DISTRIBUTED COMPUTING ON ANONYMOUS HYPERCUBES WITH FAULTY COMPONENTS*

Evangelos Kranakis †
(kranakis@scs.carleton.ca)

Nicola Santoro ‡
(santoro@scs.carleton.ca)

Carleton University
School of Computer Science
K1S 5B6 Ottawa, Ontario, Canada

Abstract

We give efficient algorithms for distributed computation on anonymous, labeled, asynchronous hypercubes with possible faulty components (i.e. processors and links) and deterministic procoessors. Initially, the processors know only the size of the network and that they are inter-connected in a hypercube topology. Faults may occur only before the start of the computation (and that despite this the hypercube remains a connected network). However, the processors do not know where these faults are located. As a measure of complexity we use the total number of bits transmitted during the execution of the algorithm and we concentrate on giving algorithms that will minimize this number of bits. The main result of this paper is an algorithm for computing Boolean functions on anonymous hypercubes with bit complexity $O(N\delta_n(\gamma)^2\lambda^2 \log\log N)$, where γ is the number of faulty components (i.e. links plus processors), λ is the number of links which are either faulty, or non-faulty but adjacent to faulty processors, and $\delta_n(\gamma)$ is the diameter of the hypercube. An identical complexity bound holds for computing the automorphism group of the faulty hypercube under this distributed model of computation.

1980 Mathematics Subject Classification: 68Q99
CR Categories: C.2.1
Key Words and Phrases: Anonymous labeled hypercube, Boolean function, diameter, faulty component, group of automorphisms.

*A preliminary version of this paper has appeared in the proceedings of the 6th International Workshop on distributed algorithms, held in Haifa, Israel, November 2-4, 1992.

†Research supported in part by NSERC grant #907002.
‡Research supported in part by NSERC grant #A2415

1 Introduction

In this paper we consider algorithms which are appropriate for distributed computation on anonymous, labeled, asynchronous, n-dimensional hypercubes Q_n with faulty components (i.e., processors and links). The processors occupy the nodes of a hypercube and want to compute a given Boolean function f on $\leq N = 2^n$ variables. Initially each non-faulty processor p has an input bit b_p. When the computation terminates all processors must output the same value $f(< b_p : p \text{ non-faulty} >)$. A boolean function is called computable if all the processors of the network compute its value correctly on all inputs. (Our notation b_p for the bit associated with processor p does not mean that we assign names to processors. In addition, the input $< b_p : p \in$ non-faulty $>$ represents the assignment of bits to all the non-faulty processors of the network, and it will be computed by all the processors via an "input collection" algorithm.)

The problem arising is to determine the computability and associated bit complexity (i.e. total number of bits transmitted) of Boolean functions on faulty hypercubes. In the present paper we give efficient algorithms for computing Boolean functions on such networks and also consider the related problem of computing the automorphism group of the network.

1.1 Assumptions and Related Literature

The network we consider is the anonymous, asynchronous hypercube with possible faulty components. The number of faulty components may be arbitrary as long as the hypercube remains connected. If a processor is faulty then all the links adjacent to it are also interpreted as faulty. Faults may occur only before the start of the computation. We assume that the network links are FIFO, and that the processors have a sense of direction. By this we mean that the hypercube is canonically labeled (the label of link xy is i if and only if x, y differ at exactly the ith bit) and that these labels are known to the processors concerned. In addition we assume that the following assumptions hold:

- the processors know the network topology (in this instance hypercube), and the size of the network, but they do not necessarily know where the faulty components may be,
- the processors are anonymous (i.e., they do not know either the identities of themselves or of the other processors), they are deterministic (i.e. they all run deterministic algorithms), and they all run the same algorithm given the same data,
- the processors can distinguish the faulty links adjacent to it, as well as non-faulty links adjacent to a faulty processor.

