Power-Aware Routing for Well-Nested Communications On The Circuit Switched Tree

Hatem M. El-Boghdadi
Computer Engineering Department
Cairo University, Giza, EGYPT
helboghdadi@eng.cu.edu.eg

Abstract

Although algorithms that employ dynamic reconfiguration are extremely fast, they need the underlying architecture to change structure very rapidly, possibly at each step of the computation. This increases the power requirement of such algorithms which is not acceptable in nowadays devices that strive to reduce the power requirements. This paper deals with the circuit switched tree (CST), an interconnect used to implement dynamically reconfigurable architectures.

In this paper, we introduce a new technique called Power Aware Dynamic Reconfiguration (PADR). Under this technique, we propose a power-aware algorithm for configuring the CST and scheduling a class of communications, called the well-nested communications on the CST. We show that the algorithm is power optimal. The algorithm requires only local information at processing elements (PEs), yet it correctly establishes paths between communicating PEs. We also show that the algorithm is optimal and efficient.

1 Introduction

Dynamic Reconfiguration has been proven to be a very powerful computing technique. Models such as the reconfigurable mesh (R-Mesh) [5] provide very fast solutions to many problems. This is due to the ability of these models to change the interconnection between processors very fast, possibly at each step of the computation. Changing the interconnection between processors is usually done through altering configuration of the switches. This in turn translates to increasing the power requirements for the algorithms being considered.

Nowadays, the speed of the algorithms is not the only issue the algorithm designer should care about. Energy-efficiency is another important issue in most architectures of portable devices. Energy-efficiency is one of the biggest hurdles against deploying dynamically reconfigurable structures in portable devices. Algorithms for such devices should be designed with an eye on the power consumption and not only the speed.

In this paper we are interested in algorithms that get the advantage of dynamic reconfiguration (speed) with reduced power requirements. We call the new technique *Power Aware Dynamic Reconfiguration* (PADR). We achieve this by designing algorithms that dont require the switches to alter its configuration at each step. A switch is configured by setting a one-to-one connections between its inputs and its outputs. This configuration is required by the algorithm to satisfy certain communications requirements. An algorithm designed under the PADR umbrella, sets each switch into certain configuration in a certain step and then tries to satisfy all communications requirements that need this configuration in the following steps before altering the switchs' configurations to satisfy other communications requirements. This indeed reduces the power requirements of the algorithm.

One way to implement dynamically reconfigurable architectures is through self reconfiguration where the architecture can generate the configuration information from within and does not need configuration to be loaded from outside. This technique reduced the gap between theoretical models [5] and practical platforms such as conventional Field Programmable Gate Arrays (FPGAs) and its variations [2].

Sidhu *et al.* [7] introduced the Self-Reconfigurable Gate Array (SRGA) architecture which is an array of PEs and has self-reconfiguration ability. Each row/column is connected using a circuit switched tree (CST). The CST basic structure is a binary tree whose leaves are processing elements (PEs) and whose internal nodes are 3-sided switches. Edges of the tree are full duplex links. Figure 1 shows how communications could be established on the CST.

Two PEs can communicate by setting a path from one PE (the source) to the the other PE (the destination.); the pairing between a source PE and a destination PE is called a

1-4244-0910-1/07/$20.00

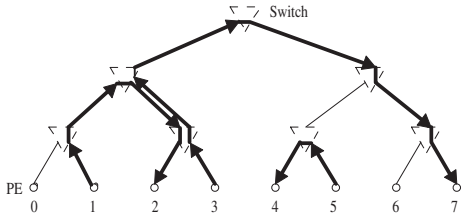

Figure 1. Communications over the CST.

communication. This requires configuring all the switches in the path from the source PE to the destination PE to establish the required path. A set of communications can be performed simultaneously if no two communications use the same edge in the same direction [3]. Such communication set is called a *compatible* set. However, if some communications need to use the same edges in the same direction, then performing this set of communications requires multiple rounds of scheduling. Each round may require the same switch to have a different configuration. Alternating between configurations is a major source of power consumption which we strive to decrease.

Also the CST was shown to schedule some important classes of communications optimally [3]. One of these classes is the *well-nested* sets [3] which is a superset of the communications required by the segmentable bus; a fundamental reconfigurable architecture. Also the CST can be used as an interconnect in Infiniband networks [4] and network on Chips [1].

Figure 2 shows an example for well-nested sets. In the figure, all communications are in the same direction; i.e. the communication set is *oriented* to the right. Performing the communication set in a number of rounds is called *scheduling*. Each round of the schedule performs some of the communications in the set that are compatible. If at most w communications require to use the same link in the same direction, the communication set is of *width* w.

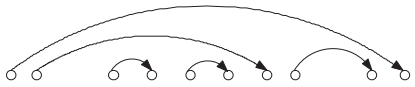

Figure 2. Well-Nested communication set.

