A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns

Wai Hong Ho and Timothy Mark Pinkston*
SMART Interconnects Group
University of Southern California

Abstract

As the level of chip integration continues to advance at a fast pace, the desire for efficient interconnects—whether on-chip or off-chip—is rapidly increasing. Traditional interconnects like buses, point-to-point wires and regular topologies may suffer from poor resource sharing in the time and space domains, leading to high contention or low resource utilization. In this paper, we propose a design methodology for constructing networks for special-purpose computer systems with well-behaved (known) communication characterictics. A temporal and spatial model is proposed to define the sufficient condition for contention-free communication. Based upon this model, a design methodology using a recursive bisection technique is applied to systematically partition a parallel system such that the required number of links and switches is minimized while achieving low contention. Results show that the design methodology can generate more optimized on-chip networks with up to 60% fewer resources than meshes or tori while providing blocking performance closer to that of a fully connected crossbar.

Keywords: *On-chip Interconnects, Communication Model, Low-Contention Communication, Network Partitioning, Irregular Topology*

1 Introduction

Many parallel computers deployed today have special-purpose use, e.g., weather forecast systems, radar signal processing systems, database systems, web servers and data centers, etc. These systems oftentimes run a specific set of characterizable applications during their lifetime. By taking advantage of the known application communication behavior, special-purpose networks may be designed for well-behaved communication requirements, resulting in networks that are more resource/performance-effective. Traditionally, regular topologies have been widely used, but irregular/arbitrary topologies are increasingly being considered these days as switch-based networks are becoming more popular for better support of flexibility, reliability and reconfigurability. Therefore, a systematic way of designing networks with possibly arbitrary topology is gaining importance.

In addition to the system level, specialized network design is also becoming increasingly important at the on-chip level [1]. Many embedded systems consist of one or more application-specific integrated circuits [2], designed for a single or narrow set of applications with highly characterizable (i.e., well-behaved) communication. Encryption systems based on bit permutation, for example, may benefit from specially designed networks [3]. Many researchers are working on automatic synthesis of such systems on a single chip [4] using vendor-provided processing cores with customized placement and route. This system-on-chip (SoC) design concept is becoming more realistic as the size of transistors continues to scale down and the level of integration on a single IC chip increases. Given the current rate of advancement, processing cores on a single chip may number well into the high tens within the next decade. Interconnection networks in such an environment are, therefore, becoming more and more important as they provide connectivity among the processing cores as well as transportation of data from off-chip I/O to the processing cores [5].

Currently, on-chip interconnection networks are mostly implemented using buses and, occassionally, dedicated wire connections. The main limitations of such networks are scalability and restricted sharing of resources between communicating entities. For bus networks, the whole bus is occupied by a single communication even if multiple communications could operate simultaneously on different portions of the bus. For dedicated wire networks, routability and routing area limits the scalability; moreover, connections not overlapping in time still use separate wires, which is an inefficient use of resources. For example, in [4], the interconnection network between cores is generated by considering a simple cost function for each communication between processing cores without analyzing any timing information. There are cases where communications can share a connection because they occur at different times, but they are not combined

*This research is supported partly by NSF grant CCR-0209234.

because the timing information is not considered. This can result in low resource utilization.

Another way to satisfy on-chip as well as off-chip communication is with switched networks. The more commonly used topologies are k-ary n-cubes and fat trees, which are regular. For example, in RAW [6], tiles of processing elements are connected on-chip with a mesh network, which may not match well the needs of target applications. If the provided network has insufficient resources to handle the communication requirements, applications could suffer poor performance. On the other hand, if the network has much more bandwidth than that which is required, chip area for implementing the network (i.e., switches and routing of wires) is wasted. Therefore, it is important to find design solutions based on the communication requirements of the target application(s) to best optimize performance and cost tradeoffs/objectives. Minimizing the required resources is very important for on-chip networks since switches and links may take up a large portion of area, e.g., the new RAW prototype has allocated nearly 50% of the area to the on-chip network.

Some studies suggest that many scientific applications show similar communication characteristics [7, 8]. A large portion of the communication is done via point-to-point communications with large payload sizes upwards of thousands of bytes. The communication patterns are mostly static; each process regularly communicates with a small, fixed subset of other processes. Under such conditions, optimizing the network to support application specific communication patterns is highly possible.

To achieve the best potential performance, one could design the network to support a set of target applications relatively contention-free. Contention in the network increases the latency and decreases the bandwidth substantially, especially for long messages [9]. Some applications (e.g., scientific) have large payload size for point-to-point communications and are highly susceptible to network contention. If not well controlled, contention could have a large negative impact on the performance of parallel programs even if it occurs only in small portions of the network. This is because a stalled process may slow down others that are communicating with it in lock-step fashion. It has been shown that this could account for as much as a 30% degradation in performance [9].

Conflict-free networks (a.k.a., non-blocking networks) have been extensively researched in the past. However, these networks are designed for general-purpose processing; that is, they allow *all* permutations to be realized between communicating end-nodes in a conflict-free manner. However, this capability may not be necessary for a large cross-section of the target application set and, thus, may employ excessive resources. Extensive research has also been done on supporting conflict-free communication for specific patterns like multicast and all-to-all personalized/non-personalized broadcast. For example, a way of supporting all permutations based on dividing a message into smaller fragments and distributing them according to a routing matrix (i.e., in the form of a Latin square) in two passes through the network is described in [10]. A problem with this approach and others like it is scalability, since the number of fragments grows with the size of the network and extra passes are required. Although a spate of such schemes have been proposed in the past, thus far few have considered *application-specific permutations* consisting of a combination of full or possibly only *partial* permutations, i.e., permutations in which some source to destination combinations are *don't cares.*

What's more, in addition to static networks, it may also be of interest to allow the network to be reconfigurable, at run-time [11]. For example, reconfigurable computing paradigms (e.g., FPGAs) have increasingly become more practical alternatives recently [12, 13]. A large portion of the reconfigurable fabric's area is dedicated to routing connections since existing synthesis tools only consider dedicated point-to-point connections. The area may be reduced considerably if the sharing of resources over both the time and space domains is better addressed. Optical networks also have the potential for reconfiguration. Whether free-space or guided-wave (WDM), the physical or logical topology of the network may be made to match the requirements of a particular application/algorithm platform [14, 15].

In this paper, we propose a design methodology for finding minimal topologies that support low contention or contention-free communication for well-behaved (known) communication patterns. The proposed design methodology addresses the problem of optimizing both the temporal and spatial sharing of communication resources. A contention model based on the spatial overlap of messages in time is defined and the communication requirements of target applications are characterized with a set of permutations represented as a set of "cliques." A network generation algorithm based on a recursive bisection technique constructs a network topology via systematic partitioning. In each step, a coloring approach similar to that used in [16] is used for determining the number of links required between partitions. The main difference in our algorithm is that we do not solve the coloring problem exactly until the network is finalized. Instead, we exploit knowledge about the set of permutations that characterizes the communication requirements to obtain a tight lower bound on the number of colors/links required and, thus, reduce the complexity of the algorithm. Results show that the design methodology can produce networks with up to 60% fewer resources than meshes or tori and improve performance by up to 18%.