The assumptions listed above are meant to take "maximum" advantage of network distributivity. For a discussion regarding the necessity of some of the above assumptions see [2]. Routing algorithms on hypercubes have been studied in [6]. Faulty hypercube networks have been examined in several papers under the much stronger assumption of synchronous and/or non-identical processors. In such networks it is possible to apply reconfiguring techniques [7] (nodes of an $n-1$-dimensional hypercube are mapped into non-faulty nodes of an n-dimensional hypercube with $O(1)$ dilation) or even non-faulty subcube techniques [5] (for a given k determine an $n-k$-dimensional subcube with no faulty links). However such techniques are not applicable in our case since they require the availability of processor identities.

1.2 Notation

Let γ denote the number of faulty components of the network, i.e. faulty links plus faulty processors. Let π be the number of faulty processors and λ the number of links which are either faulty, or non-faulty but adjacent to a faulty processor. Since a hypercube has $\log N$ faulty links per faulty processor we obtain $\leq \pi \log N$ faulty links associated with these π faulty processors. In general we have that $\gamma \leq \lambda + \pi$ and it is easy to see that equality may not be true. Notice that our definition of λ suggests that in the complexity results we encounter in section 4 we interpret as faulty all the links which are adjacent to a faulty processor.

Let Q_n denote the n-dimensional hypercube on $N = 2^n$ nodes; xy is a link of Q_n, where $x = x_1 \cdots x_n$ and $y = y_1 \cdots y_n$, if $x_i \neq y_i$ for a unique i; in addition, i is called the label of xy and we write $\ell(xy) = i$. (At this point it is necessary to emphasize that this global labeling is unknown to the processors. The algorithm to be discussed in the sequel uses only the fact that the processors know the labels of their adjacent links.) Let $Q_n[l_1, \ldots, l_\lambda]$ denote the hypercube Q_n with the links $l_1, \ldots, l_\lambda$ faulty. In general, the hypercube always remains a connected graph if the number λ of faulty links is $< \log N$. However it is possible that the hypercube remains connected even if $\lambda \geq \log N$.

We define $\delta_n(\lambda)$ as the maximal possible diameter of a connected hypercube with at most λ faulty links, i.e. $\delta_n(\lambda) := \max\{diam(Q_n[l_1, \ldots, l_\rho]) : \rho \leq \lambda$ and $Q_n[l_1, \ldots, l_\rho]$ is connected$\}$. We define similarly $\delta_n(\gamma)$ for the more general case of hypercubes with at most γ faulty components.

1.3 Results of the paper

Previous results on computing Boolean functions on asynchronous, anonymous, labeled networks can be summarized as follows.

Network	Bit Complexity	Paper
Rings	$O(N^2)$	[3]
n-Tori, n constant	$O(N^{1+1/n})$	[4]
Hypercubes: $\gamma = 0$	$O(N \log^4 N)$	[8]
Hypercubes: $\gamma \geq 1$	$O(N\lambda^2\delta_n(\gamma)^2 \log\log N)$	This paper

The result of [3] is valid both for oriented as well as unoriented rings. The result of [4] is valid for n-dimensional tori, where n is a constant (independent of the number of nodes). Moreover, the constant implicit in the bit complexity bound $O(N^{1+1/n})$ depends on n [4]. Hence this result cannot apply to the hypercube which has variable dimension n. Bit complexity bounds for non-faulty hypercubes are given in [8].

In this paper we give an algorithm for computing Boolean functions on asynchronous, anonymous hypercubes having bit complexity

$$O(N\lambda^2\delta_n(\gamma)^2 \log\log N).$$

Here N is the number of nodes, $n = \log N$. Since a connected, n-dimensional hypercube with polylogarithmic number of faulty components has diameter $O(\log N)$ (see [1]) we have an $O(N polylog(N))$ bit complexity for n-dimensional hypercubes with $1 \leq \gamma = polylog(N)$ faulty components.