This paper presents a PADR-based algorithm for configuring the CST and scheduling a set of communications. Here, we consider scheduling oriented well-nested sets.

Sidhu *et al.* [7] presented an algorithm to configure the switches for only one communication; This algorithm was extended to support multiple communications if they are disjoint communications [3]; two disjoint communication dont use the same tree edges even in different directions.

Recently, Roy *et al.* [6] presented an algorithm for configuring and scheduling communications sets over the CST. The algorithm first assign an ID to each communication and use this ID to configure the switches and set the path between the communicating PEs. Although the algorithm was shown to be optimal and efficient for well-nested communications, it could alter the configuration of the same switch at each round of the schedule. In other words, for a well-nested communication set of width w, the schedule requires $\Theta(w)$ rounds and a switch needs $O(w)$ configuration changes. Our algorithm shows that setting the path between corresponding PEs does not need assigning an ID for each communication. Our algorithm is optimal, efficient, and guarantees that for a well-nested communication set of width w, the schedule requires $\Theta(w)$ rounds and a switch needs $O(1)$ configuration changes; i.e. a switch makes a constant number of configuration changes which makes the algorithm power optimal.

The main idea of making the algorithm power aware is that each switch, u, select the *most outer* communication, $O_c(u)$, possible to schedule at each round. This forces the switch to satisfy all sources (resp. destinations) from its left (resp. right) subtree then change its configuration to satisfy sources (resp. destinations) from its right (resp. left) subtree. This reduces the number of configuration changes and consequently reducing the power requirement of the schedule.

In the next section we describe the CST interconnect and and introduce some definitions. Section 3 gives the configuration and scheduling algorithm. Section 3.1 shows that the algorithm indeed establish the required communications. In Section 4 we show that our method is time optimal. section 5 proves that the proposed algorithm is power optimal. In Section 6 we summarize our results and make some concluding remarks.

2 CST Structure and Configuration

In this section, we first describe the structure of the CST in a little more detail. We then show the general idea of configuring the CST using local data to establish dedicated paths between source PEs and their corresponding destination PEs.

The three sided switch used in the CST and shown in Figure 3(a) has three data inputs, $\{l_i, r_i, p_i\}$, coming from the left child, right child and the parent respectively. Also it has three data outputs, $\{l_o, r_o, p_o\}$, going to the left child, right child and the parent respectively. An input at a certain side can be connected to any output of the other two sides. It cannot be connected to the output of the same side. This guarantees that a path from a source PE to a destination PE cannot traverse more than $O(\log N)$ switches. This trans-

lates to a single clock cycle transfer of information [3, 7] between PEs. The CST is reconfigurable in the sense that its switches can be reconfigured to connect different PEs at different rounds of the schedule.

2.1 Well Nested Communication Sets

In this paper we consider a special class of communication sets called well nested sets. Moreover, we consider right oriented sets in which each communication has its source to the left of its destination. Any set can be decomposed into two sets each of them is oriented. Dealing with right oriented sets can be adjusted easily to left oriented sets. In a *well-nested* communication set, the communications corresponds to a balanced well-nested parenthesis expression. Figure 2 shows an example of a well-nested set.

Definition 1 Let $\mathcal{T}(u)$ be a subtree rooted at u and let $S_M(u) = \{(s_1, d_1), (s_2, d_2), .., (s_M, d_M)\}$ be a right oriented well-nested set of M communications matched at switch u. The outer most communication at switch u, $O_c(u) = (s_i, d_i) \in S_M(u)$ is the one such that $s_i \leq s_j$, $0 \leq j \leq M$ and $d_i \geq d_j$, $0 \leq j \leq M$.

Figure 3(b) shows two communications c_3 and c_4 matched at switch u. Here $O_c(u) = c_4$ since its source s_4 is to the left of s_3 and its destination d_4 is to the right of d_3. Note that the source of $O_c(u)$ need not be the most left source in $\mathcal{T}(u)$ (see Figure 3(b)). Similarly, the destination of $O_c(u)$ need not be the most right destination in $\mathcal{T}(u)$. ■

Definition 2 Let $S(u) = \{s_1, s_2, .., s_n\}$ (resp. $D(u) = \{d_1, d_2, .., d_n\}$) be a set of n sources (resp. destinations) in $\mathcal{T}(u)$ matched at u or upper level switch. The x^{th} left (resp. right) most source $S_u(x)$ (resp. destination $D_u(x)$) is the one that has x sources (resp. destinations) in $S(u)$ (resp. $D(u)$) to its left (resp. right).

