ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

David N. Turner (Ed)

Functional Programming, Glasgow 1995

Proceedings of the 1995 Glasgow Workshop on Functional
Programming,
Ullapool, Scotland, 10-12 July 1995

Paper:

Formal Specification of Interconnection Networks

John O'Donnell and Gudula Rünger

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

Formal Specification of Interconnection Networks

John O'Donnell

Computing Science Department, University of Glasgow
Glasgow, Britain

Gudula Rünger

Fachbereich Informatik, Universität des Saarlandes
Saarbrücken, Germany

Abstract

Interconnection networks are an important and well-studied topic in parallel computing and architecture, but a homogeneous and general method for defining and classifying the topologies and behaviors of interconnection networks is lacking. Topologies are usually specified informally by picture or more formally by permutations of wire enumerations. This paper presents an improved method for specifying multistage networks via permutations, along with two styles of formal functional specification of the entire network, using both a standard multistage organization and a generalized fat tree organization. This method is applied to two specific indirect multistage switch networks: the baseline and the butterfly. The functional specification emphasizes the similarities between the networks, and also captures the functionality provided by general-purpose network nodes.

1 Introduction

Indirect interconnection networks play an important role in the design of general purpose parallel architectures. A network can be seen as a black box realizing either communications between the processors and the memory (for shared memory systems) or interprocessor communications (for distributed systems). The design depends on performance criteria like latency, bandwidth, connectivity and functionality. A particularly important class is the *multistage switch network* comprising a number of stages, each consisting of primitive switching elements. Many such networks have a regular structure with the same number of switches in each stage, and where all switches have identical designs; examples include the omega, banyan, baseline and butterfly networks [1]. The input and output ports of consecutive stages are connected in order to construct the entire network. Many such networks have been shown to be topologically equivalent [9, 8]. Research issues include network structures, the relationships between different networks, and network communication algorithms [4, 6].

Although the field of interconnection networks is well studied, a concise and coherent way of defining them is lacking. In the case of multistage networks with regular stages and constant degree of wires, the stages are often defined by permutations of the enumerations of the wires, where the enumeration is usually based on bit manipulation of binary numbers. This approach may hide the relationship between networks, and it considers only network topology—it does not take account of the general processing capabilities of the switching nodes.

In this paper we study a systematic and coherent method addressing these two problems, and apply it to the baseline and butterfly networks. In common with the standard approach, we also use enumerations of the nodes to specify wires, but (unlike the standard method) we subdivide each stage into disjoint sets of nodes which are enumerated independently. This has three advantages: it simplifies the stage-combining functions, it produces a recursive network decomposition that clarifies the relationship between multistage networks and tree-structured networks (the tree and the fat-tree [5]), and it leads directly to a short and clear functional definition of the stage-combining functions, now seen as functions on lists.

Much past research on networks has focused exclusively on topology and message routing. However, it is also interesting to consider networks where the basic switch elements can compute an arbitrary function, rather than being restricted to simple routing. This provides additional functionality that has been exploited in real machines, e.g. in the emulation of shared memory machines with a butterfly network [7] where the fast access to the memory is realized by multiprefix operations [3], and in the NYU Ultracomputer [2]. Our functional specification provides a good foundation for studying this richer class of network.

Figure 1: Binary Tree and Multistage network with depth 1

In the standard multistage network specifications, only local wirings are specified directly; the global network is specified implicitly. In contrast, our functional specification is global, in that it defines a general communication/computation function describing the global behavior of the entire network as a complete set of communications takes place. The behavior of the network is a higher order function, and one of its arguments is a function defining the behavior of an individual node. This is more general than the usual approach, where it is assumed that nodes can only execute a fixed communication algorithm.

The functional specifications in this paper are expressed in Haskell. These are executable programs that formally express the connectivities of the networks and simulate their behaviors.

The remainder of the paper is organized as follows. Section 2 introduces terminology and gives the standard definitions of the baseline and banyan networks. Sections 3 and 4 then present these networks in more detail and give our new specifications for the intra-stage wiring. Section 5 defines the global network and completes the functional specifications. Section 6 compares our approach with the standard specifications and discusses the role of the functional specifications.

