

# Level-wise Scheduling Algorithm for Fat Tree Interconnection Networks

Zhu Ding\*

Union Switch & Signal

Raymond R. Hoare†

Concurrent EDA, LLC

Alex K. Jones‡

University of Pittsburgh

Rami Melhem§

University of Pittsburgh

## Abstract

This paper presents an efficient hardware architecture for scheduling connections on a fat-tree interconnection network for parallel computing systems. Our technique utilizes global routing information to select upward routing paths so that most conflicts can be resolved. Thus, more connections can be successfully scheduled compared with a local scheduler. As a result of applying our technique to two-level, three-level and four-level fat-tree interconnection networks of various sizes in the range of 64 to 4096 nodes, we observe that the improvement of *schedulability ratio* averages 30% compared with greedy or random local scheduling. Our technique is also scalable and shows increased benefits for large system sizes.

## 1 Introduction

Fat-tree interconnection networks have several good features, including scalability and simple topology. This network architecture was first proposed by Leiserson [1] as a hardware efficient and general purpose interconnection network. It has the structure of a tree and its links have different bandwidth at each level. The closer a link is to the root, the larger the bandwidth. In a fat-tree network, all processing elements are located at the leaf nodes while the paths are routed through intermediate switching nodes. It has been proven to be an area-universal interconnection network [2]. Fat-tree interconnect networks can simulate any other network topology with the same silicon area with at most poly-logarithmic slowdown [2]. Whereas, other network topologies, including 2-D arrays and simple trees, will see polynomial slowdown when simulating other networks. Due to those advantages, the fat-tree topology has become a popular network architecture for massively parallel computing systems such as the Thinking Machine CM-5 [2, 3], Meiko

supercomputer CS-2 [4], COMPAQ AlphaServer SC [5] and Quadrics QsNetII [6].

By convention, the scheduling approaches for fat-tree interconnection networks are developed for store and forward and wormhole routing. Therefore, almost all of the scheduling algorithms are based on the local knowledge within switch nodes. Adaptive distributed scheduling is a widely used scheme for scheduling communications in fat-trees [7, 8]. In this scheme, each switch selects a routing path randomly from the available local ports. Based on this scheme, heuristic routing algorithms are developed, such as Turn Back When Possible (TBWP) algorithm proposed by Kariniemi and Nurmi [9]. They suggest connecting the top-most switches together. This allows packets that are blocked through contention to continue up the tree toward the root and to select another path that is available.

Scheduling approaches that use local information provide good scalability. However, the success of routing in a local switch node does not imply the success of routing in the entire network. We define the *schedulability ratio* to be the number of successful connections divided by the number of total requests. It is often true that when scheduling with local routing information, the effort put into scheduling one switch node does not help improve the schedulability ratio of the entire network. Schedulability ratio impacts bandwidth utilization. If the scheduling ratio is far below optimal, the bandwidth utilization is inefficient. The penalty of low bandwidth utilization detrimentally impacts execution time, especially for long-lived connections.

To solve the problem of maximizing the schedulability ratio in fat-tree interconnection networks, we make a few key observations about the structure of a fat-tree network and develop the Level-wise Fat Tree scheduling algorithm. Specifically, we observe that all switch configuration options are set once a message reaches the top of the fat tree; that is to say that there is only one path between the top of a fat tree and its destination. We also observe that the upward path from the source switch to the top of the fat tree is symmetrical with the downward path from the top of the fat tree to the destination. The Level-wise Fat Tree scheduling algorithm leverages this symmetry at each level to allocate

---

\*e-mail:zhuding@connecttime.net

†e-mail:rayhoare@concurrenteda.com

‡email:akjones@ece.pitt.edu

§email:melhem@cs.pitt.edu

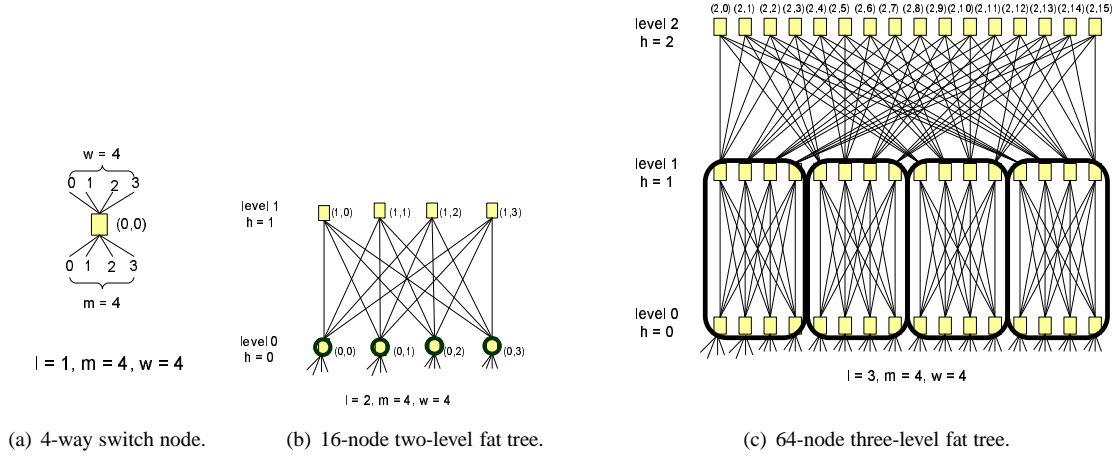


Figure 1: Fat-tree construction.

both the upward path and downward path simultaneously. Using only local switch information, our simulation results show the schedulability ratio to be 45%-70%, depending on the size and depth of the fat-tree. The Level-wise Fat Tree scheduling algorithm is able to provide a schedulability ratio of 78%-95%.

