to this work. It is an interesting and challenging problem to develop an algorithm which, given an arbitrary constant ϵ, solves the general color partition problem whose output is at most $(1 + \epsilon)opt$, where *opt* is the optimal output.

In Chapter 4, we showed how to obtain an optimal color partition of a vertex symmetric *IFG*. This was done by using the analogy between such an *IFG* and a Cayley color graph associated with a finite group Γ and its generating set Δ. We showed that there can be many optimal color partitions; however, we have chosen a particular color partition which results in an *MBS*, denoted by $M(\Gamma, \Delta)$, having many attractive properties. We proved that $M(\Gamma, \Delta)$ is symmetric. We also showed the superiority of $M(\Gamma, \Delta)$ over its static interconnection network counterpart in terms of the number of ports per processor, the number of neighbors per processor, and the diameter. As an automatic outcome of the optimal color partition of a vertex symmetric *IFG*, we showed how to find an optimal color partition for a regular *IFG*. We also presented an algorithm to perform such a partition.

Again, there are several possible avenues for future research work related to the treatise on Chapter 4. The optimality of the color partition obtained for a vertex symmetric or regular *IFG* was guaranteed only if the *IFG* belongs to a certain class; otherwise, the optimality was not guaranteed. Fortunately, *IFGs* belonging to that particular class encompass interconnection patterns of many well known algorithms as well as interconnection functions of many known *SIMD* machines. However, it may be worthwhile to explore the possibility of finding an optimal color partition for every vertex symmetric or regular *IFG*. By considering some of the unsolvable problems in group theory, it is unlikely that one would come up with a polynomial time solution for finding an optimal color partition for an arbitrary Cayley color graph. Therefore, an appropriate attempt would be for a heuristic algorithm.

In Chapter 5, we addressed the fault tolerance capabilities of the *MBS* $M(\Gamma, \Delta)$. We proved that $M(\Gamma, \Delta)$ can sustain a single driver failure or a single processor failure if and only if $|\Delta| > 1$. We also proved that $M(\Gamma, \Delta)$ can sustain a single receiver failure or single bus failure if and only if $|\Delta| > 2$. We also obtained the performance degradation due to each component failure when the *IFG* represents cube functions. Furthermore, we showed how to add redundancy to $M(\Gamma, \Delta)$ in order to increase its fault tolerance. Even though we only analyzed single component failures in Chapter 5, it can be easily extended to multiple component failures. The number of edge disjoint paths and vertex disjoint paths in Cayley color graphs can be utilized to analyze the fault tolerance of $M(\Gamma, \Delta)$ under multiple component failures. This will be a useful extension to the work done in Chapter 5.

In Chapter 6, we addressed the problem of constructing an *MBS* with a given number of processors and/or a given number of buses which can realize a given *IFG* optimally. We analyzed the cases with fixed number of processors and fixed number of buses separately. We showed that the number of processors (buses) must be a factor of the optimal number of processors (buses) in order to utilize processors (buses) efficiently. We used the properties of cosets of Γ in order to construct an *MBS* with given number of buses and/or processors. When the number of processors was specified, by partitioning the vertex set of the original *IFG*, we obtained a new *IFG* that has as many vertices as the number of processors in the target *MBS*. We showed how the partition can be done so that the derived *IFG* is regular. We also provided processor scheduling which guarantees that the source algorithm would run on the target *MBS* at maximum possible

speed. When the number of buses is specified, we showed how to combine buses in order to obtain the target *MBS*. Again, we use cosets of Γ to combine the buses in $M(\Gamma, \Delta)$.

In constructing an optimal *MBS* with given number of processors and/or buses, there are certain issues we did not address in Chapter 6. Even though the given *IFG* is vertex symmetric, the optimal *MBS* with given number of processors and/or buses may not be even regular. Our focus in the chapter was on constructing an *MBS* which is at least regular. We did not address the problem of finding an optimal *MBS* without paying attention to its structure. This could be an extension to the work reported in Chapter 6. Again, due to the computational difficulty of the general problem and due to the fact that there are certain unsolvable problems regarding groups and their generating sets, it is unlikely that we can find a polynomial time algorithm to construct an optimal *MBS* with a specified number of processors and/or buses. Therefore, one may have to be content with a heuristic algorithm.

