Self-Diagnosis of APEmille

Federico Aglietti[1], Enrico Centurioni[2], Stefano Chessa[3], Ivan D'Auria[3], Francesco Franzinelli[2], Piero Maestrini[3,4], Andrea Michelotti[1], Igor Pagliai[3], Raffaele Tripiccione[2]

[1] Istituto Nazionale di Fisica Nucleare, Sezione di Roma 1
[2] Istituto Nazionale di Fisica Nucleare, Sezione di Pisa
[3] Dipartimento di Informatica, Università di Pisa
[4] Istituto di Elaborazione dell'Informazione, CNR, Pisa

ABSTRACT

The APEmille, the third evolution of the APE family of SIMD machines, is structured as a three-dimensional array of processors. In its largest configuration, the number of processors is 4096. Its typical application range covers massive computations (e.g., those needed to solve some problems in phisics research), which may require as many as 10^{17} floating point operations. Given the long time needed to complete such jobs, the machine should be able to tolerate the occurrence of multiple faults during the job execution. To this purpose, self-diagnosis capabilities have been incorporated in its design, using an approach inspired by a family of algorithms recently introduced to perform the system-level diagnosis of regular architectures. The machine is partitioned into three subsystems, each structured as a three dimensional array, which are diagnosed separately using slightly different variants of the same diagnosis algorithm. The system units are tested by means of comparisons, either concurrently with the job execution or during special diagnosis sessions. The strategy to test the units and the diagnosis algorithms are described, and the diagnosis correctess and completeness are evaluated both theoretically and experimentally.

Keywords: Fault-tolerance, system-level diagnosis, self-diagnosis, SIMD machines, grid interconnection.

1. INTRODUCTION

The APEmille machine, the third evolution of the APE family [1], is currently in the phase of detailed design. The basic architecture of APEmille [2] consists of a set of processors, also called *processing elements* (*PE's*), each having its own private memory bank, which execute exactly the same instruction at each machine cycle. With the current design, the maximum number of processing elements is 4096. Instruction scheduling is accomplished by a functionally different unit, known as the *instruction processor*. In the traditional taxonomy of parallel computing [3], APEmille would be classed as a Single-Instruction, Multiple-Data *(SIMD)* computer.

Simple *communication units* connect the processing elements, which (in principle) are assembled at the sites of a three-dimensional mesh. The mesh links each PE with its six nearest neighbor PE's. All three directions are *wrapped around*, forming a three-dimensional torus. The communication units collects data blocks from the PE's (under program control) and move them to their destination PE's, performing a limited set of simple routing operations, to be described later. There is also a separate network that logically merges (by performing the logic *and* and the logic *or*) one condition code bit calculated by each PE and delivers the result to the instruction processor.

There are no specific interface structures, able to perform I/O operation at the basic architecture level, since APEmille is essentially an attached processor, connected to an host computer, responsible for all I/O issues. The actual (host based) I/O structure of the present APEmille implementation will be discussed in the next section.

The APE family of parallel processors was originally conceived to be solve some massive computational problems of physics research, like the *Lattice gauge Theory (LGT)* [4,5]. *LGT* is a formulation of the standard model of interactions between the elementary components of matter (so called *quarks*), which is especially suited for numerical modelling. In this problem, as well as in similar problems in the same or in different research areas, massive computations using $\sim 10^{17}$ floating point operations are not uncommon. This would require a computer power of 10*GFlops* to be sustained for more than 3 months. A favourable feature of the numerical algorithms applied in this field is that they are trivially parallelizable most of the times. Indeed, straightforward domain decomposition can be applied to the relevant data structures and the same set of operations must be applied to all domains. Load balancing is also almost automatic, and the pattern of communications favors connections between topologically nearest neighbor domains. These features, combined with the huge computing requirements described above have started, as early as the mid eighties, the development of several dedicated parallel processors. The APE family has been one of the most significant among such development.