The rest of the paper is organized as follows. Section 2 starts with some preliminaries on modeling contention and ends with a sufficient condition for contention-free networks. Based on this modeling, the design methodology is proposed in Section 3. An evaluation of network designs derived from the proposed methodology is given

via simulation and analysis in Section 4. Finally, conclusions are drawn in Section 5.

2 Preliminaries: System, Time and Path Conflict Models

Simple models used for defining the necessary and sufficient conditions for contention-free communication are presented in this section. They are applicable to networks with arbitrary or regular topology. With a contention-free model, we can better ensure that the communication bandwidth between processes will be limited only by link bandwidth, and the latency will be free from any congestion factor. This is especially beneficial when message size is long or software overhead is small [9]. This model is used by the methodology described in Section 3 for determining where and how many resources are needed, given certain design constraints.

The contention model consists of two major components which indicate the conflicts within a communication pattern and the resource conflicts in the network. It differs from other similar models in two major ways. First, the proposed model deals only with contention; communications that do not create contention are excluded from the model. Second, both time conflicts and path (space) conflicts are considered in the model.

2.1 System Model

A simplified system model is assumed in which a single processor is attached to each network interface forming an "end-node," but multiple end-nodes may connect to each switch. The topology of the network may be arbitrary. Parallel applications consisting of one process per processor is assumed. A formal definition of a *system* is given below.

Definition 1 *A* system *is represented by a strongly-connected directed graph $G(\mathcal{N}, \mathcal{L})$. The vertices of the graph, $\mathcal{N}$, represent the set of switches and processors. The set of all processors is defined as $\mathcal{P} \subset \mathcal{N}$. The edges, $\mathcal{L}$, represent the set of communication links. A given pair of switches may be connected by more than one link.*

The *communication pattern* of a system is difficult to model since it depends heavily on program behavior and is affected by a variety of factors such as the programming model, data partitioning, distribution, etc. We assume that applications running on the system are well-behaved (i.e., have similar communication patterns on every run) and data independent (i.e., have similar communication patterns for any data set). Many applications and algorithms exhibit this behavior (e.g., see Section 4). Given this, communication patterns can be obtained a number of ways, including the use of hardware monitoring, trace profiling, code analysis, data dependence analysis, and other hardware or compiler techniques. One such

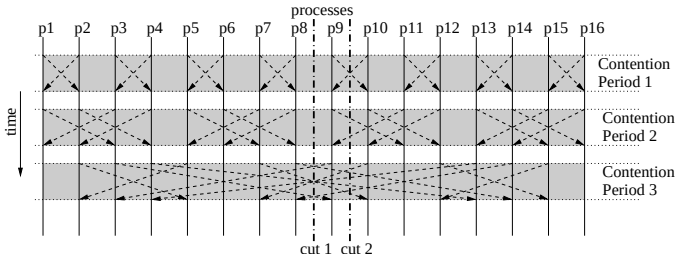


Figure 1. A communication pattern extracted from the CG benchmark [18]. End-points of the dashed arrows indicate the starting and finishing times, respectively, of various communications, i.e., contention periods.

method is assumed to be used to extract message destination and timing information, both of which are important since contention is path and timing sensitive. A formal definition of *communication pattern* is given below.

Definition 2 *The* communication pattern *of an application is characterized by the set of all messages, $\mathcal{M}$, passed between processes. Each message, m, is characterized by its source, $S(m)$, destination, $D(m)$, starting time at which it leaves its source, $T_s(m)$, and finishing time at which it is completely absorbed by its dest, $T_f(m)$.*

2.2 Time Conflict Model

Contention occurs when two or more messages compete for the same non-sharable resource at the same time, i.e., messages overlap in both time and space domains. This part of the contention model identifies potential contention by considering conflicts in the time domain.

The overlap relation defined below identifies *potentially colliding messages* that overlap in time based on the timing information in the communication pattern. Each pair of potentially colliding messages constitutes a potential contention that becomes real if the two messages compete for some common resource, e.g., a link. It is clear that resolving a potential contention event demands the use of separate resources for routing the two messages. Thus, such potential contention can be characterized by the source-destination pairs of two potentially colliding messages. The same potential contention pattern may occur multiple times, especially for programs written using the phase-parallel model [17] where program phases and corresponding communication patterns are repeated. Since they all represent the same contention pattern, it is beneficial to reduce (compress) the overlap relation into a *distinct* set of *potential communication contention* events, as defined below.

Definition 3 *Two messages are said to be* potentially colliding *with one another if they overlap in time, given by an overlap relation, $\mathcal{O}$, defined as follows:*

$\mathcal{O} := \{(m_1, m_2) \in \mathcal{M} \times \mathcal{M} \mid T_s(m_2) \leq T_s(m_1) \leq T_f(m_2) \vee T_s(m_2) \leq T_f(m_1) \leq T_f(m_2) \vee T_s(m_1) \leq T_s(m_2) \leq T_f(m_1) \vee T_s(m_1) \leq T_f(m_2) \leq T_f(m_1)\}$.

Definition 4 *The* potential communication contention set, $\mathcal{C}$, *of an application is the set of all potential contentions, each of which is defined by a 4-tuple in* $\mathcal{P}^4$ *representing the source-destination pairs of two potentially colliding messages:*

$\mathcal{C} = \{(s_1, d_1, s_2, d_2) \in \mathcal{P}^4 \mid \exists m_1, m_2 \in \mathcal{M}, (m_1, m_2) \in \mathcal{O} \wedge S(m_1) = s_1 \wedge D(m_1) = d_1 \wedge S(m_2) = s_2 \wedge D(m_2) = d_2\}$.

The potential communication contention set gives insight into the complexity of the communication pattern. A complicated communication pattern has more potential contentions than a simpler communication pattern and, therefore, a larger potential communication contention set. In order to study the resource requirements of a communication pattern, a *communication maximum clique set* based on the notion of *potential contention periods* should also be defined as given below.

A *potential contention period* is informally defined as the time period over which one or more potential contention events occur such that as messages are being communicated, no message begins or ends before that period concludes. Each potential contention period represents a permutation or partial permutation (i.e., some processors are not communicating) formed by a set of messages that compete against each other for network resources. For example, in the communication pattern shown in Figure 1, the potential contention periods are shaded. If we consider each message as a vertex and the overlap relation as an edge between two messages, the set of messages in a potential contention period forms a *clique*—a complete graph where all vertices are mutually adjacent. For example, the potential Contention Period 3 of Figure 1 is represented by the clique {(2,5), (5,2), (3,9), (9,3), (4,13), (13,4), (7,10), (10,7), (8,14), (14,8), (12,15), (15,12)}. We define the *communication clique set* as the set of all such cliques representing the potential contention periods.