The rest of the paper is organized as follows. Section 1 provides a background on fat trees and presents formal notation that will be used to describe our algorithm. Key observations and theorems are also presented in Section 3. Section 4 describes the Level-wise fat-tree scheduling algorithm in detail. Section 5 shows the system simulation results. FPGA-based hardware that efficiently implements the Level-wise algorithm is introduced in Section 6. Conclusions are offered in Section 7.

## 2 Background

A fat-tree interconnected network is denoted by  $FT(l, m, w)$ , where  $l$  is the number of levels in the tree,  $m$  is the number of children in each switch node and  $w$  is the number of parents of each switch node. Fat-tree  $FT(l, m, w)$  has  $l$  levels of switches. Each level  $h$  has  $w^{l-1}$  switch nodes. Each switch is labeled by  $SW(h, \tau)$ , where  $h = 0, 1, \dots, l-1$  and  $\tau = 0, 1, \dots, w^{l-1} - 1$ . We use a bit vector to represent  $\tau$  in order to more clearly describe algorithms later.  $t_{l-2}t_{l-3} \dots t_0$  is the base- $w$  representation of the integer  $\tau$ ,  $0 \leq \tau < w^{l-1}$ .

$$\text{Let } t_i = \tau \text{ div } w^i, \text{ then } \tau = \sum_{i=0}^{l-2} t_i w^i$$

In general fat tree interconnection networks, each switch node is symmetrical, therefore  $m$  equals  $w$ . The symmetrical fat tree is denoted as  $FT(l, w)$ , which is the assumption for the proof of the Level-wise scheduling algorithm. However, the algorithm is also applicable when  $m$  and  $w$  are not equal.

## 3 Fat tree construction

The fat-tree architecture is constructed recursively as introduced by Ohring [16]. Let  $FT(l, w)$  be a fat-tree with  $l$  levels, and each switch node has  $w$  children and  $w$  parents.  $FT(l+1, w)$  is built from  $w$  copies of  $FT(l, w)$  and  $w^l$  additional switch nodes. The top switch nodes in each copy of  $FT(l, w)$  are connected with the  $w^l$  additional switch nodes. The  $SW(l, \tau)$  is connected with  $SW(l+1, (\tau \times w) \bmod w^l)$ ,  $SW(l+1, (\tau \times w) \bmod w^l + 1)$ ,  $\dots$ ,  $SW(l+1, (\tau \times w) \bmod w^l + w - 1)$ . Figure 1 shows an example 64-node fat tree where  $w = m = 4$  (Figure 1(c)) built up from 16-node two-level fat trees (Figure 1(b)) constructed from 4-node single level switch nodes (Figure 1(a)).

Each switch node has  $w$  bi-directional links connected with the adjacent upper level and  $w$  bi-directional links connected with the adjacent lower level. The selection of ports decide the link between two levels, hence we use  $Ulink(h, \tau, i)$  to represent upward link connected through port  $i$  in switch  $(h, \tau)$  and  $Dlink(h, \tau, i)$  to represent downward link connected through port  $i$  in switch  $(h, \tau)$ , as shown in Figure 2. We use a pair of vectors,  $Ulink(h, \tau)[i]$  and  $Dlink(h, \tau)[i]$ , to represent the availability of upward and downward links, where  $i = 0, 1, \dots, w-1$ . If  $Ulink(h, \tau)[i]$  equals one, upward link connected via port  $i$  of switch  $(h, \tau)$  is available; otherwise, it is occupied. We use  $P_h$  to represent the upper port number selected at level  $h$ . When performing routing inside a switch node, a communication request from the source will be routed upward through one of the upward links until it reaches a switch which is a common ancestor of both the source and destination; then, the request will be routed downward through a downward link toward the destination [7, 8].

**Theorem 1** For switch  $(h, \tau_h)$ ,  $\tau_h = \sum_{i=0}^{l-2} t_i w^i$ , if the ports  $P_h$  for level  $h$  ( $h > 0$ ) is chosen, then the switch at level  $h+1$  that

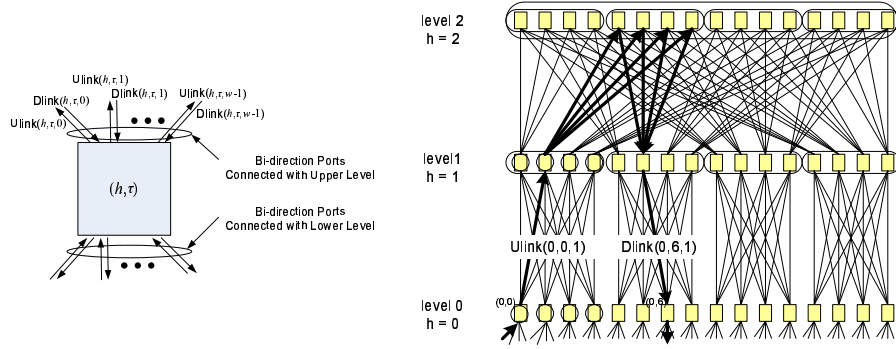


Figure 2: The link selection.