Notice the different estimates on the bit complexity implied by the algorithm for hypercubes with exactly one faulty link versus hypercubes with exactly one faulty processor; in the former case the bit complexity is $O(N \log^2 N \log\log N)$ while in the latter $O(N \log^4 N \log\log N)$. At first glance it may also come as a surprise that the bit complexity in a faulty hypercube can be lower than the bit complexity in a non-faulty hypercube (e.g. this can be the case when there are no faulty processors and $\lambda < \log N/\sqrt{\log\log N}$). This however can be explained by the fact that in hypercubes with faulty links we can take advantage of asymmetries in the network topology in order to design algorithms with improved bit complexity. Thus our main algorithm takes advantage of "symmetry breaking" by distinguishing faulty links from non-faulty links.

2 Hypercubes with Non-faulty Processors

In this section we give algorithms for computing Boolean functions on a hypercube which does not have any faulty processors, i.e. $\pi = 0$. We indicate later how to extend our results to hypercubes with arbitrary faulty components. Our main theorem is the following.

THEOREM 1 *In a hypercube with at most λ faulty links, $\lambda \geq 1$, every computable Boolean function can be computed in $O(N\lambda^2\delta_n(\lambda)^2 \log\log N)$ bits.*

PROOF. The proof of the theorem is carried out in subsections 2.1, 2.2. Before giving a detailed account of the algorithm achieving the desired complexity we present a summary of the main steps of our construction. Let f be a given Boolean function. Each processor p is given an input bit b_p and the Boolean function f. Let $Input = < b_p : p \in Q_n >$. Under the assumptions of subsection 1.1 each processor p concerned executes the following algorithm: (1) determines whether or not the hypercube has a faulty link, (2) uses a "path-generation" algorithm in order to determine the location of the faulty links relative to itself, (3) uses an input collection mechanism in order to determine the entire input configuration $Input_p$, where $Input_p$ denotes p's view of $Input$, (in executing the algorith, the processors collect input bits in a manner specified by the protocol thus forming the view $Input_p$ associated with processor p), (4) determines whether or not the given function is computable on the given input (this step is actually performed only locally and hence does not contribute to the overall bit complexity) by checking an invariance condition on the given function f, (5) if f is computable then processor p outputs $f(Input_p)$.

2.1 Determining if there are any faulty links

The first step in our algorithm is to determine whether or not the hypercube has any faulty links. This follows from the following lemma.

LEMMA **2** *There is an algorithm with bit complexity* $O(N \log^2 N)$ *which detects whether or not the hypercube has any faulty links.*

PROOF. Let $\mathbf{0}$ = "I have no faulty links" and let $\mathbf{1}$ = "I have a faulty link". Each processor initializes the variable *value*. To determine whether there is a faulty link the processors execute an algorithm for computing the Boolean function OR_N by using the Boolean constants $\mathbf{0}, \mathbf{1}$ previously defined. If the output is $\mathbf{1}$ then there is a faulty link else there is no faulty link. The algorithm they execute is as follows.

> **Faultylink**
> **Algorithm for processor** p:
> **Initialize:** $value_p$;
> **for** $i := 1, \ldots, \log N$ **do**
>   **send** $value_p$ to all neighbors of p;
>   **receive** $value_q$ from all neighbors q of p;
>   **compute** $value_p := \mathrm{OR}(\{value_q : q \text{ is neighbor of } p\}) \vee value_p$;
> **od**;
> **output** $value_p$.

There are $\log N$ iterations of the **for** loop and in each iteration $\leq \log N$ bits are transmitted by each processor. Hence the bit complexity of the algorithm is $O(N \cdot \log^2 N)$. It remains to prove the correctness of the algorithm. We show that if there is a faulty link then every processor of the hypercube is at distance $\leq -1 + \log N$ from a faulty link. Indeed, let x be an arbitrary node. We want to show that x is at distance $\leq -1 + \log N$ from a faulty link. Let y be any node which is adjacent to a faulty link. There is a path $x_0 = x, x_1, \ldots, x_d = y$ of length $d \leq \log N$ connecting x to y in the non-faulty hypercube. Since y is adjacent to a faulty link it is clear that x is at distance $\leq -1 + \log N$ from a faulty link. Hence the lemma is proved. ■

If it turns out there is no faulty link then (assuming that the given Boolean function is computable in the network) they execute the algorithm of [8] which has bit complexity $O(N \log^4 N)$. Else they proceed to the next phase of our algorithm.