REFERENCES

[1] G. B. Adams and H. J. Siegel, "The Extra-Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems", *IEEE Trans. on Computers*, vol. c-31, May 1982, pp. 443-454.

[2] A. Agarwal, "Limits on Interconnection Network Performance", *IEEE Trans. on Parallel & Distributed Systems*, vol. 2, no. 4, October 1991, pp. 398-412.

[3] A. Aggarwal, "Optimal Bounds for Finding Maximum on Array of Processors with *k* Global Buses", *IEEE Trans. on Computers*, vol. c-35, no. 1, January 1986, pp. 62-64.

[4] D. P. Agrawal, "Testing and Fault Tolerance of Multistage Interconnection Networks", *Computer*, April 1982, pp. 41-53.

[5] S. B. Akers and B. Krishnamurthy, "A Group-Theoretic Model for Symmetric Interconnection Networks", *IEEE Trans. on Computers*, vol. 38, no. 4, April 1989, pp. 555-566.

[6] B. Alspach, "Cayley Graphs with Optimal Fault Tolerance", *IEEE Trans. on Computers*, vol. 41, no. 10, October 1992, pp. 1337-1340.

[7] T. Anderson and P. A. Lee, *Fault Tolerance: Principles and Practice*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[8] J. Archibald and J. Baer, "An Evaluation of Cache Coherence Solutions in Shared-Bus Multiprocessors", *ACM Transactions on Computer Systems*, 4, 4 (November 1986), pp. 273-298.

[9] R. Arlauskas, "*iPSC/2* System: A Second Generation Hypercube", *Third Conference on Hypercube Concurrent Computers and Applications, ACM* 1988, pp. 33-36.

[10] E. R. Barnes, "Partitioning the Nodes of a Graph", *Graph Theory with Applications to Algorithms and Computer Science*, Editors: Y. Alavi et al., John Wiley & Sons, pp. 57-72.

[11] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The *ILLIAC IV* Computer", *IEEE Trans. on Computers*, vol. 17, no. 8, August 1968, pp. 746-757.

[12] K. E. Batcher, "The Flip Network in STARAN", *Proc. of the International Conference on Parallel Processing*, August 1976, pp. 65-71.

[13] K. Batcher, "Design of a Massively Parallel Processor", *IEEE Trans. on Computers*, vol. 29, no. 9, September 1980, pp. 836-840.

[14] J. Beetem, M. Denneau, and D. Weingarten, "The *GF*11 Supercomputer", *Proc. of the 12th International Symposium on Computer Architecture*, Boston, 1985, pp. 108-115.

[15] V. E. Benes, *Mathematical Theory of Communication Networks and Telephone Traffic*, Academic Press, New York, 1965.

[16] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon, "The Midway Distributed Shared Memory Systems", *Proceedings of the 1993 CompCon Conference*, February 1993, pp. 528-537.

[17] M. J. Berger and S. H. Bokhari, "A Partitioning Strategy for Nonuniform Problems on Multiprocessors", *IEEE Trans. on Computers*, vol. c-36, no. 5, May 1987, pp. 570-580.

[18] L. N. Bhuyan, "A Combinatorial Analysis of Multibus Multiprocessor Systems", *Proceedings of the 1984 International Conference in Parallel Processing*, 1984, pp. 225-227

[19] L. N. Bhuyan and D. P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network", *IEEE Trans. on Computers* vol. c-33, no. 4, April 1984, pp. 323-333.

[20] L. N. Bhuyan, Q. Yang, and D. P. Agrawal, "Performance of Multiprocessor Interconnection Networks", *IEEE Computer*, February 1989, pp. 25-37.

[21] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, and E. W. Felten, "Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, April 1994, pp. 142-153

[22] S. H. Bokhari, "Finding Maximum on an Array Processor with a Global Bus", *IEEE Trans. on Computers*, vol. c-33, no. 2, February 1984, pp. 133-139.

[23] S. H. Bokhari, "On the Mapping Problem", *IEEE Trans. on Computers*, vol. c-30, no. 3, March 1981, pp. 207-344.