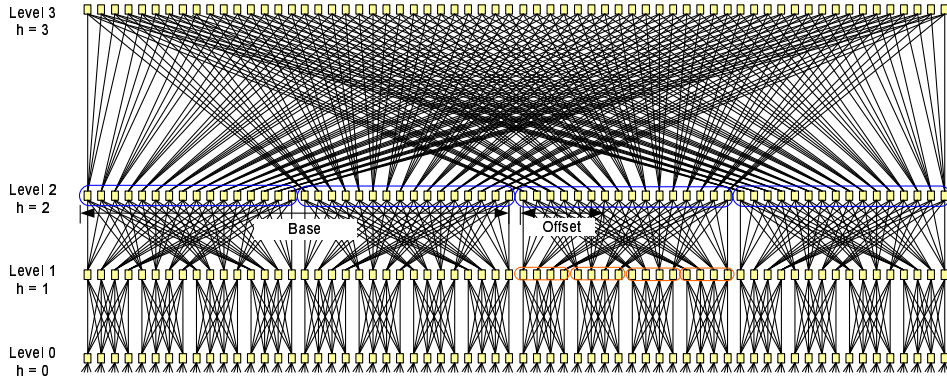


Figure 3: Switch node computation.

connected with switch  $(h, \tau_h)$  is switch  $(h+1, \sum_{i=h+1}^{l-2} t_i w^i + \sum_{i=1}^h t_{i-1} w^i + P_h)$ .

**Proof:**

Assume the switch at level  $h$  is  $SW(h, \tau_h)$ , and the  $SW(h+1, \tau_{h+1})$  at level  $h+1$  is connected with it. According to the fat tree construction rule,  $SW(h, \tau_h)$  is connected with  $SW(h+1, \tau_{h+1})$  within one  $FT(h+1, w)$ . Therefore the value of  $\tau_h$  consists of two parts,  $\Gamma_h$  and  $\Delta_h$ .  $\Gamma_h$  is counted from the most left till the beginning of the  $FT(h+1, w)$ .  $\Delta_h$  is calculated as in the  $FT(h+1, w)$ . The value of  $\Gamma_{h+1}$  is not affected by the selection of  $P_h$ , while the value of  $\Delta_{h+1}$  is determined by  $P_h$ .

$$\tau_h = \Gamma_h + \Delta_h \quad (1)$$

$$\Gamma_h = \tau_h \operatorname{div} w^{h+1}, \Delta_h = \tau_h \bmod w^{h+1} \quad (2)$$

$$\tau_{h+1} = \Gamma_{h+1} + \Delta_{h+1} = \Gamma_h + \Delta_{h+1} \quad (3)$$

According to the fat tree construction rule, we get

$$\begin{aligned} \Delta_{h+1} &= [\Delta_h w + P_h] \bmod w^{h+1} \\ &= \left[ \left( \tau_h \bmod w^{h+1} \right) w + P_h \right] \bmod w^{h+1}. \end{aligned} \quad (4)$$

Therefore

$$\begin{aligned} \tau_{h+1} &= \left( \tau_h \operatorname{div} w^{h+1} \right) w^{h+1} \\ &\quad + \left[ \left( \tau_h \bmod w^{h+1} \right) w + P_h \right] \bmod w^{h+1} \\ &= \sum_{i=h+1}^{l-2} t_i w^i + \left( \sum_{i=0}^h t_i w^{i+1} + P_h \right) \bmod w^{h+1} \\ &= \sum_{i=h+1}^{l-2} t_i w^i + \sum_{i=1}^h t_{i-1} w^i + P_h \end{aligned} \quad (5)$$

**Q.E.D.**

The selection of the upward link determines the downward link. Specifically, it will be proven in Theorem 2 that if a request is routed upward using  $Ulink(h, \sigma_h, P_h)$  at level  $h$ , it will reach its destination switch using  $Dlink(h, \delta_h, P_h)$  at the same level. In the example shown in Figure 2, we show a  $FT(3, 4)$  and consider a communication request from

SW(0,0) to SW(0,6). The communication request can be routed through Ulink(0,0,0), Ulink(0,0,1), Ulink(0,0,2), or Ulink(0,0,3). If  $P_0 = 1$ , then Ulink(0,0,1) is selected at SW(0,0) for upward routing, the request will be routed back to level 0 using the same port number, which is 1, no matter what routing path is selected above level 0. The downward link is Dlink(0,6,1). This phenomenon is explained and proved in Theorem 2.

**Theorem 2** Given a source SW(0,  $\sigma_0$ ) and a destination SW(0,  $\delta_0$ ), assume that the common ancestor of both switches is at level  $H$ , where  $H < l$ . If  $P_0, P_1, \dots, P_{H-1}$  are the upper port numbers selected at level 0, level 1, ... and level  $H-1$  for upward path from SW(0,  $\sigma_0$ ) to SW( $H, \sigma_H$ ), then the backward path from SW( $H, \sigma_H$ ) to SW(0,  $\delta_0$ ) uses the same upper port numbers  $P_0, P_1, \dots, P_{H-1}$ , but in different switches.