2.2 Path generation and input collection

The algorithm to be presented in this subsection requires the existence of faulty links. Therefore this phase is executed only if it turns out from the execution of the algorithm in subsection 2.1 that $\lambda \geq 1$. Let f be a Boolean function known to all processors of the (faulty) hypercube. We present the algorithm in three steps. The processors execute the following algorithm.

Main Algorithm ($\lambda \geq 1$):

1. PATH-GENERATION: The processors adjacent to faulty links become leaders and compute the configuration of the hypercube as follows. Let M be the set of faulty links. Let L be a processor adjacent to a faulty link. For each $x \in Q_n$ there are many paths connecting L to x. However L can choose a set of paths (in a canonical way) $\{p(L, x) : x \in Q_n\}$ such that $p(L, x)$ connects L to x, has length $\leq \delta_n(\lambda)$ and avoids the missing link(s). Each processor adjacent to a faulty link generates a set of paths, one path for each processor of the hypercube. In generating paths the processor takes into account its current knowledge of the position of the set of faulty links (which is only a subset of the set of all faulty links). Each such path is transmitted to its destination node along the sequence of links determined by this path. If during transmission of this path a faulty link is encountered then the corresponding processor adjacent to this faulty link sends back (along this same path but in the reverse direction) to the originating processor a complete list of its missing links. Based on this information each processor adjacent to a link in M updates its current list of faulty links and generates a new set of paths which avoid the previously encountered faulty links. Now iteration of this procedure continues as long as new faulty links are found. (Notice that nowhere in this algorithm do the processors need to know an upper bound on the number of faulty links. The iterated procedure terminates execution when no new faulty links are found.) After execution of this algorithm all processors receive a complete path from each processor adjacent to a link in M.

Since each iteration of this algorithm generates a new collection of paths by "eliminating" newly encountered faulty links and since there are at most λ faulty links it is clear that after at most λ iterations all processors will receive paths from all processors adjacent to processors with faulty links. The bit complexity of this algorithm depends on the length of the paths which are created during the execution of the λ iterations of this algorithm (in this instance the paths have maximal possible length $\delta_n(\lambda)$) and can be computed as before. There are $\leq 2\lambda$ processors adjacent to the λ faulty links. Paths can be coded with $\delta_n(\lambda) \log \log N$ bits (all that is needed is the sequence of labels traversed by the path). Each path is transmitted at a distance $\leq \delta_n(\lambda)$. Each iteration of the algorithm involves $\leq 2\lambda$ processors adjacent to a faulty link in M. Hence each iteration of the algorithm involves the transmission of at most $O(N\lambda\delta_n(\lambda)^2 \log\log N)$ bits. Since the number of iterations is $\leq \lambda$ the actual bit complexity of this step will be $O(N\lambda^2\delta_n(\lambda)^2 \log\log N)$ bits.

2. INPUT-COLLECTION: For each x, and each L adjacent to a link in M, processor x sends its input bit b_x together with its "identity" $p(L,x)$ to L in the reverse direction along path $p(L,x)$ ($p(L,x)$ is the path computed in step 1). Now L has a view of the entire input configuration of the hypercube, say I_L, and can compute $f(I_L)$. The bit complexity of this step is $O(N\lambda\delta_n(\lambda)\log\log N)$.

3. Let F be the set of processors which are adjacent to faulty links. By executing the above algorithm each processor $L \in F$ computes its "view" I_L of the given input configuration. In particular, each $L \in F$ will know the view $I_{L'}$ of all processors $L' \in F$. Hence all processors $L \in F$ may execute the invariance test

$$f(I_L) = f(I_{L'}), \text{ for all } L, L' \in F. \tag{1}$$

If (1) is true each processor $L \in F$ computes $f(I_L)$ and transmits it to all processors of the hypercube along the paths previously specified. Finally, $f(I_L)$ is the output bit of each processor of the hypercube. If on the other hand (1) is false then the processors $L \in F$ will transmit to all processors of the hypercube that f is not computable on the given input. Clearly, test (1) is local to the processors and does not contribute to the overall bit complexity of the algorithm. The bit complexity of this step is $O(N\lambda\delta_n(\lambda)\log\log N)$.