In Figure 3(b), $S(u) = \{s_1, s_3, s_4, s_6, s_7\}$. $S_u(2)$ is s_4 because it has two sources,s_6, s_7, in $S(u)$ to its left. Also, $D(u) = \{d_3, d_4\}$. $D_u(0)$=d_4 and $D_u(1)$=d_3 because it has one destination, d_4 to its right. ■

In the algorithm presented in Section 3, if switch u is to schedule a matched communication at u, then it selects $O_c(u) = c_4$ for scheduling. This corresponds to connecting $S_u(2)$ to $D_u(0)$.

2.2 CST Configuration

Configuration of the CST boils down to configuring its switches; the part of the CST that is reconfigurable. Two PEs can communicate by setting a path from one PE (the source) to the the other PE (the destination.) This requires configuring all the switches in the path from the source PE

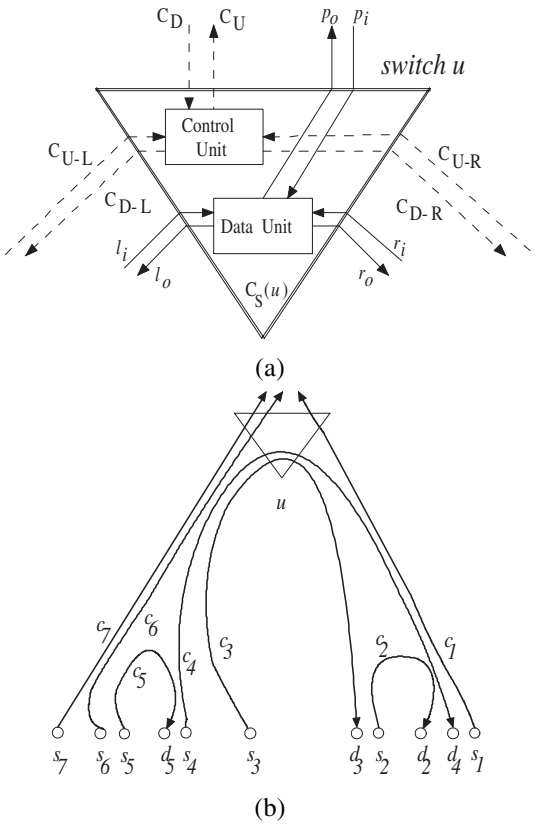

Figure 3. (a) Switch Structure. (b) Illustration of Definitions 1, 2.

to the destination PE to establish the required path. The switch (see Figure 3(a)) contains a data unit and a control unit. Configuring the switches is done through sending control information up the tree starting from the leaves. Each PE knows whether it is a source, a destination, or neither; a local information at each PE. This information flows up the tree. The switches (control unit) at internal nodes receives control information,C_{U-L} and C_{U-R}, from both children, perform some computation, store control information C_S and send up other control information, C_U to its parent; the details of received, stored, and sent control information will be detailed in Section 3. When the root receives the control information, it sends down the tree some new control information, C_{D-L} (resp. C_{D-R}), down to its left (resp. right) child. This information is used at switches along with the stored control information C_S to configure its internal paths.

The control unit use C_S and C_{D-L} or C_{D-R} to configure the data unit to establish the paths required by the algorithm.

Lemma 1 *For right oriented well-nested communication set, if a switch u receives control information, C_s, from a source PE, s, from its left child and receives control infor-*

mation, C_d, from a destination PE, d, from its right child, then s and d represent a matching source-destination pair.

Proof: Let C_s and C_d be received at u from its leftchild and right child respectively. Assume that they dont represent a matching source-destination pair, then s has its destination to the right of d (the set is right oriened) and d has its source to the left of s which contradicts the assumption of well nested communication. ■

2.3 Power Modeling

The three sided switch used in the CST has three inputs and three outputs. In this work, if the switch connects an input to an output, then it consumes one unit of power. This means that, in a step, if the switch changes configuration, it will need at most three units of power since it has three connections it can set.

In this work we consider scheduling a set of oriented well-nested communication set on the CST. If the set is of width w, then w rounds will be needed for the schedule to finish. At each round a number of communications that are compatible are selected, the switches are configured, and the data is transferred. The selection of the communications at each round is done in such a way that the switches dont change configuration until all communication requirements from that configuration is satisfied. In Section 5 we show that each switch requires only a constant number of switch changes for all the communications to be performed.

3 The Configuration and Scheduling Algorithm (CSA)

In this section we present our algorithm for power-aware scheduling and performing a set of oriented well-nested communication set of width w. The algorithm is composed of two phases, each with a number of steps. The first phase is executed only once. The second phase repeats w rounds to ensure all communications in the set are performed.

The main idea of making the algorithm power aware is that each switch, u, select the *most outer* communication, $O_c(u)$ possible to schedule at each round. This forces the switch to satisfy all sources (resp. destinations) from its left (resp. right) subtree then change its configuration to satisfy sources (resp. destinations) from its right (resp. left) subtree. This reduces the number of configuration changes and consequently reducing the power requirement of the schedule. The switch configures the outer most communication in each round based on the control information received at the switch while going up the tree, C_S, and while going down the tree, C_{D-L} or C_{D-R} depending on whether the switch is left child or right child respectively. All the switches contribute to configuring the most outer communication which is done by selecting the source and destination of $O_c(u)$.