**Proof:**

According to theorem 1, as

$$\tau_0 = \sum_{i=0}^{l-2} t_i w^i \quad (6)$$

Let  $h = 0$ , substitute 6 into 5, we obtain

$$\tau_1 = \sum_{i=1}^{l-2} t_i w^i + P_0 \quad (7)$$

Let  $h = 1$ , substitute 7 into 5, we get

$$\begin{aligned} \tau_2 &= \sum_{i=h+1}^{l-2} t_i w^i + ((t_1 w^1 + P_0) w + P_h) \pmod{w^2} \\ &= \sum_{i=2}^{l-2} t_i w^i + P_0 w + P_1 \end{aligned} \quad (8)$$

Similarly, we get

$$\begin{aligned} \tau_{h+1} &= \sum_{i=h+1}^{l-2} t_i w^i + P_0 w^h + P_1 w^{h-1} + \dots + P_h \\ &= \sum_{i=h+1}^{l-2} t_i w^i + \sum_{i=0}^h P_i w^{h-i} \end{aligned} \quad (9)$$

Let SW(0,  $\sigma_0$ ) be the source switch and SW(0,  $\delta_0$ ) be the destination, where  $\sigma_0 = \sum_{i=0}^{l-2} s_i w^i$  and  $\delta_0 = \sum_{i=0}^{l-2} d_i w^i$ . As we assume that the communication ancestor of both switches is at level  $H$ , where  $H < l$ , switch ( $H, \sigma_H$ ) equals switch ( $H, \delta_H$ ).

Since  $\sigma_0 = \sum_{i=0}^{l-2} s_i w^i$ , and  $P_0, P_1, \dots, P_{H-1}$  are the port numbers selected at level 0, level 1, ..., and level  $H-1$  for upward path,

$$\sigma_H = \sum_{i=H}^{l-2} s_i w^i + \sum_{i=0}^{H-1} P_i w^{H-1-i} \quad (10)$$

Assume a message is forwarded upward from ( $0, \delta_0$ ) to ( $H, \sigma_H$ ), where  $\delta_0 = \sum_{i=0}^{l-2} d_i w^i$ . We also assume  $P'_0, P'_1, \dots, P'_{H-1}$  are the port numbers selected at level 0, level 1, ..., and level  $H-1$  for upward path.

$$\delta_H = \sum_{i=H}^{l-2} d_i w^i + \sum_{i=0}^{H-1} P'_i w^{H-1-i}. \quad (11)$$

Because  $\sigma_H = \delta_H$ ,

$$\begin{aligned} \sum_{i=H}^{l-2} s_i w^i + \sum_{i=0}^{H-1} P_i w^{H-1-i} \\ = \sum_{i=H}^{l-2} d_i w^i + \sum_{i=0}^{H-1} P'_i w^{H-1-i}. \end{aligned} \quad (12)$$

We get

$$\sum_{i=H}^{l-2} (s_i - d_i) w^i + \sum_{i=0}^{H-1} (P_i - P'_i) w^{H-1-i} = 0. \quad (13)$$

Therefore  $P_i = P'_i$ , where  $i = 0, 1, \dots, H-1$ . By reversing the path from ( $0, \delta_0$ ) to ( $H, \sigma_H$ ), we get the exclusive backward path from ( $H, \sigma_H$ ) to ( $0, \delta_0$ ). Thus the request from switch ( $0, \sigma_0$ ) to switch ( $0, \delta_0$ ) has to be routed backward using  $P_0, P_1, \dots, P_{H-1}$  ports.

**Q.E.D.**

## 4 Level-wise routing algorithm

By taking advantage of the property of theorem 2, if the upward path is selected properly, then the number of conflicts during scheduling can be reduced. We have shown in Section III that a request will be routed upward and downward by the same port number of upward links at each level, but may use different port switch nodes. A routing conflict occurs when two requests are routed through same physical path. In conventional adaptive routing, the conflicts occur at downward paths, but are caused by improper selection of upward paths. Figure 4 shows an example of this distinction. Assume that SW(0,0) and SW(0,1) both request a connection to SW(0,8). With local routing information, the requests from SW(0,0) and SW(0,1) can be routed upward using Ulink(0,0,0) and Ulink(0,1,0), respectively, as shown in Figure 4(a). The conflict is not detected until both connections are routed back to level 0. In this case, only one of the two requests can be satisfied. However, if conflict can be detected ahead of time, it can be resolved. Since the request from SW(0,0) is forwarded using Ulink(0,0,0), from Theorem 2 it must be routed back to SW(0,8) using Dlink(0,8,0). We modify the global routing information to indicate Dlink(0,8,0) is occupied. When we schedule request from SW(0,1) to SW(0,8), Ulink(0,1,0) cannot be chosen because it will also

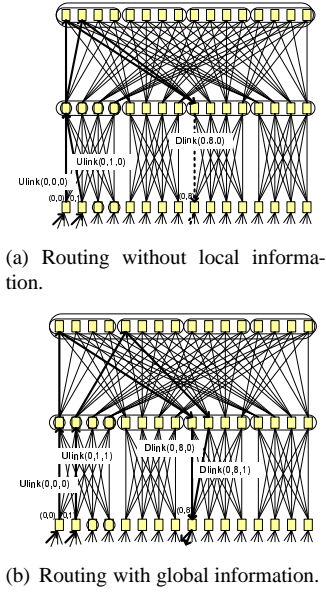


Figure 4: Routing example.

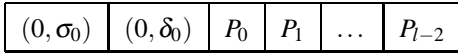


Figure 5: Data structure of a communication request.

need to use  $\text{Dlink}(0, 8, 0)$ , which is occupied. Thus, another link, such as  $\text{Ulink}(0, 1, 1)$  and  $\text{Dlink}(0, 8, 1)$  will be chosen instead as shown in 4(b). By this method, both requests can be granted.