Notice that nowhere in this algorithm did we have to assume that the processors have identities. All identities used there were generated by the algorithm and were relative to a particular leader. In addition the processors execute identical algorithms given identical input data. This completes the proof of Theorem 1. ■

An interesting observation concerns the size of the input data of a processor. In computing Boolean functions the input to a processor was assumed to be a bit. However, if the size of the iput data of a processor is $\leq s$ bits then the contribution to the overall bit complexity of the input collection step is at most $O(Ns\lambda\delta_n(\lambda)\log\log N)$. In particular, the bit complexity stated in theorem 1 remains valid, even if the size of the input data is up to $O(\lambda\delta_n(\lambda)$ bits.

2.3 Estimates depending on the number of faults

Theorem 1 raises the problem of studying $\delta_n(\lambda)$ as a function of λ. Results of B. Aiello and T. Leighton in [1] show that an n-dimensional hypercube with $n^{O(1)}$ worst-case faults can simulate the fault-free n-dimensional hypercube Q_n with only constant slowdown. In particular, this implies that $\delta_n(\lambda) = O(n)$, for $\lambda = n^{O(1)}$. As a consequence we obtain the following result for hypercubes with polylogarithmic number of faulty links.

THEOREM **3** *The bit complexity of computing Boolean functions on a hypercube with polylogarithmic number of faulty links (i.e.* $\lambda = (\log N)^{O(1)}$*) is*

$$\begin{cases} O(N \log^4 N) & \text{if } \lambda = 0 \\ O(N\lambda^2 \log^2 N \log\log N) & \text{if } \lambda > 0. \end{cases}$$

PROOF. If $\lambda = 0$ then by [8] the bit complexity of computing the function f is $O(N \log^4 N)$. If $\lambda \geq 1$ then applying Theorem 1 we see that the bit complexity of computing a Boolean function is $O(N\lambda^2 \delta_n(\lambda)^2 \log\log N)$. Since the number of faulty links is $n^{O(1)}$ we have that $\delta_n(\lambda) = O(n)$. Hence the combined bit complexity is

$$O(N \log^2 N \log^2 N \log\log N),$$

as desired. ■

Thus we see that $\log N/\sqrt{\log\log N}$ is the threshold number of faulty links for which the bit complexity of computing Boolean functions on an N node hypercube exceeds the bit complexity in a non-faulty hypercube.

A different, but simple, argument can be used to determine the complexity of computing the OR_N function on N inputs. We have the following theorem.

THEOREM **4** *There is an algorithm with bit complexity* $O(N \log^2 N)$ *for computing the function* OR_N *on a faulty hypercube, provided that the number of faults is* $\lambda = n^{O(1)}$.

PROOF. The idea of the algorithm is rather simple. Each processor sends its initial input value to all its neighbors. After receiving a value from its neighbors it compares the values it receives to the value it already has. Every processor executes these steps $\delta_n(\lambda)$ times. Eventually every input value to a node of the network will be distributed and accounted for by every other processor. The rest of the proof now follows from Lemma 2 and the above mentioned result of B. Aiello and T. Leighton in [1]. ■

If the condition $\lambda = n^{O(1)}$ is not valid then Theorem 4 is false. The reason is the following. Imagine the faulty links are such that the remaining non-faulty links form a ring on N vertices. Then computing OR_N requires $\Omega(N^2)$ bits (see [3]).