- *Phase 1: Distributing Control Information*

- Step 1.1: Each PE sends to its parent whether it is a source $[1, 0]$, a destination $[0, 1]$, or neither $[0, 0]$.
- Step 1.2: Each internal switch, u, receives from its left child y (resp., right child z) control information $C_{U-L} = [S_L, D_L]$ (resp., $C_{U-R} = [S_R, D_R]$). S_L (resp., D_L) is number of communications that require to use the link from y to u (resp. from u to y). S_R (resp., D_R) is number of communications that require to use the link from z to u (resp. from u to z).
- Step 1.3: Each switch u matches sources from left subtree and destinations from right subtree and stores $C_S = [M, S_L - min(S_L, M), D_L, S_R, D_R - min(D_R, M)]$ where M is the number of matched source-destination pairs at u. Then, each switch u sends $C_U = [S_L - min(S_L, M) + S_R, D_L + D_R - min(D_R, M)]$ to its parent. The information flows up the tree until it reaches the root.

- *Phase 2: Configuration & Scheduling*

- Step 2.1: Starting from the root, each switch u sends control information $C_{D-L} = [C_{D-L_1}, x_{s_l}, x_{d_l}]$ (resp. $C_{D-R} = [C_{D-R_1}, x_{s_r}, x_{d_r}]$) to its left child y (resp. right child z). C_{D-L_1} and C_{D-R_1} could be either $[s, null]$, $[d, null]$, $[s, d]$, or $[null, null]$. The information C_{D-L} (resp., C_{D-R}) from u to y (resp., z) tells y (resp., z) which link(s) from u to y (resp., z) will be used in the current round. The argument x_{s_l} (resp. x_{s_r}) tells y (resp. z) to connect to ${x_{s_l}}^{th}$ (resp. ${x_{s_r}}^{th}$) source, $S_y(x_{s_l})$ (resp. $S_z(x_{s_r})$).
  Similarly, x_{d_l} (resp. x_{d_r}) tells y (resp. z) to connect to ${x_{d_l}}^{th}$ (resp. ${x_{d_r}}^{th}$) destination, $D_y(x_{d_l})$ (resp. $D_z(x_{d_r})$). Consequently y (resp., z) uses C_{D-R} (resp. C_{D-L}) along with the information obtained from Phase 1 to set its path such that the outer most possible communication is established. After configuring themselves, switches y and z update the stored information at each of them, $C_S(y)$ and $C_S(z)$. All PEs that receive either $[s, null]$ or $[d, null]$ will participate in the current round of scheduling.
- Step 2.2: PEs received $[s, null]$ write their data to their destinations.
- Step 2.3: Repeat Phase 2 till all the communications are scheduled in some round. ■

Now we explain the steps of the algorithm in detail. The algorithm consists of two phases. The first phase is the distributing control information phase in which the local data at the PEs is distributed to switches of the CST. Phase 1 executes only once. At the end of this phase, each switch u has all the information about the communications that use u in order to be established.

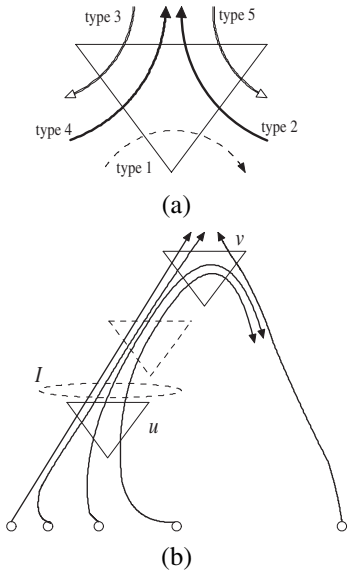

Figure 4. (a) Classification of communications. (b) Illustration of the proof of Theorem 5.

In Step 1.1 each PE sends to its parent whether it is a source, a destination or neither. This information is represented as $[1, 0]$, $[0, 1]$, or $[0, 0]$ respectively.

In Step 1.2, each switch u receives from y (resp. z), C_{U-L} (resp. C_{U_R}), where $C_{U_L} = [S_L, D_L]$ (resp.$C_{U_R} = [S_R, D_R]$). S_L (resp. S_R) is the number of sources that are in the subtree rooted at y (resp. z) and require to use the switch u to connect with their corresponding destinations. Similarily, D_L (resp. D_R) is the number of destinations that are in the subtree rooted at y (resp. z) and require to use the switch u to connect with their corresponding sources. This matches with data in Step 1.1 where $[1, 0]$ means one source and no destination and $[0, 1]$ means one destination and no source. While $[0, 0]$ means no sources and no destinations.