The idea of our centralized routing algorithm is to select upper ports carefully so that the requests can be successfully forwarded both upward and downward at each level with knowledge of the global routing information. A port number is selected only when both the port on the upward path and the port on the downward path are available. For each communication request for a connection between a pair of source node and destination node, we define the data structure shown in Figure 5.

In order to grant a communication request, we need to set a path from a source switch to a destination switch. For instance, a connection from source switch  $(0, \sigma_0)$  to destination switch  $(0, \delta_0)$  is to be established. The common ancestor of both switches is at level  $H$ , where  $H < l$ . In Figure 6, a complete path is set, which is switch  $(0, \sigma_0) \rightarrow \text{Ulink}(0, \sigma_0, P_0) \rightarrow \text{switch}(1, \sigma_1) \rightarrow \text{Ulink}(1, \sigma_1, P_1) \rightarrow \dots \rightarrow \text{switch}(H-1, \sigma_{H-1}) \rightarrow \text{Ulink}(H-1, \sigma_{H-1}, P_{H-1}) \rightarrow \text{switch}(H, \sigma_H) \rightarrow \text{Dlink}(H-1, \delta_{H-1}, P_{H-1}) \rightarrow \text{switch}(H-1, \delta_{H-1}) \rightarrow \dots \rightarrow \text{Dlink}(1, \delta_1, P_1) \rightarrow \text{switch}(1, \delta_1) \rightarrow \text{Dlink}(0, \delta_0, P_0) \rightarrow \text{switch}(0, \delta_0)$ .

By representing all links at a given level,  $h$ , as a single bit-vector, we can perform simple Boolean operations to determine which port is available on both the upward and downward paths. Specifically, the  $w$  bits,  $\text{Ulink}(h, \sigma_h)[i]$ , can be AND-ed with the  $w$ -bits  $\text{Dlink}(h, \delta_h)[i]$ , where  $i =$

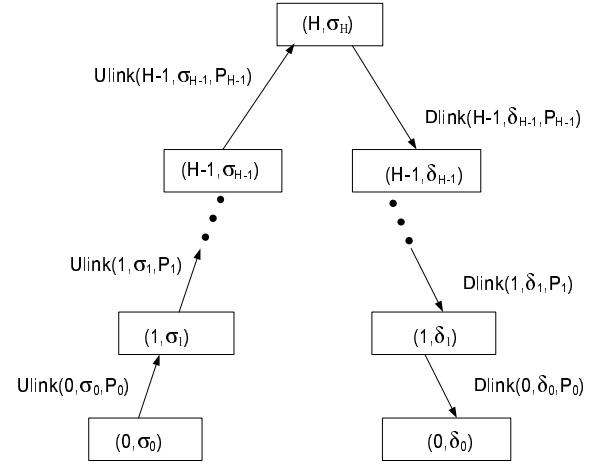


Figure 6: Level-wise scheduling Algorithm.

$0, \dots, w-1$  to form a  $w$ -bit long *available\_port* vector that represents the upward ports from the source switch that do not contain a conflict at level  $h$  in the fat tree.

All source switches at level  $h$  whose *available\_port* bit vector contains all '0' values cannot be scheduled. For all schedulable switches, one of the bits in the vector is selected for allocation. For efficiency, we select the first available port and allocate the upward and downward paths. This is performed sequentially on a per source basis. Once level  $h$  has been scheduled, a new set of sources for level  $h+1$  are created from the set of scheduled sources at level  $h$ . The algorithm iterates until all levels are scheduled.

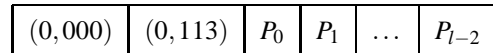
Figure 7 is the pseudo-code for scheduling requests that will be routed through level  $H$ . We assume switch  $(0, \sigma_0)$  is the source switch and switch  $(0, \delta_0)$  is the destination switch,

$$\text{where } \sigma_0 = \sum_{i=0}^{l-2} s_i w^i \text{ and } \delta_0 = \sum_{i=0}^{l-2} d_i w^i.$$

The complexity of a conventional algorithm is  $O(2l \log_l N)$ , while the complexity of our algorithm is  $O(l \log_l N)$ , where  $N$  is the system size and  $l$  is the greatest number of levels.

Figure 8 gives a fat tree  $\text{FT}(4, 4, 4)$ , which has four levels. Each switch node is a  $4 \times 4$  crossbar switch. At each level, one of four ports will be selected for upward routing.

It is assumed that a connection is requested from node 3 to node 95. source switch = switch  $(0, \sigma_0) = \text{switch}(0, s_2 s_1 s_0) = \text{switch}(0, 000)$ ; destination switch = switch  $(0, \delta_0) = \text{switch}(0, d_2 d_1 d_0) = \text{switch}(0, 113)$  The initial request is

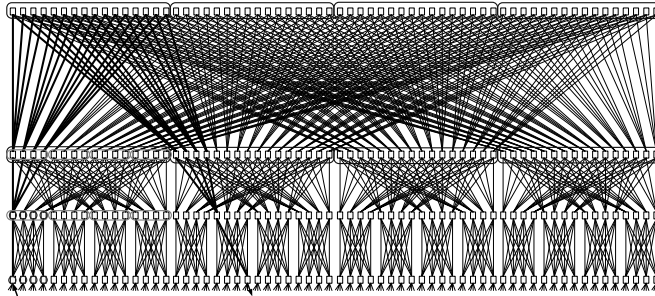


Step 1: select  $P_0$

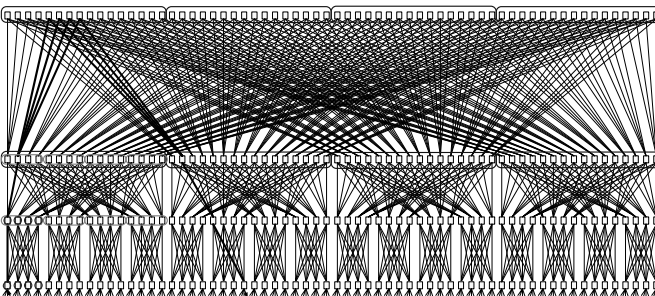
$$\begin{aligned} \text{Level} &= 0 \\ \text{source switch} &= \text{SW}(0, \sigma_0) = \\ &= \text{SW}(0, 000); \end{aligned}$$