Definition 5 *The* communication clique set, $\mathcal{K}$, *is the set of all cliques representing the potential contention periods in the communication pattern:*

$\mathcal{K} := \{k : ((s_1, d_1), (s_2, d_2), \ldots, (s_j, d_j)) \mid \exists t \in \Re, \exists m_1, m_2, \ldots, m_j \in \mathcal{M}, \forall i \in [1, j], S(m_i) = s_i \wedge D(m_i) = d_i \wedge T_s(m_i) \leq t \leq T_f(m_i)\}$.

The communication clique set contains the set of all partial or full permutations required by the target application(s). In some cases, some partial permutations are covered by a larger partial or full permutation in the set. For example, suppose both cliques $\{(1,2),(2,3)\}$ and $\{(1,2),(2,3),(3,4)\}$ are present in the communication clique set. It is obvious that if a network is contention-free for the latter clique, it is also contention-free for the former. Therefore, the former can be removed to reduce the size of the set. From this observation, a *communication maximum clique set* can be used that eliminates all redundant sub-cliques from the communication clique set. Although the communication maximum clique set does not help in defining the contention-free condition, it is useful in reducing the steps needed by the design methodology described in Section 3.

2.3 Path Conflict Model

The path conflict model defines the resource conflicts in the network, i.e., conflicts in the space domain. The paths between processors are governed by the *routing function*, which is assumed to be *source-based*. Contention is modeled among the links as opposed to the switches. This is valid, for example, for switches in which full internal crossbar functionality is supported.

Definition 6 *A* source-based routing function $F : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}(\mathcal{L})$, *where* $\mathcal{P}(\mathcal{L})$ *is the power set of all links, supplies a single ordered path of links to send a message from source node* n_s *to destination node* n_d *such that* $F(n_s, n_d) = \{l_0, l_1, \ldots, l_p\}$, *where* $l_0, l_1, \ldots, l_p$ *are links along a deterministic path from* n_s *to* n_d.

A *path* is a set of links supplied by the source-based routing function over which a message travels from its source to its destination. The routing path represents the set of resources the message must occupy. Two paths are said to be *conflicting* if one or more of their links are shared. In this case, messages travelling along the two paths at the same time result in contention over the common link(s). These network resource conflicts are collected in a *network resource conflict set* defined below. For non-blocking networks (i.e., crossbars) or fully-connected networks, the network conflict set is empty. Networks with less resources tend to have larger network resource conflict sets than networks with more resources as resource sharing is more common.

Definition 7 *The* network resource conflict set, $\mathcal{R}$, *is the set of all network resource conflicts among routing paths. Each conflict is represented by a 4-tuple representing source-dest node pairs of the two conflicting paths:*

$\mathcal{R} := \{(s_1, d_1, s_2, d_2) \in \mathcal{P}^4 \mid \exists p_1, p_2 \in \mathcal{P}(\mathcal{L}), F(s_1, d_1) = p_1 \wedge F(s_2, d_2) = p_2 \wedge p_1 \cap p_2 \neq \emptyset\}$.

2.4 Sufficient Condition for Contention Freedom

A sufficient condition for contention-free communication in a system is derived from the potential communication contention and network resource conflict sets modeled above. The intersection of these two sets identifies the set of conflicting paths that are active at the same time, which should be empty for contention-free communication.

Theorem 1 *An application that maps to a system is contention-free if the intersection of the potential network contention set and the network resource conflict set results in the empty set, i.e.,*

$\mathcal{C} \cap \mathcal{R} = \emptyset \Rightarrow$ *contention-free communication.*

Proof: Assume the intersection of the potential network contention set and network resource conflict set is empty. If the application is not contention-free, then there must be two messages transmitting at the same time involved in the contention. Therefore, the source-destination pairs of those two messages must be in the potential communication contention set. Since the two potentially colliding messages must share some resources for contention to occur, their paths must share some links. As a result, their source-destination pairs must also be in the network resource conflict set. Hence, the intersection of the two sets should at least contain the source-dest pairs of the two messages, which contradicts the initial assumption. ∎

3 The Design Methodology

A design methodology based on a recursive bisection technique is proposed which allows a network designer to determine what topology and minimal set of resources should be used to efficiently support a target application set. The goal is to provide a systematic way of designing minimal, low-contention networks for systems exhibiting well-behaved communication. The methodology makes use of the contention model described previously, which characterizes time and path conflicts between messages.

First, the communication behavior of the target application(s) is characterized by extracting the potential contention periods. For example, execution traces which logues all communication events can be used for this purpose. Some programs, especially those written using the phase-parallel model, exhibit very well-behaved communication patterns where each process executes the same library calls to commuicate with other processes. In this case, one easy way of extracting the contention period is to assume that corresponding communication library calls are synchronized, i.e., the calls start and end at the same time. In reality, the execution time in between communication calls is different, resulting in *time skew* between processes. This mismatch in alignment may introduce new potential conflicts due to new messages possibly being sent before messages from previous calls finishing, creating interaction between adjacent communication library calls. If the time skew relative to message length is short, it may suffice to assume that each communication library call represents one contention period as is assumed here. Redundant communication patterns are removed from the set of identified contention periods in creating the potential communication clique set.

Central to the design methodology is the idea of decomposing a network that meets the communication requirements of the application set into one whose components also meet the design constraint requirements of the target implementation. This is done via recursive bisection. Initially, a network is created with a single "megaswitch" (i.e., crossbar) connecting all end-nodes. Although non-blocking, such a megaswitch would be prohibitively costly, and would likely violate certain design constraints. Thus, systematically partitioning a larger switch into smaller ones is key to the design methodology. In each partitioning step, a simulated annealing technique is applied to optimize the network partitions for both placement of processors in the switches as well as determining the best routes for messages passing through the switches. By assigning conflicting communications (as indicated by the potential communication contention set) to different links, the intersection of the potential communication contention set and the network resource conflict set is ensured to be empty, i.e., the network is ideally contention-free. A quick way of estimating the number of links required between partitions based on a fast coloring algorithm is used to reduce the complexity.

The algorithms needed to carry out this design methodology are given in the Appendix. Before illustrating the design methodology through an example in Section 3.4, we first discuss some key components: specifically, partitioning, routing, and fast coloring.

3.1 Partitioning

Consider the simple case of partitioning a single switch into two switches. Associated with each switch is a set of processors and a set of communications, which includes messages generated from and destined to the processors connected to the switch as well as messages passing through the switch. The connections between two switches is represented by a *pipe*, which is characterized by two opposing sets of communications that go through it—one for each direction. The width of the pipe (which affects contention properties) is determined by coloring the *conflict graphs* [14] of the two sets of communications going through it. The conflict graph is obtained by assigning the communications passing through the pipe as nodes and the potential temporal conflicts (from the potential communication contention set) between communications as edges.