2 Multistage interconnection networks

This background section gives the standard semi-formal definitions of the baseline and butterfly networks. Alternative specifications will be developed in the following three sections.

A multistage interconnection network is an acyclic directed graph consisting of a number of subgraphs called stages. Each stage consists of a permutation graph (containing only wires) followed by a row of nodes, where each node takes two inputs and produces two outputs. The size of the network is characterized by the number of stages, which is called the depth d . The stages are enumerated $1, \dots, d$. The dimensionality is $d - 1$. A network with $d = 0$ has no stages, and consists only of a wire connecting a singleton input to a singleton output. Figure 1 shows the smallest nontrivial network, where $d = 1$, along with the corresponding tree.

The width of each stage is $n = 2^{d-1}$ nodes. Since each node has two inputs and two outputs, the entire network takes 2^d input signals and produces 2^d output signals; the inputs and outputs are organized as a list of n signal pairs. For example, the two networks in Figure 2 each have depth $d = 4$, with stages 1, 2, 3, 4, each stage contains $2^{d-1} = 8$ nodes, and the entire network has $2^d = 16$ input signals and output signals.

In Section 5 we will characterize the behavior of the network by two additional parameters: the interconnection function c and the node behavior function g .

The standard interconnection functions are defined by specifying explicitly which nodes are connected. (Section 5 will present a different approach.) To achieve this, each node is identified uniquely by its position within a stage (given as a binary number) and the number of that stage. More precisely, the node (b, s) is located in stage $s \in \{1, \dots, d\}$ and within the stage at position $b \in \{0, \dots, n - 1\}$, represented as a binary number. For regular interconnection networks, the stage interconnection function $C :: (Position, Stage) \rightarrow (Position, Stage)$ is specified as a relation on nodes. The following definitions of C_{baseline} and $C_{\text{butterfly}}$ for the baseline and the butterfly network are based on the binary enumeration (see also [4]).

Baseline network: The stage connecting function C_{baseline} of the baseline network is given by: $C_{\text{baseline}}(b, s) = (b', s + 1)$ where b' is the string obtained by performing a left cyclic shift on the rightmost s bits of b , or b' is the string obtained by performing a left cyclic shift on the rightmost s bits of b and then changing the rightmost bit of b (see Figure 2).

Figure 2: Three-dimensional baseline network and butterfly network

Figure 3: Binary fat tree network

Butterfly network: The stage connecting function $C_{\text{butterfly}}$ of the baseline network is given by: $C_{\text{butterfly}}(b, s) = (b', s+1)$ where b and b' are identical or b and b' differ in precisely bit s (see Figure 2).

Both definitions depend explicitly on the number of stages and, especially in the case of the baseline network, the stage number is strongly involved in computing the value $C_{\text{baseline}}(b, s)$. This kind of function definition is not suitable as a starting point for a more abstract specification of interconnection function using recursion and lists of nodes (of one stage).

3 The baseline network

We now present an alternative specification of the baseline network which is a more appropriate starting point for the derivation of a global specification of the network in the functional style. This new specification is based on a subdivision of each stage into several independent substages, leading to much simpler interconnection functions.

Stage s is organized into $n/2^{s-1}$ substages, for $s = 1, \dots, d$. The nodes within each substage are enumerated with natural numbers starting with 0, independently from the nodes in other substages. The substages of stage s can be enumerated $0, \dots, 2^{s-1} - 1$.

The division into substages emphasize that the connections between consecutive stages fall into groups of connections. Every substage (except in the top level) has connections to exactly one substage of the subsequent stage, and every substage (except those in the bottom level) has connections to exactly two substages of the level below. This results in a structure of substages organized like a binary tree (see Figure 3).

The enumeration leads to a very clear connection function which we call the *local baseline connection*. The local baseline connection function connects the outgoing edges of two substages X and Y of equal size with the ingoing edges of a third substage Z with double the size, using the indices of the enumeration.

Definition 3.1 (*Local baseline connection*)

Let $X = (X_0, \dots, X_{n-1})$ and $Y = (Y_0, \dots, Y_{n-1})$ be two stages with the same number of nodes n , and let $Z = (Z_0, \dots, Z_{2n-1})$ be a stage of nodes of size $2n$. Each node of X , Y , and Z has degree two, i.e. two ingoing and two outgoing edges. The nodes in each stage are numbered consecutively starting with 0.