1. for  $h = 0$  to  $H - 1$  {
2.   for a source switch and destination switch pair at level  $h$  {
3.     avail\_links = Ulink( $h, \sigma_h$ )[ $0 : w - 1$ ] Bit-Wise-AND Dlink( $h, \delta_h$ )[ $0 : w - 1$ ]
4.     if not(avail\_links = "000...0") {
5.       select  $i$  such that avail\_links[ $i$ ] = '1'
6.        $P_h = i$
7.       Ulink( $h, \sigma_h$ )[ $i$ ] = '0';
8.       Dlink( $h, \delta_h$ )[ $i$ ] = '0';
9.        $\sigma_{h+1} = s_{l-2}s_{l-3} \dots s_{h+1}P_0 \dots P_h$ ;
10.        $\delta_{h+1} = d_{l-2}d_{l-3} \dots d_{h+1}P_0 \dots P_h$ ;
11.     }

Figure 7: Pseudo-code for Level-wise scheduling algorithm



(a)  $P_0$  Selection.



(b)  $P_1$  Selection.

Figure 8: Level-wise scheduling example.

destination switch = SW( $0, \delta_0$ ) = SW( $0, 113$ ).

We assume that Ulink( $0, \sigma_0$ )[ $0$ ] is '1' and Dlink( $0, \delta_0$ )[ $0$ ] is '1'. Therefore  $P_0 = 0$ .

Step 2: select  $P_1$

Level = 1

source switch = SW( $1, \sigma_1$ ) = SW( $1, s_2s_1P_0$ ) = SW( $1, 000$ )

destination switch = SW( $1, \delta_1$ ) = SW( $1, d_2d_1P_0$ ) = SW( $1, 110$ )

We assume that Ulink( $1, \sigma_1$ )[ $0$ ] is '0' and Dlink( $1, \delta_1$ )[ $0$ ] is '1', which indicates port 0 is not available. Port 1 is checked.

Let us assume Ulink( $1, \sigma_1$ )[ $0$ ] is '1' and Dlink( $1, \delta_1$ )[ $0$ ] is '1', then  $P_1 = 1$ .

Step 3: select  $P_2$

Level = 2

source switch = SW( $2, \sigma_2$ ) = SW( $2, s_2P_0P_1$ ) = SW( $2, 001$ )

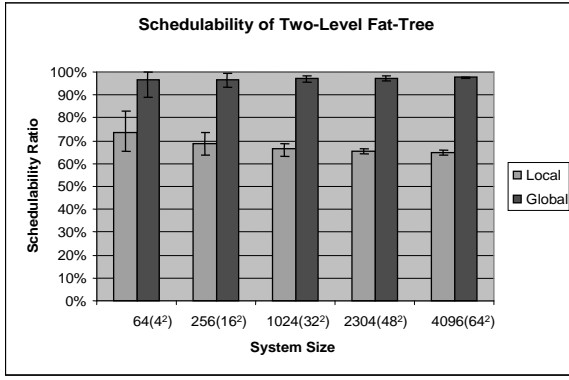
destination switch = SW( $2, \delta_2$ ) = SW( $2, d_2P_0P_1$ ) = SW( $2, 101$ )

We assume that Ulink( $2, \sigma_2$ )[ $0$ ] is '1' and Dlink( $2, \delta_2$ )[ $0$ ] is '1'

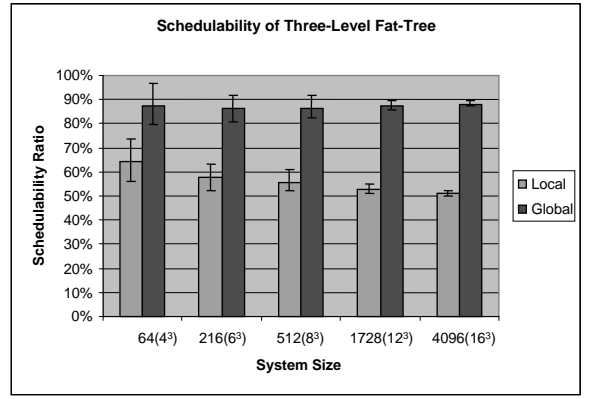
Therefore,  $P_2 = 0$

(0,000)	(0,113)	$P_0 = 0$	$P_1 = 1$	$P_2 = 0$
---------	---------	-----------	-----------	-----------

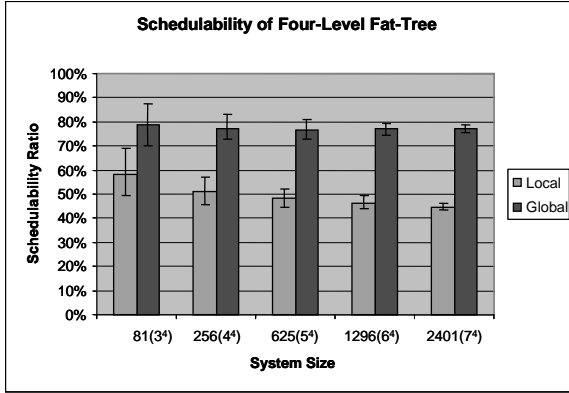
All ports are allocated successfully so the request from switch (0,000) to SW(0,113) is granted.