To support contention-free communication between switches via a pipe, the pipe must supply a sufficient number of links so that temporal conflicting communications are handled by separate links. This is to ensure that the intersection of the network resource conflict set and the potential communication contention set is empty. It follows that any adjacent nodes in the conflict graph of the communication set representing the pipe must not share the same link; otherwise, the two conflicting communications would cause contention for that link. Therefore, the problem of finding the minimum number of links required can be transformed into a problem of finding the minimum number of colors required to color the conflict

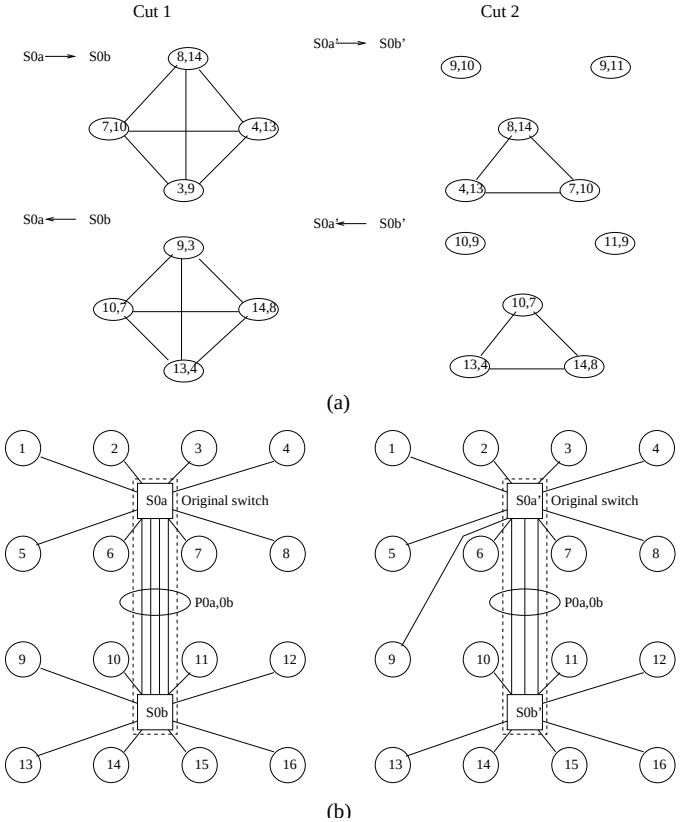


Figure 2. (a) Conflict graphs for the two cuts in the previous figure. Vertices represent a communication going through the cut, and edges represent conflicting communications. Communication going in opposite directions are handled separately. (b) Two different networks obtained by partitioning the original switch.

graph. Since the communications going along different directions of the pipe do not interfere with one another (assuming full-duplex links), the coloring problem of the conflict graphs for different directions can be treated separately.[1] The overall number of links required is equal to the maximum cardinality of the two sets of colors obtained by solving the two coloring problems, as illustrated by the following example.

Suppose each process in the sample communication pattern shown in Figure 1 is mapped to one end-node. Consider two different ways of partitioning the initial single switch S_0 that connects all end-nodes—namely, Cut 1 and Cut 2. Cut 1 divides the end-nodes into two sets, Nodes 1 thru 8 and Nodes 9 thru 16 connected by two switches S_{0a} and S_{0b}, respectively. Eight messages from Contention Period 4 pass through the cut, characterized by the conflict graphs as shown in Figure 2(a). From the

[1] Unidirectional links may also be assumed if the communication pattern is asymmetric; however, connectivity needs to be considered, which requires extra steps to ensure that the resulting network is strongly connected.

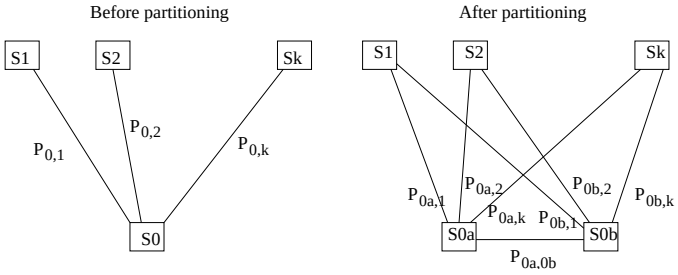


Figure 3. The general case for partitioning a switch. Connections and other switches not related to Switch S_0 are not shown.

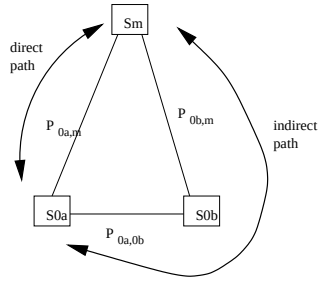


Figure 4. Two possible paths for communication between Switch S_{0a} and Switch S_m.

conflict graphs, the number of colors required to color the graph is four for both directions. Therefore, four links are required between the partitions in order to ensure contention-freedom. For Cut 2, ten messages pass through the intersection. Despite the fact that there are more messages crossing this intersection than for Cut 1, the number of links required is only three, which is one less than Cut 1 as shown in Figure 2(b). Consequently, the total number of messages between the two partitions is not an accurate measurement of the number of links required for contention-free communication; rather, the maximum number of conflicting communications over all contention periods gives a better estimate.

Extending this simple partitioning example to the general case of partitioning, consider a switch S_0 is partitioned into two switches S_{0a} and S_{0b}. Before switch S_0 is partitioned, it is connected to k other switches S_1, S_2, ..., S_k. Figure 3 shows the topologies before and after the partitioning. A pipe $P_{0,m}$ connecting S_0 to another switch S_m is split into two pipes $P_{0a,m}$ and $P_{0b,m}$ connecting S_{0a} and S_{0b} to S_m, respectively. A new pipe $P_{0a,0b}$ is introduced between S_{0a} and S_{0b}. The set of end-nodes connected to S_0 is distributed to S_{0a} and S_{0b}.

3.2 Routing

Routing of messages is handled by distributing the set of communications of S_0 to S_{0a} and S_{0b}. Figure 4 shows two possible routes for communication between S_{0a} and S_m. The route can be different for messages going in different directions, i.e., messages going from S_{0a} to S_m and messages going from S_m to S_{0a} may take different routes.

A straightforward way of distributing the communication is to choose the direct path for all communications, i.e., choosing the link from S_{0a} to S_m instead of the path through S_{0b}. However, this is not necessarily the optimal way of routing the communication when optimality of partitioning is taken into account. Instead, our methodology considers each communication in $P_{0a,m}$ (which inherits all communications from $P_{0,m}$) and tries to move it to $P_{0b,m}$ via $P_{0a,0b}$. A simulated annealing technique is applied to minimize the total number of links required by $P_{0a,m}$, $P_{0b,m}$ and $P_{0a,0b}$. This is repeated for all the pipes that connect the original switch S_0.

3.3 Fast Coloring and Complexity Analysis

Finding the minimum number of links required by a pipe is critical in determining the quality of a partitioned configuration. A way of solving the coloring problem of the conflict graphs with minimal complexity is therefore needed. Although the coloring problem is NP-hard [19], a fast way of obtaining a close lower bound (i.e., estimate) on the number of colors needed can be done heuristically by comparing the communication maximum clique set against the communications that pass through a pipe (see the *Fast_Color* procedure in the Appendix).