The local baseline connectivity is a relation bas with exactly two function values for each argument. It is defined as

$$\begin{aligned} bas : \{0, \dots, n-1\} &\rightarrow \{0, \dots, 2n-1\} \\ bas(i) &= (2i), bas(i) = (2i+1) \end{aligned} \quad (1)$$

The edges connecting the substages X (or Y) with Z are $(X_i, Z_{bas(i)})$ (or $(Y_i, Z_{bas(i)})$), i.e. outgoing edges of X_i (or Y_i) are connected with ingoing edges of Z_{2i} and Z_{2i+1} .

The connection between two substages X, Y and one subsequent substage Z is identical to the direct *shuffle* connection (in Definition 3.1 defined with natural numbers). Thus, the baseline network can be interpreted as a structured combination of shuffle functions. In contrast to the omega network, there are several shuffle functions on each level.

Messages The edges (X_i, Z_j) between nodes X_i and Z_j represent wires for transmitting messages between the two nodes connected by an edge. This means X_i (or Y_i) sends two messages, one to the node Z_{2i} and one to the node Z_{2i+1} . In order to make messages explicit, we express the local baseline connectivity for pairs of messages (or wires) which are assigned to nodes as their pairs of input or output messages. Let

$$\begin{aligned} x_i &= (x_i^0, x_i^1) \text{ be the output of node } X_i, i = 0, \dots, n-1 \\ y_i &= (y_i^0, y_i^1) \text{ be the output of node } Y_i, i = 0, \dots, n-1 \\ z_k &= (z_k^0, z_k^1) \text{ be the input of node } Z_k, k = 0, \dots, 2n-1. \end{aligned}$$

The local baseline connectivity is fulfilled if

$$(z_k^0, z_k^1) = \begin{cases} (x_i^0, y_i^0) & \text{for } k = 2i, i = 0, \dots, n-1 \\ (x_i^1, y_i^1) & \text{for } k = 2i+1, i = 0, \dots, n-1 \end{cases} \quad (2)$$

i.e. the input (z_k^0, z_k^1) of Z_k is identical to a pair composed of messages coming from X and Y .

Equation 2 follows immediately from Equation 1 by fixing an order for the pairs of messages. Because of the fixed order the specification is less general than the connectivity described in Definition 3.1. Message pairs of opposite order would also realize the more general baseline definition.

Functional specification The functional specification relies on a representation of stages X of n nodes as lists of nodes $X = [X_0, \dots, X_{n-1}]$. One combining step is the composition of two lists $X = [X_0, \dots, X_{n-1}]$ and $Y = [Y_0, \dots, Y_{n-1}]$ of the same length with a list $Z = [Z_0, \dots, Z_{2n-1}]$ of length $2n$. Similarly, we define corresponding lists of pairs of outgoing messages $xs = [x_0, \dots, x_{n-1}]$, $ys = [y_0, \dots, y_{n-1}]$ and a list of pairs of incoming messages $zs = [z_0, \dots, z_{2n-1}]$.

Functional programming provides a way to give a *global* network specification based on the list representation of nodes or message pairs, without relying on indices. The global description of the baseline connectivity uses the entire lists of input and output pairs, taking the lists of output signals $xs = [x_0, \dots, x_{n-1}]$ and $ys = [y_0, \dots, y_{n-1}]$ as arguments, and producing the input list of signal pairs $zs = [z_0, \dots, z_{2n-1}]$ as a result, while ensuring that Equation 2 holds.

The function `baseline_connect` performs the global connection of substages. It takes two lists of message pairs as inputs and creates one new list of message pairs; the list may have arbitrary size but the input lists must have the same size. The function `expandpairs` takes a list of message pairs of a list of nodes and produces the plain list of messages, i.e. the assignment of messages to nodes is released.

```
baseline_connect :: [(a,a)], [(a,a)] -> [(a,a)]
baseline_connect (xs,ys) =
    zip (expandpairs xs) (expandpairs ys)
```

```

expandpairs :: [(a,a)] -> [a]
expandpairs [] = []
expandpairs ((a,b):xs) = a : b : expandpairs xs
    
```

The following lemma establishes the equivalence of the global and local descriptions of the baseline interconnection.