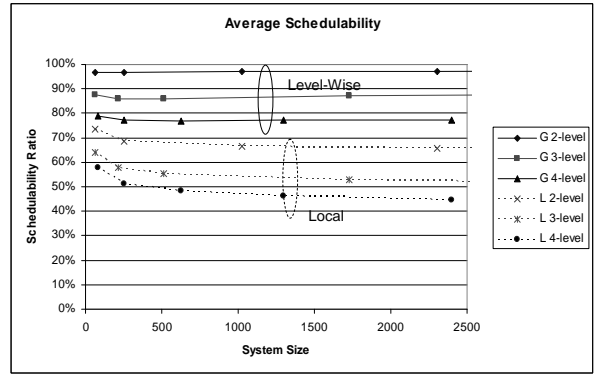
(a) Schedulability of two-level fat tree.



(b) Schedulability of three-level fat tree.



(c) Schedulability of four-level fat tree.



(d) Average schedulability.

Figure 9: Schedulability ratio comparison of fat-tree interconnected network.

## 5 Simulation results

The purpose of the simulation is to compare the performance of our Level-wise scheduling algorithm with local scheduling algorithm. We have performed system level simulation for randomly generated communication permutation based on two-level, three-level and four level fat-tree interconnected networks.

The experiments are based on SystemC based simulator to simulate scheduling procedures. We have built switch nodes and connected them in fat-tree topology. Each switch node has bidirectional input and bidirectional output. Network control signals, e.g. communication grants and requests, are passed through each switch node in parallel.

We generate random communication permutations as requested connections. Those communication requests are assigned as inputs of each source switch node. The requests are scheduled by our Level-wise scheduler and general adaptive fat-tree scheduler with local routing information. If a requested connection is successfully established, the request will be forwarded to the destination node. By checking the control signals received at destination nodes, we are able to compute the number of scheduled connections. We generate

a set of 100 random permutations for each test point. We use schedulability ratio to evaluate the scheduling capability, which is defined as the number of successful connections divided by the number of total requests.

Figure 9(a), 9(b), and 9(c) compares the schedulability ratio in two-level, three-level and four-level fat-tree communication networks. Each bar represents the average schedulability ratio in our experiments. The upper line and lower line at the top of each bar shows the maximum and minimum schedulability ratio, respectively. Figure 9(d) summarizes these results.

It can be observed that the Level-wise scheduling method provides higher schedulability ratio than the conventional local scheduler in all cases. The deviation of the schedulability ratio become less as the system size increases. This is due to technique for selecting routing paths. For both scheduling algorithms, the selection of upper port for upward routing is flexible while the path for downward routing is fixed. As the system size increases, communication requests have more upper ports from which to select when they are routed upward, which increases the possibility for successful routing. Additionally, the minimum schedulability ratio of the Level-wise scheduler is higher than the maximum

Table 1: Performance evaluation (Design targeting on Altera Stratix II FPGA)

System Size	Processing Time	
	Schedule	
	single request	all requests
64 ( $4 \times 4$ switch)	15 ns	480 ns
512 ( $8 \times 8$ switch)	17 ns	4352 ns
4096 ( $16 \times 16$ switch)	19 ns	38912 ns

schedulability ratio of the conventional scheduler. That is to say in the worst case the Level-wise scheduler is able to schedule more connections than the conventional scheduler’s best case.

It can also be observed that the conventional scheduler’s schedulability ratio decreases as the number of levels increases as shown in Figure 9(a). This is because for a high-level system, more levels need to be adjusted to configure a connection. A communication request may be successfully routed for most levels, but a failure routing in one level will destroy all previous routing for this request. However, the Level-wise scheduler using global information always outperforms the conventional scheduler using local information with negligible drop-off as system size increases. Thus, the improvement between the two approaches becomes more significant as the system size increases. In a network with more than 500 communication nodes, the improvement is over 30%. Note that only systems of size  $w^l$  are used for  $l$ -level systems.

## 6 Level-wise scheduling hardware architecture

Our scheduling algorithm can be implemented in either software or hardware. We present a pipelined hardware architecture for implementation of the Level-wise scheduler that may be used as a centralized fat tree scheduler. Our hardware design is targeted to an FPGA.

Figure 10 shows a hardware system for scheduling a three-level fat-tree interconnection network. For a three-level fat-tree, each communication request associates with no more than two upward ports,  $P_0$ , and  $P_1$ . Two main function blocks,  $P_0$  **Block** and  $P_1$  **Block** process the value of  $P_0$  and  $P_1$ , individually. When a request comes,  $P_0$  Block computes the value of  $P_0$  and the result is used by  $P_1$  Block to compute the value of  $P_1$ . While  $P_1$  Block is processing Request 1, the  $P_0$  Block can process the second request, Request 2, in pipelined fashion.