In Step 1.3, each switch u classifies the communications that pass through it into five types (see Figure 4(a)). It checks if there are matching source-destination pairs at u (type 1). By Lemma 1, for well-nested sets, if $S_L = X$ and $D_R = Y$ then there are $M = min(X, Y)$ matching pairs at u. Switch u stores this information, M, along with the following information: S_R (type 2), D_L (type 3), $S_L - min(S_L, M)$: the unmatched sources from the left child (type 4), and $D_R - min(D_R, M)$: the unmatched destinations from the right child (type 5).

It should be pointed out that a switch u knows only how many matching pairs M of type 1 at u since all type 1 require the same configuration $l_i \rightarrow r_o$. It is not important to know which source matches which destination. This also applies to other types of communications as well.

By the end of Phase 1, switches can configure itself with the proper configuration if it knows which type of communication is being configured. This information becomes available in the second phase of the algorithm.

The second phase is responsible of configuring the CST and scheduling the communication set in some rounds. In other words the phase is repeated w rounds. Each round selects some communications to be configured and performed. The selection procedure as will be described is the main factor in making the algorithm power aware. Also, the configuration process takes advantage of the well-nested sets properties to guarantee that correct paths are established.

The main idea of the selection procedure is the following. Assume that the root switch, r, has M matching source-destination pairs. All these communications are of type 1 and require the same configuration, $l_i \rightarrow r_o$. The algorithm selects $O_c(r)$ to be scheduled first. Control information is sent to the children carrying the communication requirements from each child. In other words, each switch u configures itself to establish the path for the outer most communications defined by upper level switches along with outer most communication matched at u, $O_c(u)$, (if any) if possible. Because the communications are well-nested, the outer most communication translates to connecting the source of $O_c(u)$ to the destination of $O_c(u)$. These are defined as the ${x_{s_l}}^{th}$ or ${x_{s_r}}^{th}$ left source and the ${x_{d_l}}^{th}$ or ${x_{d_r}}^{th}$ most right destination (see Definition 1.)

In Step 2.1, each switch u sends to its children y and z the communication requirement at this round of scheduling. In other words, u tells y (same for z) which links (upward or downward or both) from u to y will be used in the current round. This triggers y to set the proper internal path. If upward link is to be used (represented by $[s, null]$) then y should configure either $l_i \rightarrow p_o$ or $r_i \rightarrow p_o$ paths based on the availability of sources from left or right. For $O_c(u)$, the priority of the selection is for $l_i \rightarrow p_o$ (if any). Similarly, if downward link is to be used (represented by $[d, null]$) then y should configure either $p_i \rightarrow r_o$ or $p_i \rightarrow l_o$ paths based on the availability of destinations from right or left. For $O_c(u)$, the priority of the selection is for $p_i \rightarrow r_o$ (if any). If both links are to be used represented by $[s, d]$, then y should configure itself to connect the ${x_{s_l}}^{th}$ most left source along with the ${x_{d_l}}^{th}$ most right destination. After the switch configures its internal path, it updates the values in C_S. For example, if switch u schedules a matching communication at u, then the value of M in $C_S(u)$ is decremented by 1. A pseudocode for Step 2.1 is shown in Figure 5 for the cases $[null, null]$ and $[s, null]$. The cases for $[d, null]$ and $[s, d]$ are similar and omitted here for shortage of space. By the end of Step 2.1, each PE receives either $[s, null]$, or $[d, null]$, or $[null, null]$.

In Step 2.2 each source PE chosen in the current round writes its data to its corresponding destination. Step 2.3 guarantees that all communications are performed.

Procedure CONFIGURE($u, C_S(u), C_D(u), C_{D-L}(u), C_{D-R}(u)$)
/* The procedure configures the switch u with a correct configuration where $C_S(u)$ is control information stored at u in Phase 1, $C_D(u)$ is the information received in Step 2.1, $C_{D-L}(u)$ (resp. $C_{D-R}(u)$) is the information generated to the left (resp. right) child */