Lemma 3.1 (*Equivalence of local and global specification*) *For all nodes, the global baseline connectivity `baseline_connect` is equivalent to the local baseline connectivity Equation 2 expressed on the level of message pairs assigned to nodes.*

Proof. The equivalence is shown with equational reasoning by consecutively using the definition of `baseline`, the definition of message pairs `xs` and `ys`, the definition of `expandpairs`, and applying `zip`.

$$\begin{aligned}
 & \text{baseline } xs \text{ } ys \\
 &= \text{zip } (\text{expandpairs } xs) (\text{expandpairs } ys) \\
 &= \text{zip } (\text{expandpairs } [(x_i^0, x_i^1) \mid i \leftarrow [0 \dots n-1]]) \\
 & \quad (\text{expandpairs } [(y_i^0, y_i^1) \mid i \leftarrow [0 \dots n-1]]) \\
 &= \text{zip } [x_i^j \mid i \leftarrow [0 \dots n-1], j \leftarrow [0, 1]] \\
 & \quad [y_i^j \mid i \leftarrow [0 \dots n-1], j \leftarrow [0, 1]] \\
 &= [(x_i^j, y_i^j) \mid i \leftarrow [0 \dots n-1], j \leftarrow [0, 1]] \\
 &= [(z_k^0, z_k^1) \mid k \leftarrow [0, \dots, 2n-1]]
 \end{aligned}$$

using $k = 2i + j$, $j = 0, 1$ $i = 0, \dots, n-1$ in the last derivation step.

Lemma 3.1 shows the equivalence of local and global specification. Thus, as well as Equation (2), the functional specification is a specific realization of the original Definition 3.1.

4 The butterfly network

Although the interconnection pattern is quite different, the butterfly network can be specified in a similar manner to the baseline. This works because both networks have similar global recursive decompositions. This section summarizes the local butterfly connectivity, defines the message relation, gives the global functional specification, and proves the equivalence of the local and global definitions.

Definition 4.1 (*Local butterfly connection*)

Let $X = (X_0, \dots, X_{n-1})$ and $Y = (Y_0, \dots, Y_{n-1})$ be two stages with the same number of nodes n and let $Z = (Z_0, \dots, Z_{2n-1})$ be a stage of nodes of size $2n$. Each node of X , Y , and Z has the degree two, i.e. two ingoing and two outgoing edges. The nodes in each stage are numbered consecutively starting with 0.

The local butterfly connectivity is a relation `bly` with exactly two function values for each argument. It is defined as:

$$\begin{aligned}
 & \text{bly} : \{0, \dots, n-1\} \rightarrow \{0, \dots, 2n-1\} \\
 & \text{bly}(i) = i, \quad \text{and} \quad \text{bly}(i) = i + n
 \end{aligned} \tag{3}$$

The edges connecting the substages X (or Y) with Z are $(X_i, Z_{\text{bly}(i)})$, (or $(Y_i, Z_{\text{bly}(i)})$) i.e. outgoing edges of X_i (or Y_i) are connected with ingoing edges of Z_i and Z_{i+n} , $i = 0, \dots, n-1$.

Messages We consider lists of message pairs xs, ys , and zs , as introduced in the last section. The local butterfly connectivity on the level of messages with a fixed order of messages is expressed by the following equation:

$$(z_k^0, z_k^1) = \begin{cases} (x_k^0, y_k^0) & \text{for } k = 0, \dots, n-1 \\ (x_k^1, y_k^1) & \text{for } k = n, \dots, 2n-1 \end{cases} \tag{4}$$

Functional specification We consider substages and incoming or outgoing messages of substages in their list representation. The local butterfly connectivity as well as the relation between messages in Equation (4) contain a dependence on the length n of the arguments X and Y (or xs and ys) which is a dependence on the specific stage of X and Y . This dependence is released when defining the global functional definition. The definition is based on message pairs which also for this network has turned out to be the most intuitive and clear definition.