$P_0$  Block and  $P_1$  Block have identical hardware structures,

so only the  $P_0$  Block hardware design is described here. It consists of three function blocks, *load*, *compute*, and *update*. In the load stage, the source switch and destination switch are computed and the current Ulink and Dlink vectors containing the available links are read from the two memories. The compute stage is comprised pure combinational logic, which computes valid port number and updated Ulink and Dlink vectors. A port  $i$  can be selected only when both  $Ulink(h, \sigma_h)[i]$  and  $Dlink(h, \delta_h)[i]$  both equal ‘1’. First the Ulink and Dlink vectors are combined with a logic AND operation. Then one one available port is selected using a priority selector. Finally, in the update stage, the new Ulink and Dlink vectors are written back to the two memories. Figure 11 illustrates the hardware architecture of the  $P_0$  block. White, grey, and shaded blocks are used to indicate *load*, *compute*, and *update* function blocks respectively.

Based on the pipeline architecture, we have developed synthesizable hardware components in VHDL. These blocks were synthesized for an Altera Stratix II FPGA. The hardware for scheduling a three-level fat-tree interconnection network was tested with system size varying from 64 to 4096 nodes. Table 1 presents the post place and route synthesis results.

The result shows that for a communication system with 4096 communication nodes, our centralized scheduler is able to schedule an individual communication request within 20ns. Using less than 40  $\mu s$ , all 4096 communication requests can be scheduled.

## 7 Conclusion

This paper presents a Level-wise scheduling algorithm for fat tree interconnection networks. Our simulation results show the Level-wise algorithm improves schedulability, while still allowing the algorithm to be easily implemented in hardware with a reasonable complexity and performance. For example, by using global routing information, our Level-wise scheduler is capable of configuring a fat-tree interconnect with a 30% improvement in schedulability ratio over a conventional local scheduler. This technique is especially beneficial to setup long-lived connections. We have developed pipelined hardware architecture to evaluate complexity and processing capability of the scheduler. For a 4096 3-level fat tree, it is possible to schedule all connections in hardware in less than 40  $\mu s$ .

## References

- C.E.Leiserson, “Fat trees: universal networks for hardware-efficient supercomputing,” *IEEE Trans. on Computers*, vol. 34, no. 10, pp. 892–901, 1985.
- C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, and et al., “The network

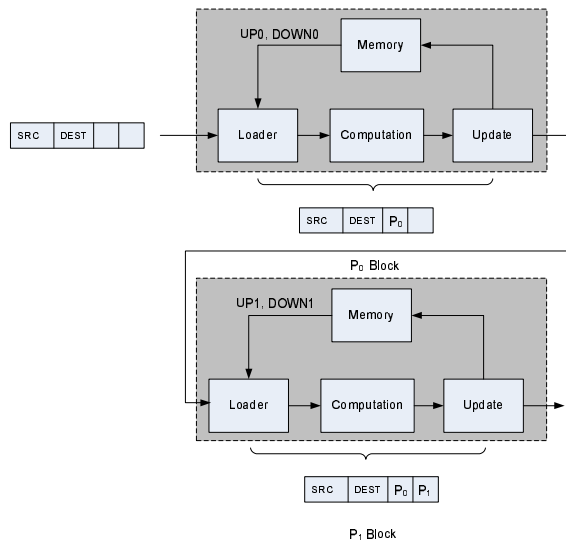


Figure 10: Level-wise scheduling Algorithm.

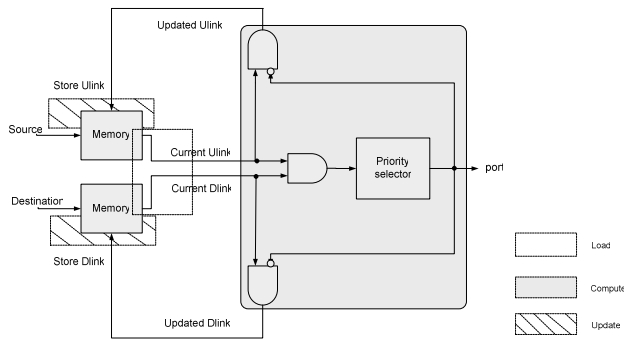


Figure 11: Level-wise scheduling Algorithm.

architecture of the connection machine CM-5,” in *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pp. 272–285, 1992.

Z.Bozkus, S.Ranka, and G.Fox, “Benchmarking the CM-5 multicomputer,” in *Fourth Symposium on the Frontiers of Massively Parallel Computation*, pp. 100–107, 1992.

K.E.Schauser and C.J.Scheiman, “Experience with active messages on the Meiko CS-2,” in *9th International Parallel Processing Symposium*, pp. 140–149.

“AlphaServer SC:terascale single-system-image supercomputing,” tech. rep., COMPAQ Inspiration Technology, 2002. <http://h18002.www1.hp.com/alphaserver/download/>.

J. Beecroft, D. Addison, F. Petrini, and M. McLaren, “Qs-NetII: an interconnect for supercomputing applications,” *IEEE Micro*, 2003.

L. M. Ni, Y. Gui, and S. Moore, “Performance evaluation of switch-based wormhole networks,” *IEEE transactions on parallel and distributed systems*, vol. 8, no. 5, pp. 462–474, 1997.

Y. Aydogan, C. B. Stunkel, C. Aykana, and B. Abali, “Adaptive source routing in multistage interconnection networks,” in *10th International Parallel Processing Symposium (IPPS '96)*, pp. 258–267, 1996.

H.Kariniemi and J. Nurmi, “New adaptive routing algorithm for extended generalized fat trees on-chip,” in *International Symposium on System-on-Chip (IEEE, ed.)*, pp. 113–118, 2003.