Begin

If $C_D(u) = [null, null,]$ then /* [null,null] received*/
  if $M \neq 0$ then /* in Step 2.1 */
    connect $l_i \rightarrow r_o$
    $M = M - 1$
    if $S_L - min(S_L, M) \neq 0$
      $x_{s_l} = S_L - min(S_L, M)$
    endif
    if $D_R - min(D_R, M) \neq 0$
      $x_{d_r} = D_R - min(D_R, M)$
    endif
    $C_{D-L}(u) = [s, null, x_{s_l}, 0]$
    $C_{D-R}(u) = [d, null, 0, x_{d_r}]$
  endif
else if $C_D(u) = [s, null]$ then /* [s,null] received */
  if $S_L - min(S_L, M) > x_s$ then /* in Step 2.1 */
    connect $l_i \rightarrow p_o$
    $S_L - min(S_L, M) = S_L - min(S_L, M) - 1$
    $C_{D-L}(u) = [s, null, x_s, 0]$
    $C_{D-R}(u) = [null, null, 0, 0]$
  else
    connect $r_i \rightarrow p_o$
    $S_R = S_R - 1$
    $x_{s_r} = x_s - (S_L - min(S_L, M))$
    $C_{D-L}(u) = [null, null, 0, 0]$
    $C_{D-R}(u) = [s, null, x_s, 0]$
    if $M \neq 0$ then
      connect $l_i \rightarrow r_o$
      $M = M - 1$
      $x - s_l = S_L - min(S_L, M)$
      $C_{D-L}(u) = [s, null, x_{s_l}, 0]$
      $C_{D-R}(u) = [s, d, x_{s_r}, 0]$
    endif
  endif
endif

End

Figure 5. Pseudocode for Step 2.1 in Phase 2 of the algorithm.

3.1 Correctness of The Algorithm

In this section, we show that the algorithm establishes a path between each source and its corresponding destinations in some round.

Lemma 2 *For right oriented well-nested communication set, if a switch receives control information $[s, d]$ in Phase 2 of the algorithm, that corresponds to two communications matched at switches u and v, then, the algorithm establish the connections for source and the destination of the most outer communications matched at v and u respectively.*

Proof: Since the communication set are right oriented well-nested set, then the source of one communication is to the right of the second communication. If $[s, d]$ is received at a switch, then the algorithm connects the ${x_s}^{th}$ most left source (source of $O_c(u)$) and the ${x_d}^{th}$ most right destination (destination of $O_c(v)$). This connects the required source and the required destination since the source is to the right of the destination. Moreover, by Definition 1 the source for $O_c(v)$ is the ${x_s}^{th}$ most left source in $\mathcal{T}(v)$. Similarly, the destination for $O_c(u)$ is the ${x_d}^{th}$ most right source in $\mathcal{T}(u)$. Thus the algorithm establishes the connections for source and the destination for $O_c(v)$ and $O_c(u)$ respectively. ■

Lemma 3 *If at some round of the algorithm, switch u schedules a matching communication at u, then the algorithm correctly, establishes the outer most communication, $O_c(u)$, matched at u.*

Proof: Assume that switch u has M matching communications. At some round of the algorithm, u schedules a matched communication at u. Here, we show that the algorithm selects to establish the outer most communication, $O_c(u)$, at u. First, switch u connect $l_i \rightarrow r_o$ and sends control information to its children. All the switches at lower levels than u (if involved in the required path) select to connect the ${x_s}^{th}$ most left source matched at u, $S(O_c(u))$ and the ${x_d}^{th}$ most right destination matched at u, $D(O_c(u))$. By Definition 1 this establishes the path for $O_c(u)$.

In Step 2.1, switch u sends to its children y and z the communication requirement at this round. In other words, u tells y (same for z) which links from u to y will be used in this round. This triggers y to set the proper internal path. Let h be a lower level switch and let C be the control information received by h from its parent. Four cases exist;

- *Case 1:* If $C_{D-L_1} = [null, null]$, then h is not involved in establishing $O_c(u)$.
- *Case 2:* If $C_{D-L_1} = [s, null]$, then h configures either $l_i \rightarrow p_o$ or $r_i \rightarrow p_o$ paths based on the availability of sources from left or right (This information is known priori from Phase 1.) For $O_c(u)$, the priority of the selection is for $l_i \rightarrow p_o$ (if any).

- *Case 3:* If $C_{D-L_1} = [d, null]$, then h configures either $p_i \rightarrow r_o$ or $i_i \rightarrow l_o$ paths based on the availability of destinations from right or left (This information is known priori from Phase 1.) For $O_c(u)$, the priority of the selection is for $p_i \rightarrow r_o$ (if any).
- *Case 4:* If $C_{D-L_1} = [s, d]$, then by Lemma 2, h configures to connect the ${x_s}^{th}$ most left source along with the ${x_d}^{th}$ most right destination. By Definition 1 this corresponds to the most outer communications that are represented by $[s, d]$.

All the cases establish a connection for $S(O_c(u))$ and $D(O_c(u))$ thus establishing the communication $O_c(u)$. ■

Since in each round of the algorithm, the algorithm selects to establish at least one communication, then in a number of rounds the algorithm correctly performs all the communications in the set. We have the following theorem.