[24] S. H. Bokhari, "Partitioning Problems in Parallel, Pipelined, and Distributed Computing", *IEEE Trans. on Computers*, vol. 37, no. 1, January 1988, pp. 48-57.

[25] B. L. Bondar and A. C. Liu, "Modelling and Performance Analysis of Single Bus Tightly Coupled Multiprocessors", *IEEE Trans. on Computers*, vol. 38, no. 3, March 1989, pp. 464-470.

[26] R. E. Buehrer et al. "The *ETH*-Multiprocessor *EMPRESS*: A Dynamically Configurable *MIMD* System", *IEEE Trans. on Computers*, vol. c-31, no. 11, November 1982, pp. 1035-1044.

[27] L. Campbell, G. E. Carlson, M. J. Dinneen, V. Faber, M. R. Fellows, M. A. Langston, J. W. Moore, A. P. Mullhaupt and H. B. Sexton, "Small Diameter Symmetric Networks from Linear Groups", *IEEE Trans. on Computers*, vol. 41, No. 2, February 1992, pp. 218-220.

[28] M. J. Carey and C. D. Thompson, "A Pipelined Architecture for Search Tree Maintenance", *Algorithmically Specialized Parallel Computers*, Academic Press, Inc., 1985, pp. 37-46.

[29] D. A. Carlson, "Performing Tree and Prefix Computations on Modified Mesh-Connected Parallel Computers", *Proceedings of the International Conference on Parallel Processing*, 1985, pp. 715-718.

[30] G. Chartrand and L. Lesniak, *Graphs and Digraphs*, Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California, 1986.

[31] V. Chaudhary and J. K. Aggarwal, "A Generalized Scheme for Mapping Parallel Algorithms", *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 3, March 1993, pp. 328-346.

[32] M. S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, Routing, and Broadcasting in Hexagonal Mesh Multiprocessors", *IEEE Trans. on Computers*, vol. 39, no. 1, January 1990, pp. 10-18.

[33] W. T. Chen and J. P. Sheu, "Performance Analysis of Multiple Bus Interconnection Networks with Hierarchical Requesting Model", *IEEE Trans. on Computers*, vol. 40, no. 7, July 1991, pp. 834-842.

[34] D. M. Chiarulli, S. P. Levitan, and R. Melhem, "Optical Bus Control for Distributed Multiprocessors", *Journal of Parallel and Distributed Computing* **10**, 1990, pp. 45-54.

[35] A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, and W. Zwaenepoel, "Software versus Hardware Shared-Memory Implementations: A Case Study", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, April 1994, pp. 106-117

[36] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken, and T. Blackadar, "Performance Measurements on a 128-node Butterfly Parallel Processor", *Proceedings of the International Conference on Parallel Processing*, August 1985, pp. 531-540.

[37] W. J. Dally, "Performance Analysis of *k*-ary *n*-cube Interconnection Networks", *IEEE Trans. on Computers*, vol. 39. no. 6, June 1990, pp. 775-785.

[38] W. J. Dally et al., "The *J*-Machine: A Fine-Grain Concurrent Computer", *Information Processing* 89, Elsevier North Holland, 1989.

[39] R. F. Demara and D. I. Moldovan, "The *SNAP*-1 Parallel *AI* Prototype", *Proceedings of the 18th Annual International Symposium on Computer Architecture*", May 1991, pp. 2-11.

[40] A. M. Despain and D. A. Patterson, "X-Tree: A Tree Structured Multiprocessor Computer Architecture", *Proc. 5th Annual Symp. on Computer Architecture*, 1978, pp. 144-151.

[41] O. M. Dighe, R. Vaidyanathan, and S. Q. Zheng, "Bus-Based Tree Structures for Efficient Parallel Computation" *Proceedings of the International Conf. on Parallel Processing*, August 1993, pp. *I*-158 - *I*-161.

[42] M. Dubois, "Throughput Analysis of Cache-Based Multiprocessors with Multiple Buses", *IEEE Trans. on Computers*, vol. 37, no. 1, January 1988, pp. 58-70.

[43] R. Duncan, "A survey of Parallel Computer Architectures", *IEEE Computer*, February 1990, pp. 5-16.