The global butterfly connectivity on message pairs is described by the function `butterfly`. The function separates emitted message pairs of xs and ys and produces new message pairs by combining first items of xs and ys and by combining second items of xs and ys .

```
butterfly_connect :: [(a,a)], [(a,a))] -> [(a,a)]
butterfly_connect (xs,ys) =
    zip (map fst xs) (map fst ys) ++
    zip (map snd xs) (map snd ys)
```

Lemma 4.1 (*Equivalence of local and global specification*) *For all nodes, the global butterfly connectivity `butterfly_connect` is equivalent to the local butterfly connectivity Equation 3 expressed on the level of message pairs assigned to nodes.*

Proof. The proof uses equational reasoning based on the definitions of `butterfly_connect`.

$$\begin{aligned}
 & \text{butterfly_connect } xs \text{ } ys \\
 &= \text{zip } (\text{map fst } [(x_i^0, x_i^1) \mid i \leftarrow [0, \dots, n-1]]) \\
 & \quad (\text{map fst } [(y_i^0, y_i^1) \mid i \leftarrow [0, \dots, n-1]]) ++ \\
 & \quad \text{zip } (\text{map snd } [(x_i^0, x_i^1) \mid i \leftarrow [0, \dots, n-1]]) \\
 & \quad (\text{map snd } [(y_i^0, y_i^1) \mid i \leftarrow [0, \dots, n-1]]) \\
 &= \text{zip } ([x_i^0 \mid i \leftarrow [0, \dots, n-1]]) ([y_i^0 \mid i \leftarrow [0, \dots, n-1]]) \\
 & \quad ++ \text{zip } ([x_i^1 \mid i \leftarrow [0, \dots, n-1]]) ([y_i^1 \mid i \leftarrow [0, \dots, n-1]]) \\
 &= [(x_i^j, y_i^j) \mid i \leftarrow [0, \dots, n-1], j \leftarrow [0, 1]] \\
 &= [(z_k^0, z_k^1) \mid k \leftarrow [0, \dots, 2n-1]] \\
 & \text{using } k = i + nj, \quad i = 1, \dots, n-1, \quad j = 0, 1 \text{ in the last derivation step.}
 \end{aligned}$$

The equational reasoning in Lemma 4.1 (as well as in Lemma 3.1) rely on list comprehension. The final list $[(x_i^j, y_i^j) \mid i \leftarrow [0, \dots, n-1], j \leftarrow [0, 1]]$ of output messages corresponds to $[(z_k^0, z_k^1) \mid k \leftarrow [0, \dots, 2n-1]]$ because of the order of indices i and j . This order is the essential difference to the result in the proof of Lemma 3.1 $[(x_i^j, y_i^j) \mid j \leftarrow [0, 1], i \leftarrow [0, \dots, n-1]]$ where the order of indices is the reverse, i.e. j and then i . Thus, the derivation of the functional specification of stage connectivities for both the baseline and the butterfly network results in a simple characteristics of the difference between the networks: the difference is captured in the order of indices of the list representation of messages.

5 Unified functional specifications

Sections 3 and 4 specified the interconnections between substages of two adjacent stages. This section defines the entire multistage network in two different styles: a nonrecursive definition similar to the standard definition (although more precise), and a recursive definition in the form of an abstract fat tree.

The definitions of the stage and networks are higher order, taking the interconnection function c and the node behavior function g as arguments. This enables a large class of multistage networks, including the baseline and butterfly, to be defined simply by varying the interconnection function.

5.1 Nonrecursive network definition

The nonrecursive definition views a network as the composition of $d > 0$ stages, each $n = 2^{d-1}$ units wide (suggesting a rectangular layout for the network). This definition formalizes the standard network specification, and it corresponds to the definition given in Section 2.

First a stage is defined as the composition of an interconnection function c and a row of nodes with behavior g . The interconnection function operates on a sequence of substages each containing 2^{s-1} nodes, where $s \in \{1, \dots, d\}$ is the stage number.