Theorem 4 *The algorithm correctly establishes dedicated paths between each source and its corresponding destination in some round.*

Proof: Since each communication in the set is matched at some switch. Then when the algorithm schedules a communication, by Lemma 3, it correcly establishes this communication. Since each communication is scheduled at some round, then the algorithm correctly establishes all the communications in the set at some rounds. ■

Theorem 4 shows that the algorithm correctly establishes the paths for some communications in some round. Repeating the process for a number of rounds establishes all the communications. For the oriented well-nested sets the number of rounds is optimal. We prove that in the next section.

4 Proof of Optimality

Here, we show that for width-w oriented well nested sets, the algorithm in Section 3 routes all communications in exactly w rounds. A width-w communication set translates to having sets of w sources (or destinations) that use the same link in the same direction. Such sets are called maximum incompatibles. To prove that the algorithm is optimal, it is sufficient to show that it schedules one source and destination from each maximum incompatible in each round.

Assume that a set of w communications, I, uses the link from a switch u to an upper level switch v. In Phase 2, if v received from its parent $[null, null]$, then v schedules $O_c(v)$ (see Figure 4(b)) thus reducing the size of I.

If v received $[s, null]$ from its parent then v schedules the communication corresponding to the ${x_{d_l}}^{th}$ or ${x_{d_r}}^{th}$ source. If this source is in the left subtree of v then I is reduced by 1. If this source is in the right subtree of v then, v schedules also a matching communication at v reducing I by 1.

If v received $[s, d]$ from its parent then v schedules the communication corresponding to ${x_{d_l}}^{th}$ or ${x_{d_r}}^{th}$ source. If this source is in the left subtree of v then I is reduced by 1. If this source is in the right subtree of v and the destination is in the left subtree of v then v schedules also a matching communication at v reducing I by 1. If this source is in the right subtree of v and the destination is in the right subtree of v then I is not a maximal incompatible.

Similar argument holds for destination incompatibles and thus proving the optimality of the algorithm. It is clear that the algorithm is efficient. Each switch stores a constant number of words in Phase 1. Also it passes to its parent a constant number of words. In Phase 2, each switch passes to its children a constant number of words and performs constant time computations. Thus we have the following result.

Theorem 5 *The algorithm routes well-nested communications optimally on the CST. Also, each switch stores a constant number of words, passes to its neighbors a constant number of words, and performs a constant-time computations.* ■

5 Proof of Power-Awareness

In this section we show that our algorithm is power optimal. We show that each switch requires $O(1)$ configuration changes during all the w rounds of the algorithms. Consequently, a constant number of power units is needed for each switch. This is in contrast to the algorithm given in [6] in which each switch requires $O(w)$ configuration changes.

In this proof we deal with the control information in Phase 2 that requires certain switch settings. Without loss of generality, the proof handles the information sent to the left child (right child can be handled in the same way). Assume that, in Phase 2, a switch u sends to its left child, y, $C_{D-L} = [C_{D-L_1}, x_{s_l}, x_{d_l}]$. C_{D-L_1} could be either $[null, null]$, $[s, null]$, $[d, null]$, $[s, d]$. Here also we deal with values of C_{D-L_1} that contain s or $null$ ($[null, null]$, $[s, null]$). Dealing with values that contain d is similar.

We first show that if a switch y receives either $[null, null]$ or $[s, null]$ then, these values can alternate at most only twice. In other words, y could receive a subsequence of $[null, null]$ in subsequent rounds then a subsequence of $[s, null]$ in some following rounds then other subsequence of $[null, null]$. Or, y could receive a subsequence of $[s, null]$ in subsequent rounds then a subsequence of $[null, null]$ in some following rounds then other subsequence of $[s, null]$. Call the first (resp. second) sequence $\mathcal{Q}_1$ (resp. $\mathcal{Q}_2$). Note that C_{D-L_1} only alternate twice in $\mathcal{Q}_1$ and $\mathcal{Q}_2$. We show that if this happens, switch y needs only a constant number of switch changes. Then we show that in Phase 2 of the algorithm, each switch only generates to its children the first or the second sequence.

Lemma 6 *Let $\mathcal{Q}_1$ be a sequence composed of a subsequence of $[null, null]$ followed by a subsequence of $[s, null]$ followed by a subsequence of $[null, null]$. Let $\mathcal{Q}_2$ be a sequence composed of a subsequence of $[s, null]$ followed by a subsequence of $[null, null]$ followed by a subsequence of $[s, null]$. In Phase 2, if switch y receives $\mathcal{Q}_1$ or $\mathcal{Q}_2$, then y makes only a constant number of switch changes.*

Proof: Assume y receives $\mathcal{Q}_1$, then, because of the subsequence $[null, null]$, y connects $l_i \rightarrow r_o$ to the end of the subsequence $[null, null]$ (if $\exists$ any matching communications at y). When $[s, null]$ is received, then y connects $l_i \rightarrow p_o$ or $r_i \rightarrow p_o$ whatever is available till the end of subsequent $[s, null]$. When $[null, null]$ is received, then y connects $l_i \rightarrow r_o$ to the end of the subsequence $[null, null]$. Thus $\mathcal{Q}_1$ requires only a constant number of switch changes. A similar argument can be said about $\mathcal{Q}_2$. ■