[44] A. El-Amawy and Priyalal Kulasinghe, "Optimal Mapping of Feedforward Neural Networks onto Multiple Bus Architectures", *IEEE 37th Midwest Symposium on Circuits and Systems* 1994, pp 477-483.

[45] R. D. Etchells and G. R. Nudd, "Software Metrics for Performance Analysis of Parallel Hardware", *DARPA Image Processing Workshop*, June 1983, pp. 137-147.

[46] T. Y. Feng, "A Survey of Interconnection Networks", *IEEE Computer* **14**, December 1981, pp. 12-27.

[47] T. Feng, "Data Manipulating Functions in Parallel Processors and Their Implementations", *IEEE Trans. on Computers*, vol. c-23, no. 3, March 1974, pp. 309-318.

[48] T. Y. Feng, "Some Characteristics of Associative/Parallel Processing", *Proc. 1972 Sagamore Computer Conference*, Syracuse University, 1972, pp. 5-16.

[49] C. M. Fiduccia, "Bused Hypercubes and Other Pin-Optimal Networks", *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 1, January 1992, pp. 14-24.

[50] M. J. Flynn, "Some Computer Organizations and Their Effectiveness", *IEEE Trans. on Computers*, vol c-21, 1972, pp. 948-960.

[51] M. I. Frank, "A Hybrid Shared Memory / Message Passing Parallel Machine", *Proceedings of the International Conference on Parallel Processing*, August 1993, pp. *I*-232 - *I*-236.

[52] D. D. Gajski, "Does General Purpose Mean Good for Nothing", *Algorithmically Specialized Parallel Computers*, Academic Press, Inc., 1985, pp. 249-250.

[53] M. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[54] W. H. Gates and C. H. Papadimitriou, "Bounds for Sorting by Prefix Reversal", *Discrete Mathematics* 27, 1979, pp. 47-57.

[55] G. R. Goke and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessor Systems", *Proc. First Annual Symp. on Computer Architecture*, December 1973, pp. 21-28.

[56] M. G. and M. Minoux (translated by S. Vajda), *Graphs and Algorithms*, John Wiley and Sons, 1984.

[57] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The *NYU* Ultracomputer - Designing an *MIMD* Shared Memory Parallel Computer", *IEEE Trans. on Computers*, vol. c-32, no. 2, February 1983, pp. 175-189.

[58] Paul E. Green, "The Future of Fibre-Optic Computer Networks", *IEEE Computer*, September 1991, pp. 78-87.

[59] W. Handler, "The Impact of Classification Schemes on Computer Architecture" *Proc. International Conference on Parallel Processing, IEEE*, 1977, pp. 7-15.

[60] P. Hansen and K. W. Lih, "Improved Algorithms for Partitioning Problems in Parallel, Pipelined, and Distributed Computing", *IEEE Trans. on Computers*, vol. 41, no. 6, June 1992, pp. 769-771.

[61] J. P. Hayes, T. N. Mudge, Q. F. Stout, S. Colley, and J. Palmer, "Architecture of a Hypercube Supercomputer", *Proceedings of the 1986 International Conference on Parallel Processing*, 1986, pp. 653-660.

[62] W. D. Hillis, "*The Connection Machine*, *MIT* Press, 1985.

[63] M. D. Hill et al., "*SPUR*: A *VLSI* Multiprocessor Workstation", *IEEE Computer*, vol. 19, November 1986, pp. 8-22.

[64] R. W. Hockney and C. R. Jesshope, *Parallel Computers*, Adam Hilger Ltd., 1981.

[65] E. Hokens and A. Louri, "Performance Considerations Relating to the Design of Interconnection Networks for Multiprocessing Systems", *Proceedings of the 1993 International Conference on Parallel Processing*, August 1993, pp. *I*-206 - *I*-209.

[66] M. A. Holloday and M. K. Vernon, "Exact Performance Estimates of Multiprocessor Memory and Bus Interfaces", *IEEE Trans. on Computers*, vol. c-36, no. 1, January 1987, pp. 76-85.

[67] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM J. Computing* 2, 1973, pp. 225-231.

[68] A. Hopper, A. Jones, and D. Liopis, "Multiple vs. Wide Shared Bus Multiprocessors", *16th Annual Symposium on Computer Architecture*, 1989, pp. 300-306.