Each clique in the communication maximum clique set contains a set of communications that are potentially conflicting with each other in time. The communications that are in common for both the clique and the pipe will form a clique in the conflict graph as well. In the CG example in Figure 1, the maximum communication clique set contains three cliques representing the three contention periods, respectively. Consider the five communications, $(9, 10)$, $(9, 11)$, $(8, 14)$, $(4, 13)$ and $(7, 10)$, that go from switch S'_{0a} to S'_{0b} in Cut 2. Communication $(9, 10)$ belongs to the first contention period, communication $(9, 11)$ belongs to the second period, and the rest belong to the third period. Therefore, the maximum number of common communications between the pipe and the maximum communication clique set is three for the clique representing the third contention period. In this case, this number is exactly the number of colors required to color the conflict graph. In general, this maximum number represents a close lower bound on the number of colors needed to color the conflict graph.

As the partitioning goes on, communication is spread across the network, and the number of communications passing through a pipe decreases. The number of potential conflicts in the pipe will also decrease. This leads to a decrease in the number of links required by the pipe. When the number of links required drops below three, the coloring problem becomes solvable in polynomial time [19]. This is important since formal (not fast) coloring must be done in the final step of the partitioning algorithm to find the exact number of required links.

The fast coloring algorithm runs in $O(KL)$ time, where K is the number of different contention periods (communication cliques) and L is the size of the cliques. The main partitioning algorithm bisects the network $O(N)$ times. For each bisection, our simulated annealing technique only considers a constant number of moves between partitions. Each move takes $O(NKL)$ time to update the estimated number of links for all the pipes connected to the two involved partitions. After that, routes will be optimized the $O(N)$ number of pipes, each taking a $O(KL)$ amount of time assuming a constant number of moves are considered between them. At the finalization of the topology we need to run the true coloring algorithm to find out the exact number of links between partitions. We assume that the number of links are less than or equal to 2 between any pair of partitions so that the final coloring is done in polynomial time or $o(KL)$. Therefore, the overall complexity of the algorithm, dominated by the main partitioning algorithm, is $O(N^2KL)$.

3.4 Illustration Through a Design Example

The main partitioning algorithm given in the Appendix uses a simulated annealing technique. The initial network constructed from a single switch connecting all processors is partitioned recursively until specified design constraints are met by all switches. A number of different design constraints can be used. One of the simpler ones used in this example is maximum node degree, which limits the number of inputs and outputs of each switch to be less than some constant.

When the main partitioning algorithm partitions a switch, it first creates a new switch and automatically moves half of the processors in the original switch to the new switch. The number of links required is then estimated by applying fast coloring on the pipe connecting the original switch and the new switch and also on the pipes connecting them to other switches. Communications that use the two switches are then assigned a routing path using the simulated annealing technique given by the *Best_Route* procedure in the Appendix.

Possible moves between the partitions are then considered. The expected number of links required after the move is estimated by the fast coloring algorithm, assuming the use of direct paths. Only those moves that give a less expected (estimated) number of required links that do not make the partition unbalanced are considered. The difference between the number of processors on the original and new switches are limited to two in order to allow balanced movement of a node from one switch to another to be simplified. If no such moves are found, the algorithm proceeds to check the design constraints and partition another switch, if required.

A step-by-step illustration of the design methodology applied to the CG benchmark of Figure 1 is given below. A design constraint which limits the maximum node degree to five is imposed. This allows straightforward comparison of the generated network to mesh and torus. The initial switch is partitioned using Cut 1 shown in

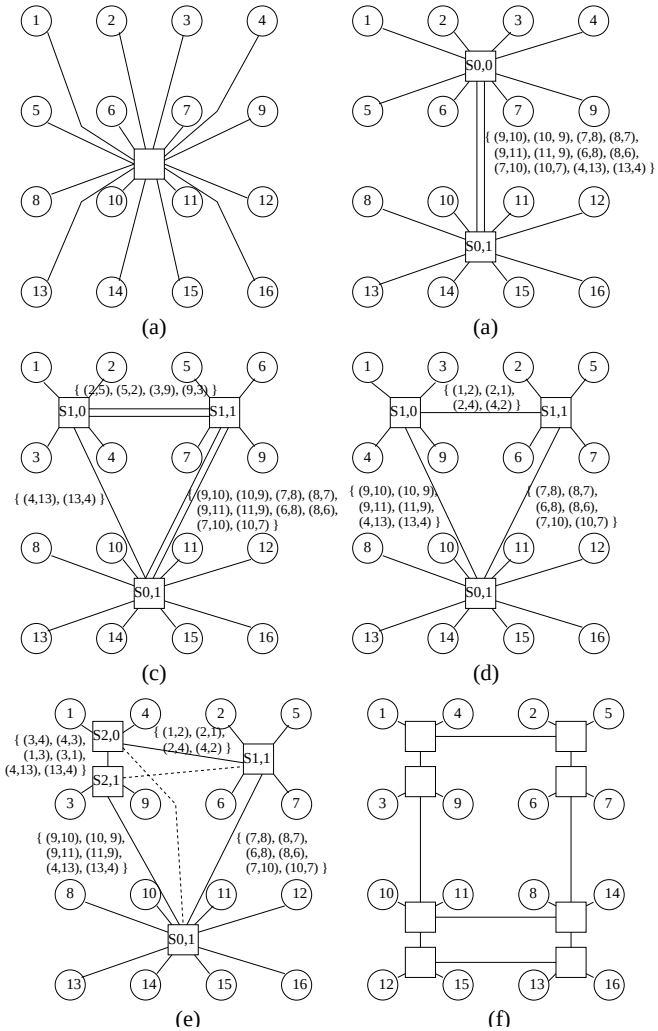


Figure 5. Partitioning in progress. Communications that go across switches are represented by the ordered pairs enclosed by curly brackets; communications in different contention periods are listed in different rows. The dotted lines in (e) indicate empty pipes between the switches. The final network is shown in (f).

Figures 1 and 2. The eight communications between the two switches all belong to the third contention period. Four of the communications go in the forward direction of the pipe and the other four go in the backward direction. Therefore, the fast coloring algorithm returns four as the number of bidirectional links required. After that, possible moves between the switches are considered. In this case, Processor 9 is selected since the estimated number of required links is only three by moving it to form Cut 2. Finally, Processor 8 is moved to form the network in Figure 5(b). The number of links required is only two after the move. No further moves are considered because none decrease the expected number of required links.

The two resulting switches, however, still violate the design constraint. The algorithm therefore selects $S_{0,0}$ to form switch $S_{1,0}$ and $S_{1,1}$ as shown in Figure 5(c). Processor 9 is moved from $S_{1,1}$ to $S_{1,0}$ and, next, Processor 2 is moved from $S_{1,0}$ to $S_{1,1}$ to form the network in Figure 5(d). So far, only direct paths are used in the network because the number of links required does not change even if the indirect paths are considered in these steps. Next, $S_{1,0}$ is selected for partitioning. The resulting network is shown in Figure 5(e). Communications (4,13) and (13,4) are redirected via $S_{2,1}$ (i.e., indirect path is used) to decrease the number of links required.