The definition first splits the list of n input pairs into a sequence of $n/2^{s-1}$ groups of width $w = 2^{s-1}$. The inputs to each substage must be further subdivided into two parts of width $w/2$, since each substage is connected to two smaller substages in the next level down. Therefore the application *group* w xs , where w is the width of a substage, splits the flat list xs of input pairs into a sequence of pieces containing n/w group-pairs organized into a left part and a right part, each of length $w/2$. For example

```
group 4 [(0,1),(2,3),(4,5),(6,7),(8,9),(10,11),(12,13),(14,15)]
==> [[[(0,1),(2,3)], [(4,5),(6,7)]],
      [(8,9),(10,11)], [(12,13),(14,15)]]
```

i.e. *group* w with $w = 4$ splits a list of $n = 8$ message pairs into $n/w = 2$ group-pairs each pairs with left part and right part of length $w/2 = 2$.

Thus each group of inputs of width w consists of a pair (l, r) where l and r are lists of length $w/2$, and each element of these lists is a pair of input signals. The function is defined as an iteration over xs .

```
group :: Int -> [a] -> [[a],[a]]
group w [] = []
group w xs = (take w' xs, take w (drop w' xs))
              : group w (drop w xs)
  where w' = w `div` 2
```

Each group has type $[(S, S)]$, where S is the signal type. After the inputs are grouped, the connection function c is mapped over the list of groups, thus applying it independently to each group of inputs. The application of connection function c to one group-pair corresponds to the connectivity of two substages to one substage in the next stage described in Sections 3 and 4.

The resulting outputs from the interconnections are then concatenated into a flat list of signal pairs; finally the node function g is mapped over these to produce the outputs from the stage.

```
stage
  :: (([a,a]],[a,a])->[a,a]) -- c = connection
  -> ([a,a]->[a,a])           -- g = node behavior
  -> Int                        -- s = stage number
  -> [[a,a]]                   -- xs = inputs to stage s
  -> [[a,a]]                   -- outputs from stage s
```

```
stage c g s
  | s==1 = map g
  | s>1  = map g . concat . map c . group (2^(s-1))
```

The complete network can now be defined as the composition of a sequence of stages:

$$\text{multistage } c \ g \ d = \text{stage } c \ g \ d \circ \text{stage } c \ g \ (d-1) \circ \dots \circ \text{stage } c \ g \ 1$$

The most complicated aspect of the network is the splitting of inputs into groups of different sizes within different stages; since that has already been accomplished by *stage*, the definition of *multistage* is a simple iteration:

```
multistage
  :: (([a,a]],[a,a])->[a,a]) -- c = connection
  -> ([a,a]->[a,a])           -- g = node behavior
  -> Int                       -- d = depth of network
```



```

-> [(a,a)]                -- xs = inputs to first stage
-> [(a,a)]                -- outputs from last stage

multistage c g d
| d==1 = stage c g 1
| d>0  = stage c g d . multistage c g (d-1)
    
```

Each stage consists of one or more independent groups of connections and nodes, so a sufficiently wide stage can be grouped into the concatenation of a sequence of independent shorter stages. This property is stated formally in the following lemma.

Lemma. If $\text{length } xs = \text{length } ys = 2^k$ where $k \geq s - 1$, then

$$\text{stage } c \ g \ s \ (xs ++ ys) = \text{stage } c \ g \ s \ xs ++ \text{stage } c \ g \ s \ ys$$

Proof. The lemma is proved by a direct simplification. The crucial step is the second one, where a grouping of $xs ++ ys$ is factored into separate groupings over xs and ys ; this step is justified since $\text{length } (xs ++ ys)$ is at least twice the group size 2^s .

$$\begin{aligned}
 & \text{stage } c \ g \ s \ (xs ++ ys) \\
 &= \text{map } g \ (\text{concat} \ (\text{map } c \ (\text{group } 2^{s-1} \ (xs ++ ys)))) \\
 &= \text{map } g \ (\text{concat} \ (\text{map } c \ (\text{group } 2^{s-1} \ xs ++ \text{group } 2^{s-1} \ ys)))) \\
 &= \text{map } g \ (\text{concat} \ ((\text{map } c \ (\text{group } 2^{s-1} \ xs)) ++ \\
 & \quad (\text{map } c \ (\text{group } 2^{s-1} \ ys)))) \\
 &= \text{map } g \ (\text{concat} \ (\text{map } c \ (\text{group } 2^{s-1} \ xs)) ++ \\
 & \quad \text{concat} \ (\text{map } c \ (\text{group } 2^{s-1} \ xs)))) \\
 &= \text{map } g \ (\text{concat} \ (\text{map } c \ (\text{group } 2^{s-1} \ xs)) ++ \\
 & \quad \text{map } g \ (\text{concat} \ (\text{map } c \ (\text{group } 2^{s-1} \ ys)))) \\
 &= \text{stage } c \ g \ s \ xs ++ \text{stage } c \ g \ s \ ys
 \end{aligned}$$