[69] E. Horowitz and A. Zorat, "The Binary Tree as an Interconnection Network: Applications to Multiprocessor Systems and *VLSI*", *IEEE Trans. on Computers*, vol. c-30, no. 4, April 1981, pp. 247-253.

[70] K. Hwang, "Advanced Parallel Processing with Supercomputer Architectures", *Proceedings of the IEEE*, vol. 75, no. 10, October 1987, pp. 1348-1379.

[71] K. Hwang, P. Sheng, and D. Kim, "An Orthogonal Multiprocessors for Parallel Scientific Computations", *IEEE Trans. on Computers*, vol. 38, no. 1, January 1989, pp. 47-60.

[72] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1977, pp. 32-40.

[73] L. H. Jamiesson, "Characterizing Parallel Algorithms", *The Characteristics of Parallel Algorithms*, The *MIT* Press, 1987, pp. 64-100.

[74] H. Jiang and K. C. Smith, "A Partial-Multiple-Bus Computer Structure with Improved Cost-Effectiveness", *Proceedings of the 15th Annual Int. Symp. on Computer Architecture*, 1988, pp. 116-122.

[75] S. I. Kartashev and S. P. Kartashev, "A Multicomputer System with Dynamic Architecture", *IEEE Trans. on Computers*, vol. c-28, October 1979, pp. 704-720.

[76] H. Kasahara and S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing", *IEEE Trans. on Computers*, vol c-33, no. 11, November 1984, pp. 1023-1029.

[77] J. Killian, S. Kipnis, and C. E. Leiserson, "The Organization of Permutation Architectures with Bused Interconnections", *IEEE Trans. on Computers*, vol. 39, no. 11, November 1990, pp. 1346-1357.

[78] A. C. Klaiber and H. M. Levy, "A Comparison of Message Passing and Shared Memory Architectures for Data Parallel Programs", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, April 1994, pp. 94-105.

[79] S. C. Kothari and E. Gannett, "Optimal Design of Linear Flow Systolic Architectures", *Proceedings of the International Conference on Parallel Processing*, August 1989, pp. *I*-247 - *I*-256.

[80] J. S. Kowalik (ed.), *Parallel MIMD Computation: HEP Supercomputer and its Applications*, MIT Press, Cambridge, MA, 1985.

[81] R. Krishnamurti and E. Ma, "The Processor Partitioning Problem in Special Purpose Partitionable Systems", *Proceedings of the Int. Conf. on Parallel Processing*, 1988, pp. 434-443.

[82] D. J. Kuck, E. S. Davidson, D. H. Lawrie, and A. H. Sameh, "Parallel Supercomputing Today and the Cedar Approach", *Science* **231**, February 1986, pp. 967-974.

[83] D. P. Kulasinghe and A. El-Amawy, "On the Complexity of Optimal Bused Interconnections", *IEEE Trans. on Computers*, to appear.

[84] D. P. Kulasinghe and A. El-Amawy, "Optimal Realization of Sets of Interconnection Functions on Multiple Bus Systems" Submitted to the *IEEE Trans. on Computers* for second revision.

[85] H. T. Kung, "Why Systolic Architecture", *IEEE Computer*, January 1982, pp. 37-46.

[86] S. Y. Kung, "*VLSI* Array Processors", *IEEE ASSP Magazine*, vol. 2, no. 3, July 1985, pp. 4-22.

[87] J. Kuskin et al., "The Stanford *FLASH* Multiprocessor", *Proceedings of the 21st Annual International Symposium on Computer Architecture*, April 1994, pp. 302-313.

[88] T. Lang and M. Valero, "M-users B-servers Arbiter for Multiple-Buses Multiprocessors", *Microprocessing and Microprogramming*, 1982, pp. 11-18.

[89] T. Lang, M. Valero, and I. Alledre, "Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors", *IEEE Trans. on Computers*, vol. c-31, no. 12, December 1982, pp. 1227-1234.

[90] T. Lang, M. Valero, and M. A. Fiol, "Reduction of Connections for Multibus Organization", *IEEE Trans. on Computers*, vol. c-32, no. 8, August 1983, pp. 707-715.