The rest of the network is systemically partitioned in a similar way to form the final network configuration shown in Figure 5(f). The main algorithm finalizes the number of links between the switches by executing graph coloring (not fast coloring) on each pipe. At this point, this is trivial because the number of links between switches is at most one. The partitioning algorithm terminates at this point since all the switches satisfy the design constraint, i.e., node degree is less than or equal to five. Clearly, the generated network requires far fewer resources than a mesh for this application. The link utilization, layout area and performance of a cross-section of networks generated by our design methodology are further analyzed and compared with other networks in the next section.

4 Evaluation and Preliminary Results

In this section, we evaluate the usefulness of our proposed design methodology using five benchmark programs, most of which exhibit well-behaved communication. Execution traces are obtained by running the benchmark programs and analyzed to extract the communication pattern. The communication pattern is then fed into a topology generator written in C++. The design methodology is applied by the topology generator in constructing minimal, low-contention networks for the communication patterns. The generated topologies are compared with other networks in terms of resource usage by devising possible floorplans for each. For performance comparisons, trace driven simulation using IRFlexSim [20]—a flit-level network simulator—is performed on the generated topology. The performance is compared to a fully-connected crossbar, mesh and torus.

The parallel benchmark suite is composed of a set of pseudo applications and kernel codes which represent typical scientific workload. We have chosen five benchmarks in our study for their well-behaved characteristics and simple programming model which enable us to straightforwardly extract their communication patterns. The five benchmarks used are BT (Block Tridiagonal solver), CG (Conjugate Gradient), FFT (3-D Fast Fourier Transform), MG (Multi-Grid solver) and SP (Scalar Pentadiagonal solver) taken from NAS [18]. The BT and SP benchmarks exhibit very similar communication patterns which consists mostly of point-to-point communications (both are based on a similar algorithm).

The CG benchmark's communication behavior is dominated by reduction and matrix transpose communication in the main loop. The FFT benchmark is implemented by a 2-D blocking algorithm, the communication of which is mainly all-to-all communication within a row or column. Finally, MG consists mainly of reduction to all nodes and broadcast communication of short messages.

The communication patterns of the five benchmarks are characterized by compiling the benchmark programs with the MPE profiling library included in the MPICH library [21]. The programs were executed on a PC-cluster with 8-node and 16-node configurations, except for the BT and SP benchmark on which a 9-node configuration is used since these benchmarks require a number of processors equal to a perfect square. An execution trace which consists of a logue of all communication events and the parameters passed to communication library calls is obtained. The communication pattern analyzer, written in C++, locates calls to the same communication library function across all the processors from the communication event logue. The communications from the same communication library call are consider to be a part of the same contention period, potentially overlapping in time ideally if there is no time skew between processors.

The communication patterns are extracted as a set of distinct contention periods represented by communication cliques. The contention periods obtained by this method do not take into account any unpredictable interaction that might occur between consecutive communication library calls across all processors. Networks generated by applying the design methodology on these contention periods, as we assume, would be contention-free in the ideal case of there being no time skew between processors. However, as spurious time skew could cause partial overlap at the edges of consecutive "distinct" contention periods (creating additional contention periods not accounted for), the above assumption trades off potential blocking for added simplicity and reduced resources in the generated networks. Results in the following sections confirm the validity of this tradeoff.

4.1 Resource Comparison

On-chip networks must satisfy 2-D planar layout constraints. Therefore, instead of comparing the number of links and switches directly, we find a possible floorplan for each generated network manually and compare the amount of area dedicated to switches and links to that of a corresponding standard mesh network. The chip is assumed to be composed of processor tiles (à la the MIT RAW [6]), each consisting of a single processor and a network interface at the corner of the tile. Space is reserved at the corner for the switch and at the border for possible links that go across the tile. Figure 6(a) illustrates this for a mesh network. To increase the flexibility in constructing different topologies, we also assume the layout of tiles are such that they can take on different

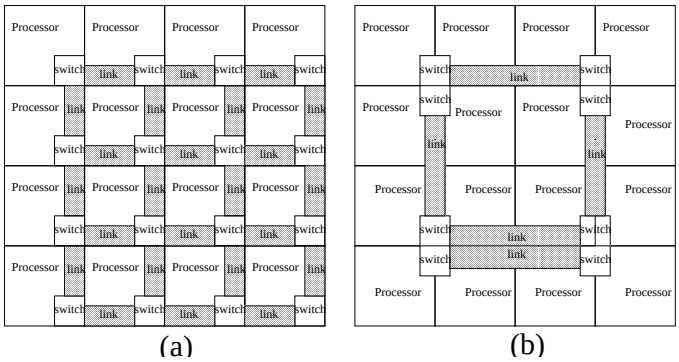
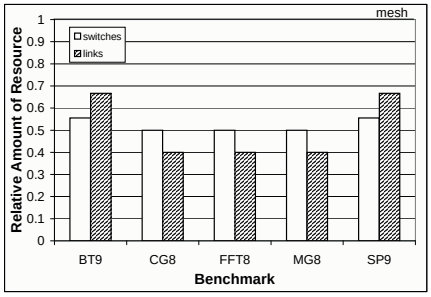


Figure 6. Floorplans for: (a) original mesh network and (b) generated network for CG.

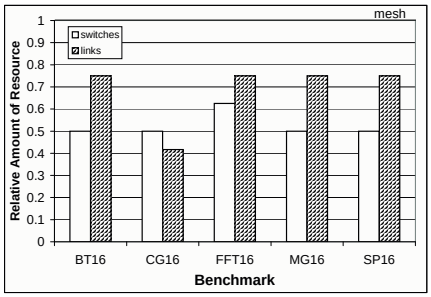
orientations and can share reserved space at the corners of adjacent tiles. This allows a reduction in the amount of reserved space at tile corners if it can be determined that multiple tiles need to connect to a given switch. For example, Figure 6(b) shows the generated network for the CG benchmark, which consists of rotated tiles sharing switch ports with their neighbors. Compared with the regular fixed tile approach, this variable orientation approach may support topologies other than meshes but with a more complicated tiling design.

The area used by switches and links are modeled assuming the following design contraints. The number of ports for all switches are assumed to be five and, thus, each individual switch consumes the same area irrespective of topology. The links between physically adjacent switches (see Figure 6(b)) are assumed to consume zero area, but the other links are assumed to take up an amount of area proportional to the "manhattan" distances between the switches they connect. For simplicity, it is assumed to be equal to the number of tiles crossed, i.e., one for Figure 6(a) but as much as two for links between non-adjacent switches in Figure 6(b).