Lemma 7 *In Phase 2, of the algorithm, and for the source requirements, each switch only generates either $\mathcal{Q}_1$ or $\mathcal{Q}_2$ till the end of the algorithm.*

Proof: Assume only source requirement ($C_{D-L_1} = [s, null]$ or $[null, null]$). Assume the leaves of the tree is at level 0 and the root of the tree is at level $\ell og\ n$. We proceed by the induction on the switch level. The root switch, r, represents the base case. The root schedules all the M communications passing though it in the first M rounds; i.e. it sends to its left child $[s, null]$ in the first M rounds. Then r sends $[null, null]$ to its left child through the rest of the algorithm. In other words the sequence alternate only once. Assume the lemma to hold for a switch u at level k and consider its left child y (right child z is similar) at level $k-1$.

Assume that switch u receives $\mathcal{Q}_1(u)$. Because of the $[null, null]$, u connects $l_i \rightarrow r_o$ and generates $[s, null]$ to its left child to the end of the received subsequence $[null, null]$. When $[s, null]$ is received at u, then u connects $l_i \rightarrow p_o$ and generates $[s, null]$ to its left child or connects $r_i \rightarrow p_o$ and generates $[null, null]$ to its left child till the end of subsequence $[s, null]$. When $[null, null]$ is received, then u connects $l_i \rightarrow p_o$ (for any matching communication still to be scheduled) and generates $[s, null]$ to its left child or disconnect and and generates $[null, null]$ to its left child. It is clear that u, at most, generates either subsequence of $[null, null]$ followed by subsequence of $[s, null]$ followed by a subsequence of $[null, null]$; i.e. generating $\mathcal{Q}_1(u)$. Similar argument holds if u receives $\mathcal{Q}_2(u)$. This proves that each switch generates a sequence of control information that can alternate at most twice proving the lemma. ■

Similar argument holds for the existence of destination requirement, d, in the value of C_{D-L_1}. The existence of s and d in C_{D-L_1} does not pose a problem since they use different links. Thus we have the following result.

Theorem 8 *The algorithm requires each switch to change configuration $O(1)$ times and thus requires a constant number of power units.* ■

6 Concluding Remarks

In this paper, we have introduced a new technique called Power-Aware Dynamic Reconfiguration (PADR). We have used the idea behind this technique to develop a configuration and scheduling algorithm for a set of well-nested communications on the CST. We have shown that each switch requires only $O(1)$ power units to schedule all the communications in the set. This is far lower than the $O(w)$ power units required by each switch in previous algorithms. The algorithm is optimal; a width-w communication set requires w rounds for scheduling. We have shown that the algorithm also is efficient; a constant number of words is stored at each switch and a constant number of words is transferred between neighboring switches.

One obvious extension to this work is the study of other communication patterns on the CST and on other reconfigurable interconnect. Other extensions include using the PADR technique to develop computational algorithms for reconfigurable models and architectures.

References

[1] L. Benini and G.De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, Jan. 2002, pp. 70-78.

[2] K. Bondalapati and V. K. Prasanna, "Hardware Object Selection for Mapping Loops onto Reconfigurable Architectures," *Proc. Int'l. Conf. Par.& Distr. Proc. Tech. and App.*, 1999.

[3] H. M. El-Boghdadi, R. Vaidyanathan, J. L. Trahan and S. Rai, "On the Communication Capability of the Self-Reconfigurable Gate Array Architecture," *Procs. 9th Reconfigurable Architectures Workshop (RAW 2002)*, included in Procs. 16^{th} Int. Parallel & Distributed Proc. Symp., Florida, April 15, 2002.

[4] X.Y.Lin, Y.C.Chung, and T.Y. Haung, "A Multiple LID Routing Scheme for Fat-Tree Based Infiniband Networks," *Proc. Int. Parallel and Distrib. Proc. Symp.*, 2004.

[5] K. Nakano, "A Bibliography of Published Papers on Dynamically Reconfigurable Architectures," *Parallel Proc. Letters*, vol. 5, 1995, pp. 111–124.

[6] K. Roy, R. Vaidyanathan and J. L. Trahan, "Routing Multiple Width Communications on the Circuit Switched Tree," *International Journal of Foundations of Computer Science*, vol. 17, No. 2, April 2006, pp. 271-285.

[7] R. Sidhu, S. Wadhwa, A. Mei, and V. K. Prasanna, "A Self-Reconfigurable Gate Array Architecture," *Int. Conf. on Field Programmable Logic and Applications*, 2000, Springer Verlag Lecture Notes in Computer Sc., vol. 1896, pp. 106–120.