[91] S. Latifi and A. El-Amawy, "On Folded Hypercubes", *Proceedings of the 1989 International Conference on Parallel Processing*, August 1989, pp. *I*-180 - *I*-187.

[92] D. H. Lawrie, "Access and Alignment of Data in an Array Processor", *IEEE Trans. on Computers*, vol. c-24, no. 12, December 1975, pp. 1145-1155.

[93] T. J. LeBlanc, "Problem Decomposition and Communication Tradeoffs in a Shared Memory Multiprocessor", *Numerical Algorithms for Modern Parallel Computer Architectures, IMA Volumes in Mathematics and its Applications*, vol. 13, Springer-Verlag 1988, pp. 145-163

[94] T. J. LeBlanc, "Shared Memory Versus Message-Passing in a Tightly-Coupled Multiprocessor: A Case Study", *Proceedings of the 1986 International Conference on Parallel Processing*, August 1986, pp. 463-466.

[95] I. Lee and D. Smitley, "A Synthesis Algorithm for Reconfigurable Interconnection Networks", *IEEE Trans. on Computers*, vol. 37, no. 6, June 1988, pp. 691-699.

[96] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Goopta, and J. Hennessy, "The DASH Prototype: Implementation and Performance", *19th Annual International Symposium on Computer Architecture*, May 1992, pp. 92-103.

[97] K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems", *ACM Transactions on Computer Systems*, vol. 7, no. 4, November 1989, pp. 321-359.

[98] K. Li and R. Schaefer, "A Hypercube Shared Virtual Memory System", *Proceedings of the International Conference on Parallel Processing*, August 1989, pp. *I*-125 - *I*-132.

[99] C. Lin and L. Snyder, "A Comparison of Programming Models for Shared Memory Multiprocessors", *Proceedings of the 1990 International Conference on Parallel Processing*, August 1990, pp. 163-170.

[100] R. Lin and S. Olariu, "Application-Specific Array Processors for Binary Prefix Sum Computation", *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, October 1994, pp. 118-125.

[101] T. Lovett and S. Thakkar, "The Symmetry Multiprocessor System", *Proceedings of the International Conference on Parallel Processing*, 1988, pp. 303-311.

[102] W. Magnus, A. Karrass D. Solitar, *Combinatorial Group Theory*, Dover Publications, Inc, New York, 1976.

[103] Y. Mansour and L. Schulman, "Sorting on a Ring of Processors", *Journal of Algorithms*, vol. 11, no. 4, December 1990, pp. 622-630.

[104] M. A. Marson et. al, "Modeling Bus Contention and Memory Interference in a Multiprocessor System", *IEEE Trans. on Computers*, January 1983, pp. 60-72.

[105] M. Martonosi and A. Gupta, "Tradeoffs in Message-Passing and Shared-Memory Implementations of a Standard Cell Router", *Proceedings of the 1989 International Conference on Parallel Processing*, August 1989, pp. *III*-88 - *III*-96.

[106] A. Mathialagan and N. N. Biswas, "Optimal Interconnections in the Design of Microprocessor and Digital Systems", *IEEE Trans. on Computers*, vol. c-29, no. 2, February 1980, pp. 145-148.

[107] C. A. Mead and L. A. Convey, *Introduction to VLSI Systems*, Reading MA, Addison-Wesley, 1980.

[108] M. D. Mickunas, "Using Projective Geometry to Design Bus Connection Networks", *Proc. Workshop Interconnection Networks for Parallel and Distributed Processing, ACM/IEEE*, April 1990, pp. 47-55.

[109] T. N. Mudge, J. P. Hayes, and D. C. Winsor, "Multiple Bus Architectures", *IEEE Computer*, June 1987, pp. 42-48.

[110] P. A. Nelson and L. Snyder, "Programming Paradigms for Nonshared Memory Parallel Computers", *The Characteristics of Parallel Algorithms*, The *MIT* Press, 1987, pp. 3-20.

[111] A. Osterhaug, *Guide to Parallel Programming*, Sequent Computer Systems Inc., 1986.