A comparison of switch and link areas for the generated networks and meshes is shown in Figure 7. For torus networks, we can easily know that the same total switch area as that in a mesh is needed, but double the total link area is required. For 8-node or 9-node configurations shown in Figure 7(a), the reduction in area by the generated network versus the others is significant for the CG, FFT and MG benchmarks. They consume only 50% switch area and 40% link area compared to the mesh network (20% link area compared to the torus). The BT and SP benchmarks have more complicated communication patterns which leads to a higher requirement on network resources. However, the reduction is still substantial: switch and link area are reduced for the mesh by 45% and 23%, respectively (62% less link area as compared to the torus). Results are similar for 16-node configurations shown in Figure 7(b). The CG benchmark achieves the best resource reduction ratio with a 50% and 58% savings in switch and link area for the mesh, respec-

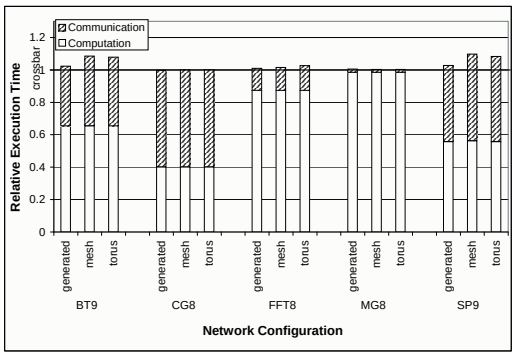


(a)

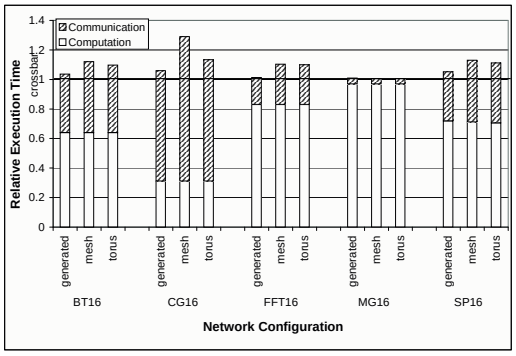


(b)

Figure 7. Resources for generated networks normalized with respect to a mesh: (a) 8 or 9-node configurations and (b) 16-node configurations.



(a)



(b)

Figure 8. Performance of generated networks, mesh and torus normalized with respect to a crossbar network: (a) 8 or 9-node configuration and (b) 16-node configuration.

tively (79% savings in link area for the torus). The other generated networks are very similar with approximately 40-50% reduction in switch area and 25% reduction in link area as compared to the mesh (63% reduction in link area as compared to the torus). Clearly, the reduction in resources depends on the complexity of the communication pattern. The CG benchmark has the simpliest communication patterns of the five benchmarks. However, complexity in communication patterns may change with the number of nodes. For example, the relative amount of resources required by the FFT and MG benchmarks increases from 8-node configurations to 16-node configurations due to the increase in complexity of the collective communications dominating the two benchmarks.

4.2 Performance Comparison

The performance of the generated networks are measured via trace-driven simulation using execution traces obtained from the aforementioned benchmarks. IRFlexSim allows the use of different topologies on the same trace file to facilitate performance comparisons. For each benchmark, a fully-connected nonblocking crossbar network, a mesh, a torus and the generated topology are used for both an 8 or 9-node configuration and a 16-node configuration. Each processor is assumed to be attached to the network via one physical link to a switch. Switches are connected by one or more physical links according to the topology generated. Each physical link is assumed to have 3 virtual channels. This helps to alleviate contention problems for the mesh and torus. It may also help the generated network since contention may occur due to the simplifying assumption we have taken in determining the contention periods as described earlier. Physical links and flit sizes are assumed to be 32-bits and operate at 800 MHz. These parameters are similar to those used for the on-chip router of the Alpha 21364 [22]. Delay through a physical link is assumed to be equal to its length in number of tiles, with a minimum of one clock. Send and receive overhead of ten cycles is assumed [23].

Dimension-order routing (DOR) is assumed on the mesh and true fully-adaptive routing (TFAR) is assumed on the torus. Source routing is used for the generate topology to give the necessary flexibility to assign communications to different routes and minimize contention. Deadlocks in the torus and generated networks, if occur, are handled by detection and regressive recovery, i.e., deadlocked messages are killed and retransmitted. For all execution traces simulated on all of the above networks and configurations, no deadlocks were detected. This result is consistent with prior observations [20].

The total execution time and communication time (including waiting time and overhead) for the five bench-

marks on various topologies are measured and plotted in Figure 8. For 8-node or 9-node configurations, the differences in performance between the generated networks and the other networks are small. For the CG benchmark, the mesh network is already contention-free for the 8-node configuration, so there is no improvement in performance using the generated topology, torus or non-blocking crossbar. For the FFT and MG benchmarks, the communication to computation ratio is small for the 8-node configurations. This results in very little difference in performance for the various topologies. The BT and SP benchmarks have more complicated communication patterns for the 9-node configurations compared to the other benchmarks. The performance for the generated network is better for these two benchmarks. Communication time is approximately reduced by 8%, resulting in a overall improvement in execution time of 5%.

For 16-node configurations, the improvement of the generated network over the mesh and torus is more prominent. The communication to computation ratio is generally higher in 16-node configurations. The CG benchmark shows the best improvement in performance. The generated network reduces the communication time by about 26% compared to the mesh and by about 10% compared to the torus. The overall improvement in performance is about 18% compared to the mesh and 8% compared to the torus. The BT, FFT and SP benchmarks also show some improvement in performance (about 6–10% for the generated networks). There is no significant difference in performance for MG, partly due to the relatively small communication to computation ratio and partly due to the small message size which makes the program more sensitive to latency than contention.

In general, the generated networks perform very well compared with the prohibitively expensive non-blocking crossbar, which is the ideal case. The difference in performance between the generated network and the crossbar is less than 4% for all the benchmarks and configurations. We believe this difference exists due to the time skew between processes, which creates interaction between consecutive potential contention periods. Tori, on the other hand, only show significant improvement in performance over the mesh network on the CG benchmark with 16-node configuration. Since a torus requires two times the link resources compared to a mesh network due to the wrap-around links and the 2-D constraint of a chip, it is not a very cost-efficient on-chip network.

We have also run the execution traces from BT and FFT benchmarks on the network generated for the CG benchmark to see the ability of the generated networks in handling communication patterns other than the one for which it is specifically designed. This, in effect, shows sensitivity of the generated networks to possibly varying communication patterns of applications in a workload. Results show that FFT runs fine on the network generated for CG. Although not plotted due to space limitations, the degradation in performance as compared to the network generated for FFT is less than 2%. This is because FFT's communication consisting of all-to-all communications among rows and columns of a 2-D array is similar to the reduction communication pattern which dominates in the CG benchmark. However, the BT benchmark suffers an approximate 20% degradation in performance when run on the CG network as compared to the network generated especially for BT, e.g., only slightly worse than mesh. Hence, generated networks are still applicable under moderate changes in communication patterns.

Judging from the above results, the design methodology is especially suitable for applications that are communication bound (high communication to computation ratio) and have characterizable communication (balanced and well-behaved workload). In this case, both significant improvement in performance and reduction in network resources are highly possible. Although the performance may not improve by applying the design methodology to applications that have a low communication to computation ratio or characterizable communication, the methodology can still generate a network that uses significantly fewer resources than mesh on-chip networks without significantly degrading the relative performance. This is attractive for on-chip networks where resources or power maybe are limited.