Corollary. If $\text{length } xs = \text{length } ys = 2^k$ where $k \geq d - 1$, then

$$\text{multistage } c \ g \ d \ (xs ++ ys) = \text{multistage } c \ g \ d \ xs ++ \text{multistage } c \ g \ d \ ys$$

5.2 Recursive network definition

Each group of nodes at any stage is connected to exactly two groups in the stage below. This property gives the multi-stage network an abstract tree structure which can be exploited by redefining the network using a general tree function. Using this method, each tree node is actually a group of network nodes; to avoid confusion, we call this a *supernode*. A general tree supernode corresponds to a substage of the network (see Figure 3).

Unlike an ordinary tree, the total number of wires remains the same ($2n$) at every level of the tree. Consequently, our tree definition of the multistage network is similar to the fat tree architecture [5]. However, the standard fat tree assumes a particular supernode architecture. Our tree is more general in two respects: it allows any interconnection function within the supernode (the c parameter) and it allows any individual node behavior (the g parameter). Therefore we call the definition below a *general fat tree*.

The general fat tree takes exactly the same arguments as the multistage function, but it replaces the grouping operation by an explicit tree recursion. The behavior of a supernode is $\text{map } g \circ c$. The input to the tree is a list of $n = 2^{d-1}$ signal pairs, so there are $2n$ input signals. The most important characteristic of the fat tree is that bandwidth remains constant at all levels; therefore the output is also a list of n signal pairs.

```

gfattree
  :: ([[(a,a)],[(a,a)]]->[(a,a)]) -- c = connection
    
```

```

-> ((a,a)->(a,a))      -- g = node behavior
-> Int                  -- d = depth of tree
-> [(a,a)]              -- xs = inputs to first stage
-> [(a,a)]              -- outputs from last stage

gfattree c g d xs
| d==1 = map g xs
| d>1  = map g (c (gfattree c g (d-1) (take i xs),
                  gfattree c g (d-1) (drop i xs)))
where i = 2^(d-2)
    
```

5.3 Equivalence of network definitions

The following theorem establishes that *multistage* and *gfattree* define identical networks, although their differing forms emphasize that they have different origins, a multistage network and a fat tree.

Theorem. *multistage* = *gfattree*

Proof. Both functions are applied to arbitrary well typed arguments *c*, *g*, *d* and *xs*, such that $\text{length } xs = 2^{d-1}$, resulting in

$$\text{multistage } c \ g \ d \ xs = \text{gfattree } c \ g \ d \ xs$$

This equation is proved by induction over *d*.

Base case ($d = 1$).

<i>multistage</i> <i>c g 1 xs</i>	
= <i>stage</i> <i>c g 1 xs</i>	{ <i>multistage</i> }
= <i>map</i> <i>g xs</i>	{ <i>stage</i> }
= <i>gfattree</i> <i>c g 1 xs</i>	{ <i>gfattree</i> }

Inductive case ($d > 1$). Assume $\text{length } xs = \text{length } ys = 2^{d-1}$.