Given the domain of application of the APEmille machine, which implies huge computations which require very long time to be completed even in a massively parallel environment, we may expect that multiple faults occur during any single job execution. The machine should be able to tolerate, or to overcome, such faults, in order to ensure the correcness of computation and to safeguard the resource investment in the job assignment.

The massively parallel architecture of APEmille and its regular structure have suggested that *system level diagnosis* could be exploited to identify the faulty components of the system, at least at the level of the smallest replaceable units. The System Level Diagnosis, in which the individual units participate in self-diagnosis using the outcome of a set of reciprocal tests, was introduced by Preparata, Metze and Chien in their seminal paper [6], and has been deeply investigated in many subsequent papers. A comprehensive survey of this theory, as well as of related problems, is reported in [7]. Recent contributions [8,9,10,11] deal with the application of system level diagnosis to massive parellel architectures featuring regular interconnection patterns, such as two or three dimensional arrays, of which APEmille is a notable example.

To the purpose of identifying the faulty units, the self-diagnosis of APEmille is activated periodically while the job execution proceeds, using the outcomes of tests which are performed concurrently with the job. This self-diagnosing process covers some critical units of the system. If any fault is detected, the job execution is halted to start a *diagnosis session*, in which all the remaining units perform self-diagnosis. The diagnostic session, which uses the outcomes of tests which cannot be performed concurrently with the job, is also started upon the completion of any job.

Based on the results of self-diagnosis, all the replaceable units which were identified as faulty (if any), are replaced to reestablish the correct machine behavior. Of course, the job which was harmed by the occurrence of faults needs to be restarted. As it will clarified in the following, the output of self-diagnosis will usually provide more detailed information, enabling identification of faults within functional units smaller than, and contained into, the replaceable units. This additional information may be exploited as a guideline to repair the replaceable units, as well as to evaluate reliability of functional units.

2. THE ARCHITECTURE OF APEmille

As it is apparent from the sketchy description presented in the introduction, there are three basic elements in the APEmille architecture; namely, the *instruction processor*, also called *Tmille,* the *communication unit*, also called *Cmille*, and the *processing element*, also called *Jmille*.

The Tmille instruction processor has the basic task of controlling program flow. It can therefore be regarded as a simple CPU, optimized for control rather than manipulation of data operands. It contains however an arithmetic-logic-unit, used to manipulate its own control variables (a typical example of such a variable is the index of a loop). These are stored in the Tmille processor's private memory bank.

The Jmille PE's are the basic components of the APEmille system. Each PE is a combination of a data processor and its memory bank, and its structure is biased toward efficient floating point computation. It is based on a large multi-port register file, where data from the private memory bank can be loaded. All data operations are register to register and include a somewhat standard set of floating point arithmetic operations in single and double precision as well as a standard set of arithmetic and logic operations on integer and boolean values. Memory addresses are evaluated by the PE's.

The Cmille communication unit has both a simple topological structure and a simple set of capabilities. As already remarked, the communication network connects nearest neighbor elements of a three-dimensional mesh. In principle, for each Jmille processor there exists an associated Cmille unit, which has an interface with the processor and six links stretching to the neighbor Cmille units.

The communication structure is shaped by the fact that the machine exhibit SIMD behavior. This means that at some given machine cycle all the Jmille processors start a communication process to a distant neighbour. The process follows this basic pattern:

- each Jmille processor sends a data block to the associated Cmille unit;
- all Cmille units route data to their destinations;

- data arriving at the destination Cmille units are passed to the associated Jmille processors.

One basic point in this process, specific of the SIMD structure of APEmille, must be stressed. There is an apparent contraddiction between the fact that data blocks may take different time to reach different destination, while all node execute all operations (inclusive of receiving data from the communication units) sinchronously. This problem is solved by delivering data only when all of them have reached destination. In practice, the network of Cmille units produces a *kill* signal that stops machine execution until data are ready to be loaded.