3 Determining the Computability of f

Condition (1) tests the computability of the Boolean function f on the given input. However, in the case where the set F of nodes which are adjacent to the set of faulty links $\{l_1, \ldots, l_\lambda\}$ is transitive (i.e. for any two processors $L, L' \in F$ there exists an automorphism $\phi \in Aut(Q_n[l_1, \ldots, l_\lambda])$ (by definition, $Aut(Q_n[l_1, \ldots, l_\lambda])$ is the group of automorphisms of the graph $Q_n[l_1, \ldots, l_\lambda]$ which preserves edge labels) such that $\phi(L) = L'$) we can in fact test whether the given function f is computable on all inputs. This is done by checking whether or not the given Boolean function f is invariant under all automorphisms of the network. Indeed, assume the function f is computable on the hypercube $Q_n[l_1, \ldots, l_\lambda]$. Let $I =< b_p : p \in Q_n >$ be an input configuration and let ϕ be an automorphism of $Q_n[l_1, \ldots, l_\lambda]$. Let I^ϕ denote the configuration $< b_{\phi(p)} : p \in Q_n >$. Let p be a node and q its image under ϕ, i.e. $q = \phi(p)$. But is is clear that $f(I) = f(I^\phi)$ since p, q execute the same algorithm given identical input data. Conversely, assume that f is invariant under all automorphisms of the above faulty hypercube. The previous input collection algorithm shows that for any processors $L, L' \in F$ the views $I_L, I_{L'}$ generated by the algorithm are identical up to automorphism. Notice that the condition on the transitivity of the set F is always satisfied when $\lambda = 1$. Hence we have proved the following theorem.

THEOREM **5** *Assume that the set of processors adjacent to the faulty links of the connected hypercube $Q_n[l_1, \ldots, l_\lambda]$ is transitive. Then a Boolean function f is computable in $Q_n[l_1, \ldots, l_\lambda]$ if and only if it is invariant under all the automorphisms in $Aut(Q_n[l_1, \ldots, l_\lambda])$. Moreover the bit complexity of computing all such Boolean functions is $O(N\lambda^2\delta_n(\lambda)^2 \log\log N)$.* ■

To check efficiently the invariance of a Boolean function under all automorphisms of the network the processors execute locally the algorithm specified in Lemmas 6, 7. This requires computing the group of automorphisms of the corresponding hypercube. Consider the *bit-complement* automorphisms that complement the bits of certain sets of components components, i.e. for any set $S \subseteq \{1, \ldots, n\}$ let $\phi_S(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, where $y_i = x_i + 1$, if $i \in S$, and $y_i = x_i$ otherwise (here addition is modulo 2). Let F_n denote the group of bit-complement automorphisms of Q_n. Let $Aut(Q_n[l_1, \ldots, l_\lambda])$ be the set of automorphisms of $Q_n[l_1, \ldots, l_\lambda]$ that preserve the labels of its links.

LEMMA **6** *Let $l_1, \ldots, l_\lambda$ be arbitrary links of the hypercube Q_n. If the network $Q_n[l_1, \ldots, l_\lambda]$ is connected then $Aut(Q_n[l_1, \ldots, l_\lambda])$ is a vector subspace of $Aut(Q_n)$ of dimension $O(\log \lambda)$ which has at most $2\lambda^2$ elements. Moreover these elements can be computed in time $O(\lambda^3)$.*

PROOF. First we show that $Aut(Q_n[l_1, \ldots, l_\lambda]) \leq Aut(Q_n)$. As in [8] we can show that all the automorphisms of $Q_n[l_1, \ldots, l_\lambda]$ must be of the form ϕ_S, for some $S \subseteq \{1, 2, \ldots, n\}$. Indeed, let ϕ be an arbitrary automorphism and let x, y be arbitrary nodes in $Q_n[l_1, \ldots, l_\lambda]$. We claim that $\phi(x) + \phi(y) = x + y$ (here addition is componentwise modulo 2). To see this take a path, say $x_0 := x, x_1, \ldots, x_k := y$, joinning x to y. Since by definition ϕ preserves labels we must have that $\phi(x_i) + \phi(x_{i+1}) = x_i + x_{i+1}$, for all $i < k$. Hence the claim follows by adding these inequalities modulo 2. Now if $\phi(0^n) = (p_1, \ldots, p_n)$ then it is clear that $\phi = \phi_S$, where $S = \{1 \leq i \leq n : p_i \neq 0\}$.