<i>multistage</i> <i>c g (d + 1) (xs++ys)</i>	
= <i>stage</i> <i>c g d (multistage c g d (xs++ys))</i>	{ <i>multistage</i> }
= <i>stage</i> <i>c g d (multistage c g d xs++</i>	
<i>multistage c g d ys)</i>	{Corollary.}
= <i>map</i> <i>g (concat (map c (group 2^{d-1}</i>	
(<i>multistage c g d xs ++ multistage c g d ys))))</i>	{ <i>stage</i> }
= <i>map</i> <i>g (concat (map c</i>	
[(<i>multistage c g d xs, multistage c g d ys</i>)]))	{ <i>group</i> }
= <i>map</i> <i>g (concat [c (multistage c g d xs,</i>	
<i>multistage c g d ys]))</i>	{ <i>map</i> }
= <i>map</i> <i>g (c (multistage c g d xs,</i>	
<i>multistage c g d ys))</i>	{ <i>concat</i> }
= <i>map</i> <i>g (c (gfattree c g d xs, gfattree c g d ys))</i>	{Hypothesis}
= <i>map</i> <i>g (c (gfattree c g d (take 2^{d-1} (xs++ys)),</i>	
<i>gfattree c g d (drop 2^{d-1} (xs++ys))))</i>	{ <i>xs</i> = 2 ^{d-1} }
= <i>gfattree</i> <i>g c (d + 1) (xs++ys)</i>	{ <i>gfattree</i> }

An interesting point in this proof is that *concat* appears only in a trivial way; therefore it might appear that it could be omitted from the definition of *stage*. However, this proof relies on the Corollary and Lemma, which make nontrivial use of the *concat*.

5.4 Complete network specifications

Both networks (baseline and butterfly) can now be defined using either network definition (nonrecursive or general fat tree), producing four definitions. These definitions represent special realizations of `multistage` and `gfattree`, respectively, and `multistage` and `gfattree` capture the identical characteristics of the networks.

```
baseline1, baseline2, butterfly1, butterfly2
  :: ((a,a)->(a,a)) -- g = node behavior
  -> Int           -- n = number of input pairs
  -> [(a,a)]       -- xs = inputs to baseline network
  -> [(a,a)]       -- outputs from baseline network

baseline1 = multistage baseline_connect
baseline2 = gfattree    baseline_connect
butterfly1 = multistage butterfly_connect
butterfly2 = gfattree    butterfly_connect
```

From the theorem in Section 5.3 we get the following equalities.

Corollary.

1. *baseline1 = baseline2*
2. *butterfly1 = butterfly2*

6 Conclusion

Previous specifications of multistage networks have several limitations: they rely on properties of the binary representations of node positions, which tends to hide the relationships between similar networks, they specify only the interconnections precisely but the entire network specification is informal, and they often assume a simple fixed node architecture which places an unnecessary limitation on the capabilities of the entire network.

This paper has addressed these problems by presenting three new ways to specify a network. The first method is based on a permutation function similar to the traditional approach, except we number the nodes with each substage independently, which greatly simplifies the interconnection function. The second method is a functional specification which completely formalizes the traditional approach, producing an executable specification suitable for formal reasoning. The third method is another functional specification that interprets the multistage network as a generalized tree.

Topics for future work include investigation of the properties of the global network behavior functions. Past experience suggests this to be a promising line of inquiry, since the full capabilities of tree networks did not become clear until global tree functions were studied. Furthermore, the methods presented here should be applied to a broader range of networks.

References

- [1] G. Almasi and A. Gottlieb. *Highly Parallel Computing*. The Benjamin/Cummings Publishing Company, Inc., 1994.
- [2] A. Gottlieb, B. Lubachevsky and L. Rudolph. *Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors*. In *ACM Transactions on Programming Languages and Systems*, 5(2):164–189, 1983.
- [3] Thomas Grün, Thomas Rauber, and Jochen Röhrig. The programming environment of the SB-PRAM. In *Proc. 7th IASTED/ISMM Int.l Conf. on Parallel and Distributed Computing and Systems*, Washington DC, October 1995.

- [4] F.T. Leighton. *Introduction to Parallel Algorithms and Architecture*. Morgan Kaufmann Publishers, 1992.
- [5] Ch.E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.
- [6] K. Padmanabhan. Cube structures for multiprocessors. *Communications of the ACM*, 33(1):43–52, 1990.
- [7] A.G. Ranade. How to emulate shared memory. In *28th Annual Symposium on Foundations of Computer Science*, pages 185–194. IEEE Computer Society Press, 1987.
- [8] H.J. Siegel. *Interconnecting Networks for Large-Scale Parallel Processing*. Lexington Books, 1985.
- [9] Chuan-Lin Wu and Tse-Yun Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, C-29(8):694–702, August 1980.