[112] R. C. Pearce, J. A. Field, and W. D. Little, "Asynchronous Arbiter Module", *IEEE Trans. on Computers*, September 1975, pp. 931-932.

[113] C. C. Pinter, *A Book of Abstract Algebra*, McGraw-Hill Publishing Company, 1990.

[114] D. P. Pradhan, Fault-Tolerant Multiprocessor Link and Bus Network Architectures", *IEEE Trans. on Computers*, vol. 34, no. 1, January 1985, pp. 33-45.

[115] F. P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation", *Computer Architecture and Systems*, vol. 24, no. 5, May 1981, pp. 300-309.

[116] U. Ramachandran, G. Shah, and S. Ravikumar, "Scalability Study of the *KSR-1*", *Proceedings of the 1993 International Conference on Parallel Processing*, August 1993, pp. *I*-237 - *I*-240.

[117] D. A. Rennels, "Distributed Fault-Tolerant Computer Systems", *Computer*, vol. 13, no. 3, March 1980, pp. 55-65.

[118] C. D. Rose, "Encore Eyes Multiprocessor Market", *Electronics*, July 8, 1985.

[119] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes", *IEEE Trans. on Computers*, vol. 37, no. 7, July 1988, pp. 867-872.

[120] P. Sadayappan and F. Ercal, "Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes", *IEEE Trans. on Computers*, vol. c-36, no. 12, December 1987, pp. 1408-1424.

[121] M. R. Samatham, and D. K. Pradhan, "The De Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for *VLSI*", *IEEE Trans. on Computers*, vol. 38, no. 4, April 1989, pp. 567-581.

[122] C. L. Seitz, "The Cosmic Cube", *Communications of the ACM*, vol. 28, no. 1, January 1985, pp. 22-33.

[123] W. Shang and J. A. B. Fortes, "Independent Partitioning of Algorithms with Uniform Dependence", *IEEE Trans. on Computers*, vol. 41, no. 2, February 1992, pp. 190-206.

[124] W. Shang and B. W. Wah, "Dependence Analysis and Architecture Design for Bit-Level Algorithms", *Proceedings of the 1993 International Conference on Parallel Processing*, August 1993, pp. *I*-30 - *I*-38.

[125] H. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington MA, 1984.

[126] H. J. Siegel and J. T. Kuehn, "*PASM*: A Partitionable *SIMD/MIMD* System for Parallel Image Processing Research", *Algorithmically Specialized Parallel Computers*, Academic Press, Inc., 1985, pp. 69-78.

[127] H. J. Siegel, R. J. McMillen, and P. T. Mueller, "A Survey of Interconnection Methods for Reconfigurable Parallel processing systems", *National Computer Conference*, 1979, pp. 529-541.

[128] H. J. Siegel and S. D. Smith, "Study of Multistage *SIMD* interconnection Networks", *Proc. of 5th Annual Symp. on Computer Architecture*, April 1978, pp. 223-229.

[129] D. B. Skillicorn, "A New Class of Fault-Tolerant Static Interconnection Networks", *IEEE Trans. on Computers*, vol. 37, no. 11, November 1988, pp. 1468-1470.

[130] L. Snyder, "Introduction to the Configurable, Highly Parallel Computer", *IEEE Computer* **15**, January 1982, pp. 47-56.

[131] H. S. Stone, *Discrete Mathematical Structures and Their Applications*, Science Research Associates, Inc., Chicago, Palo Alto, Toronto, Henley-on-Thomas, Sydney, 1973.

[132] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", *IEEE Trans. on Software Engineering*, vol. SE-3, no. 1, January 1977, pp. 85-93.

[133] H. S. Stone, "Special-Purpose vs. General-Purpose Systems: A Position Paper", *Algorithmically Specialized Parallel Computers*, Academic Press, Inc., 1985, pp. 251-252.

[134] H. S. Stone, "Parallel Processing with Perfect Shuffle", *IEEE Trans. on Computers*, vol. c-20, February 1971, pp. 153-161.

[135] H. S. Stone and J. Cocke, "Computer Architecture in the 1990s", *IEEE Computer*, September 1991, pp. 30-38.

[136] Q. Stout, "Mesh-Connected Computers with Broadcasting", *IEEE Trans. on Computers*, vol. c-32, no. 9, September 1983, pp. 826-830.