Next we give an algorithm for computing the elements of the automorphism group $Aut(Q_n[l_1, \ldots, l_\lambda])$. Put $L = \{l_1, \ldots, l_\lambda\}$. The automorphisms of the faulty hypercube $Q_n[l_1, \ldots, l_\lambda]$) act naturally on the set of links L in the following way: if $l = xy$ then $\phi(l) = \phi(x)\phi(y)$. For this action it is easy to see that for all $l, l' \in L$ there exist at most two automorphisms, say $\phi_{l,l'}, \psi_{l,l'}$, which map l into l'. This implies that $|Aut(Q_n[l_1, \ldots, l_\lambda])| \leq 2\lambda^2$. Since the automorphisms of $Q_n[l_1, \ldots, l_\lambda]$ are precisely the automorphisms of Q_n which leave the set L invariant we are lead to the following algorithm whose output S is the set of automorphisms of $Q_n[l_1, \ldots, l_\lambda]$.

Algorithm for computing the automorphism group
begin $S := \emptyset$;
for $l, l' = l_1, \ldots, l_\lambda$ **do**
  compute $\phi_{l,l'}, \psi_{l,l'}$;
  if $\phi_{l,l'}(L) \subseteq L$ **then** $S := S \cup \{\phi_{l,l'}\}$;
  if $\psi_{l,l'}(L) \subseteq L$ **then** $S := S \cup \{\psi_{l,l'}\}$;
  fi;
  od;
output S.

The output S of the above algorithm is the desired group of automorphisms of $Q_n[l_1, \ldots, l_\lambda]$ since $Aut(Q_n[l_1, \ldots, l_\lambda]) = \{\phi \in Aut(Q_n) : \phi(L) \subseteq L\}$. Clearly, the complexity of this algorithm is $O(\lambda^3)$. ■

LEMMA **7** *There is an algorithm computing the group $Aut(Q_n[l_1, \ldots, l_\lambda])$ in $O(N\lambda^2\delta_n(\lambda)^2 \log\log N)$ bits.*

PROOF. Using the first part of the algorithm of subsection 2.2 the processors adjacent to faulty links can compute the missing links of the entire hypercube. At the end of this algorithm "only" the processors adjacent to faulty links can compute the automorphism group of $Q_n[l_1, \ldots, l_\lambda]$ using the algorithm of Lemma 6. These processors now compute a basis of the automorphism group consisting of $O(\log \lambda)$ automorphisms and transmit this to the rest of the processors. This proves the lemma. ■

We also mention an interesting observation concerning the size of the automorphism group of the faulty hypercube.

THEOREM **8** *If for some i the number of faulty links labeled i is odd then*

$$|Aut(Q_n[l_1, \ldots, l_\lambda])| \leq 2.$$

PROOF. Let $G = Aut(Q_n[l_1, \ldots, l_\lambda])$ and assume that G is not the identity group. For $1 \leq i \leq n$ define

$$\begin{aligned} L &= \{l_1, \ldots, l_\lambda\} \\ L_i &= \{l \in L : \text{ label of } l \text{ is } i\}. \end{aligned}$$

For each i every automorphism in G permutes L_i. Now we can show that for all $l \in L$, $|G_l| \leq 2$, where G_l is the group of automorphisms fixing l. Indeed, if

$\phi \neq id$ and $\phi(l) = l$ then $\phi = \phi_{\{i\}}$ where i is the label of link l. In fact, if the label of l is i then

$$G_l = \begin{cases} < id > & \text{if } \phi_{\{i\}} \notin G \\ < \phi_{\{i\}} > & \text{if } \phi_{\{i\}} \in G. \end{cases}$$