5 Conclusion and Future Work

In this paper, a sufficient conditions for contention-free comunication based on the notion of both temporal and spatial resource sharing is introduced. A design methodology based on this model is proposed for generating minimal, low-contention networks for applications with well-behaved communication patterns. We apply our methodology to the communication patterns extracted from five parallel benchmarks. The resulting networks uses up to 60% fewer resources than meshes and tori, while providing performance closer to that of a fully-connected non-blocking crossbar. The generated networks also show significant improvement in performance (up to 18%) over mesh networks for well-behaved applications that have a high communication to computation ratio. In the near future, the design methodology will be extended to shared memory programs with well-behaved memory access patterns. We believe that automated design of interconnection networks for special-purpose computing systems such as can be done using our proposed design methodology will become increasingly important as the level of chip integration continues to advance. Although we have only considered reduction of resources in this paper, this work can be extended to include other important optimization criteria such as power to produce power-efficient on-chip networks and optimization over multiple objectives.

Acknowledgements

We are grateful for the insightful comments and suggestions made by Yuanyuan Yang and the anonymous reviewers.

Appendix

Main Partitioning Algorithm:

1. Form an initial network with a single "mega" switch connecting all processors.
2. If some switch does not satisfy the design constraints, then *goto 4*.
3. Finalize the number of links in the switches by running the formal coloring algorithm. If the all switches satisfy the design constraints, then *End*.
4. Randomly select a switch S_i that violates the design constraints.
5. Create a new switch S_j and randomly move half of the processors attached to S_i to S_j. Create a new pipe, $P_{i,j}$, between S_i and S_j. For each switch to which S_i connects, create a new pipe between that switch and S_j.
6. Calculate the best route for communications through the two switches using *Best_Route(S_i, S_j)*.
7. Using *Fast_Color(Pipe P)* on the pipes connected to S_i and S_j, get the expected number of links required for each possible move of processors between the two switches assuming direct routes.
8. If no move with less expected number of required links is found, *goto 2*. If the moves with less expected number of required links make the number of processors between S_i and S_j unbalanced by more than 2, *goto 2*.
9. Select a processor to move between S_i and S_j which has the best expected number of required links and does not make the number of processors between S_i and S_j unbalanced by more than 2. *Goto 6*.

Procedure Best_Route(S_i, S_j):

1. Mark all pipes connecting to S_i as unoptimized.
2. Select an unoptimized pipe $P_{i,k}$ which connects S_i to S_k.
3. For each communication in the $P_{i,k}$, try the indirect route which passes through $P_{i,j}$ and $P_{j,k}$ to find moves that can decrease the number of links required. Commit all such moves. Mark $P_{i,j}$ to be optimized.
4. If there are more unoptimized pipes connectd to S_i, *goto 2*, else *return*.

Procedure Fast_Color(Pipe P):

1. Find two sets of communications, C_f and C_b, through the pipe for the forward direction and the backward direction, respectively.
2. For each clique K in the communication maximum clique set, the number of common communications with the sets C_f and C_b is calculated, i.e., $\|K \cap C_f\|$ and $\|K \cap C_b\|$. The maximum of these is *returned*.

References

[1] W. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proc. of the Design Automation Conference*, pp. 684–689, June 2001.

[2] W.H. Wolf. "Hardware-software Codesign of Embedded Systems". *Proc. of IEEE*, 82(7):967–989, July 1994.

[3] R.B. Lee, Z. Shi, and X. Yang. "Efficient Permutation Instructions for Fast Software Cryptography". *IEEE Micro*, 21(6):56–69, Nov.-Dec. 2001.

[4] R.P. Dick and N.K. Jha. "MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, October 1998.

[5] "What will have the greatest impact in 2010: The processor, the memory, or the interconnect?" *Panel Discussion at HPCA8*, www.usc.edu/dept/ceng/pinkston/presentations/statistics.html.

[6] M.B. Taylor et. al. "The Raw Processor - A Scalable 32-bit Fabric for Embedded and General Purpose Computing". In *Proceedings of Hotchips XIII*, August 2001.

[7] J.S. Vetter and F. Mueller. "Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures". In *Proceedings of the IPDPS 2002*, April 2002.

[8] D. Gautier de Lahaut and C. Germain. "Static Communications in Parallel Scientific Programs". *PARLE'94*.

[9] S.Q. Moore and L.M. Ni. "The Effects of Network Contention on Processor Allocation Strategies". In *Proceedings of the 10th International Parallel Processing Symposium*, pp. 268–273, 1996.

[10] Y. Yang. "Routing Permutations with Link-Disjoint and Node-Disjoint Paths in a Class of Self-Routable Networks". *Proceedings of the International Conference on Parallel Processing*, pp. 239–246, August 2002.

[11] Timothy Mark Pinkston. Theoretical support for deadlock-free dynamic network reconfiguration. In *Workshop on Self-Healing, Adaptive, and self-MANaged systems (SHAMAN) with ICS*, June 2002.

[12] J. Rose and S. Brown. "Flexibility of Interconnection Structures for Field-Programmable Gate Arrays". *IEEE J. of Solid-State Circuits*, 26(3):277–282, March 1991.

[13] K. Compton and S. Hauck. "Reconfigurable Computing: A Survey of Systems and Software". *ACM Computing Surveys*, 34(2):171–210, June 2002.

[14] X. Qin and Y. Yang. "Nonblocking WDM Switching Networks with Full and Limited Wavelength Conversion". In *Proc. of the 10th Int. Conf. on Comp. Comm. and Networks (IC3N'01)*, pp. 48–54, October 2001.

[15] M. Raksapatcharawong and T.M. Pinkston. "Design Issues for Core-based Optoelectronic Chips: A Case Study of the WARRP Network Router". *IEEE JSTQE, Special Issue on Smart Photonics*, 5(2):330–339, March 1999.

[16] Q.P. Gu and S. Peng. "Wavelengths Requirement for Permutation Routing in All-optical Multistage Interconnection Networks". In *Proc. of the 14th IPDPS 2000*, pp. 761–768, May 2000.

[17] D. Hwang and Z. Xu. In *Scalable Parallel Computing*. WCB/McGraw Hill, 1997.

[18] "The NAS Parallel Benchmark". *http://www.nas.nasa.gov/Software/NPB*.

[19] T. Coremen, C. Leiserson, and R. Rivest. In *Introduction to Algorithms*. McGraw-Hill, 1997.

[20] Sugath Warnakulasuriya and Timothy Mark Pinkston. Characterization of Deadlocks in Irregular Networks. *J. of Par. and Dist. Comp.*, 62(1):61–84, Jan 2002.

[21] "MPICH: A Portable Implementation of MPI". *http://www-unix.mcs.anl.gov/mpi/mpich/*.

[22] S. Mukherjee et al. The Alpha 21364 Network Architecture. In *Symposium on HOT Interconnects 9*, pp. 113–117, August 2001.

[23] D.E. Culler et al. "LogP: A Practical Model of Parallel Computation". *Communications of the ACM*, 39(11):78–85, November 1996.