[137] R. J. Swan, S. H. Fuller, and D. P. Siewiorek, "Cm* - A Modular Multiprocessor", *Proc. AFIPS 1977 Fall Joint Computer Conference*, vol. 46, 1977, pp. 637-644.

[138] T. Szymanski, "Graph Theoretic Models for Photonic Networks", *Proceedings of the New Frontiers: A Workshop on Future Directions of Massively Parallel Processing*, McLean, VA, 1992, pp. 85-96.

[139] C. Thacker and L. Stewart, "Firefly: A Multiprocessor Workstation", *2nd intl. Conference on Architectural Support for Programming Languages and Operating Systems, ACM*, October 1987, pp. 164-172.

[140] S. Todd, "Algorithms and Hardware for a merge sort using multiple processors", *IBM Journal of Research and Development*, vol. 22, no. 5, September 1987, pp. 509-517.

[141] H. C. Torng and N. C. Wilhelm, "The Optimal Interconnection of Circuit Modules in Microprocessor and Digital System Design", *IEEE Trans. on Computers* vol. c-26, no. 5, May 1977, pp. 450-457.

[142] L. W. Tucker and G. G. Robertson, "Architecture and Applications of the Connection Machine", *IEEE Computer*, August 1988, pp. 26-38.

[143] C. H. Tung and C. W. McCarron, "A High Performance Multiprocessor with Partially Orthogonal Multibus and Memory", *ISMM International Conference on Parallel and Distributed Computing, and Systems*, October 1990, pp. 62-66.

[144] R. Vaidyanathan, "Design of Multiple Bus Interconnection Networks for Fan-in Computations", *Proc. 29th Annual Conf. on Comm., Control & Comp.*, 1991, pp. 1093-1102.

[145] R. Vaidyanathan and A. Padmanabhan, "Bus-Based Networks for Fan-in and Uniform Hypercube Algorithms", submitted to *Parallel Computation.*

[146] A. T. White, *Graphs, Groups and Surfaces*, North-Holland Mathematics Studies, 1984.

[147] A. W. Wilson, "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors", *Proceedings of the 14th Annual Symposium on Computer Architecture*, June 1987, pp. 244-252.

[148] D. C. Winsor and Trevor N. Mudge, "Analysis of Bus Hierarchies for Multiprocessors", *Proc. 15th Annual Int. Symp. on Computer Architecture* , June 1988 pp. 100-107.

[149] M. Wolfe and U. Banerjee, "Data Dependence and Its Application to Parallel Processing", *International Journal of Parallel Programming*, vol. 16, no. 2, 1987, pp. 137-179.

[150] Q. Yang and L. N. Bhuyan, "Analysis of Packet-Switched Multiple-Bus Multiprocessor Systems", *IEEE Trans. on Computers*, vol. 40, no. 3, March 1991, pp. 352-357.

[151] Q. Yang and S. G. Zaky, "Communication Performance in Multiple-Bus Systems", *IEEE Trans. on Computers*, vol. 37, no. 7, July 1988, pp. 848-853.

[152] Z. Zlatev, J. Wasniewski, M. Venugopal, and J. Moth, "Optimizing Air Pollution Models on Two Alliant Computers", *Parallel Computation*, Editors: A. E. Fincham and B. Ford, Clarendon Press 1993, pp. 115-133.

VITA

Priyalal Kulasinghe received the B.Sc. degree in Electrical and Electronics Engineering in 1981 from University of Peradeniya, Sri Lanka. He received the M.S. degree in Electrical Engineering in 1990 from Louisiana State University, Baton Rouge. His research interests include design and analysis of algorithms, graph theory and graph algorithms, high performance computer architecture, interconnection networks, and theory of computing.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Priyalal D. Kulasinghe

Major Field: Electrical Engineering

Title of Dissertation: Combinatorial Design and Analysis of Optimal Multiple Bus Systems for Parallel Algorithms

Approved:

A. Elmaghraby
Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

David Bellagarda

Subramanian Kolla

R. Vaidyanathan

C. Y. S. K. Lee

J. Ramanujam

Oscar N. Garcia

Date of Examination:

April 4, 1995