If $l^G := \{\phi(l) : \phi \in G\}$ denotes the orbit of l in G then it follows from the identity $|G_l| \cdot |l^G| = |G|$ (see Wielandt [9]) that

$$|l^G| = \begin{cases} |G| & \text{if } \phi_{\{i\}} \notin G \\ |G|/2 & \text{if } \phi_{\{i\}} \in G \end{cases}$$

Since the orbits partition the orbit space L_i, it follows easily that

$$|L_i| = (\# \text{ of orbits of } G \text{ acting on } L_i) \cdot \begin{cases} |G| & \text{if } \phi_{\{i\}} \notin G \\ |G|/2 & \text{if } \phi_{\{i\}} \in G. \end{cases}$$

Now the conclusion of the theorem follows from the fact that G is a group whose order is a power of 2 (recall that by lemma 6, G is a vector subspace of $Aut(Q_n)$) and $|L_i|$ is odd. ■

Notice that $\phi_{\{i\}}$ is the only automorphism in G that has "fix-points" when acting on L_i. In particular, if for all S with $\phi_S \in G$ we have that $|S| \geq 2$ then for all i $|L_i|$ is even.

4 Hypercubes with Faulty Components

So far we have considered the case of hypercubes having only faulty links. However, it is straightforward how to adapt the Path-generation and Input-collection algorithms presented in section 2 to the case of hypercubes whose faulty components may be links and/or nodes. If a node is faulty then all its adjacent links are interpreted as faulty. The Path-generation algorithm is initiated by non-faulty processors which are adjacent to faulty links (there are $\leq 2\lambda$ such processors) and the iterated procedure is repeated $\leq \lambda$ times. Thus we can prove the following theorem.

THEOREM **9** *In a hypercube with γ faulty components exactly λ of which are faulty links, $\lambda \geq 1$, the bit complexity of computing Boolean functions is*

$$O(N\delta_n(\gamma)^2\lambda^2 \log\log N). \blacksquare$$

5 Conclusion

We have presented algorithms for distributed computation on anonymous asynchronous hypercubes with faulty components. Our algorithms rely on the possibility of distinguishing faulty links from non-faulty ones and are based on broadcasting and path generation. The hypercubes may be faulty but the faults may occur only before the start of the computation. An interesting problem would be to design more "adaptive" algorithms that allow for faults to occur at different parts of the computation. In addition, very little is known on the optimality of the algorithms presented.

Acknowledgements

Many thanks to Danny Krizanc and Hisao Tamaki for useful advice and the referees for useful comments.

References

[1] B. Aiello and T. Leighton, "Coding Theory, Hypercube Embedding and Fault Tolerance", Proceedings of 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, 1991, 125 - 136.

[2] D. Angluin, "Local and Global Properties in Networks of Processors", 12th Annual ACM Symposium on Theory of Computing, 1980, 82 - 93.

[3] H. Attiya and M. Snir and M. Warmuth, "Computing on an Anonymous Ring", Journal of the ACM, 35 (4), 1988. Short version has appeared in proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computation, 1985, 845 - 875.

[4] P. W. Beame and H. L. Bodlaender, "Distributed Computing on Transitive Networks: The Torus", 6th Annual Symposium on Theoretical Aspects of Computer Science, STACS, 1989, B. Monien and R. Cori, editors, Springer Verlag Lecture Notes in Computer Science. 294-303.

[5] B. Becker and H.-U. Simon, "How Robust is the n-Cube?", Proceedings of IEEE 27th Annual Symposium on Foundations of Computer Science, 1986, 283 - 291.

[6] M-S. Chen and K. G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers", IEEE Transactions on Computers, 39 (12), December 1990, 1406 - 1416.

[7] J. Hastad and T. Leighton and M. Newmann, "Reconfiguring a Hypercube in the Presence of Faults", Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, 274 - 284.

[8] E. Kranakis and D. Krizanc, "Distributed Computing on Anonymous Hypercube Networks", Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas, Dec. 2-5, 1991, 722 - 729.

[9] H. Wielandt, "Finite Permutation Groups", Academic Press, 1964.