Routing capabilities are very simple, and divided in three groups:

- the basic routing style is callled *rigid routing*. In this case a single destination address is given to the network in the form of a 3-ple, indicating the relative offsets $(\Delta X, \Delta Y, \Delta Z)$ between each source and destination nodes. In other terms, the whole network is translated rigidly by a given vector (hence the name *rigid*).
- a second routing style is a *broadcast*. In this case one node is tagged as "broadcaster". Data coming from the "broadcaster" are delivered to each node of the machine. Broadcast can be three-dimensional *(3-d)*, or two- dimensional *(3-d)*, or one-dimensional *(3-d)*. In *3-d* mode, data are delivered to each node of the machine. In *2-d* mode, the array is divided geometrically in *2-d* planes, each plane having its own broadcaster. The *1-d* mode is obviously similar to the previous one.
- in *soft* routing style, each node provides a destination address for its own transmitted data block. This routing style, which is not supported by the current APEmille implementation, although it is planned for a forthcoming version, will not be described in more details in the rest of this paper.

The basic APEmille architecture which has been presented so far overlooks some relevant details. In fact, the present implementation of APEmille differs from the above picture in three major points:

- Jmille processors are clustered in blocks of eight, logically arranged at the edges of a three dimensional unit cube.
- there is only one Cmille communication unit in charge of all the Jmille processors belonging to the same cluster.
- there is one Tmille instruction processor for each cluster of Jmille processors. Tmille processors belonging to different clusters execute exactly the same program and are initialized with the same data values; so, pending failures, they behave exactly as one single Tmille processor distributing the instruction stream to the whole array.

Each cluster of 8 Jmille processors, togheter with the associated Tmille instruction processor and Cmille communication unit, is physically assembled on one printed circuit board. This system is in principle a stand-alone system, connected by point-to-point links with the adjoining boards to build up a larger machine. There is also a communication channel connecting the boards to the *host computer* (see later).

When comparing the basic architecture with its implementation, an obvious difference is seen in the communication network. In practice, eight logically independent Cmille units are physically merged into one real unit, whose task is that of performing communication operation in such a way that its actual implementation is transparent for the user. Consider, for instance, a rigid communication with offset (1,0,0). The data word stemming from Jmille processor (0,0,0) of the cluster (we label PE'swith their cartesian coordinates) will never leave the merged Cmille unit in order to reach destination at (1,0,0). On the other hand, the same rigid communication starting from location (1,0,0) involves transfer to a neighbor cluster.

The second basic addition to the basic architecture is the interface to the host computer. This is accomplished in the form of a channel oriented link which plugs into the Tmille processors. The host sends data blocks onto the link that go either to the Tmille processors' memories or to the memory bank of any one of the Jmille processors. In the latter case, the relevant data are routed from the Tmille processor to the Cmille communication unit and then to the destination Jmille processor. The Cmille-Jmille interface used in this case is the same as the one used for standard network operation. This path can of course be taken also in the opposite direction to move data from any Jmille processor to the host computer. When accessing Cmille processors, the host computer is also able to download the test outcomes which are stored in a register bank within these units.

3. THE DIAGNOSIS ALGORITHM

In this section, we introduce a general algorithm to diagnose regular systems where $n=x \cdot y \cdot z$ units are arranged in a three-dimensional array, with toroidal interconnections. This algorithm derives from similar algorithms [10, 11] which were introduced to diagnose two-dimensional arrays of processors, with which it shares the property of providing a diagnosis which is provably correct and, with high probability, complete or almost complete[1]. These algorithms for two-dimensional arrays are also related to other diagnosis algorithms [8,9] whose diagnosis, however, cannot be proved to be correct, although its completeness is highly probable. The algorithms exploited in the diagnosis of APEmille, to be presented in the next section, derive immediately from this general algorithm.

We identify each unit with a coordinate triple (i,j,l), where $0 \le i \le x$-1, $0 \le j \le y$-1 and $0 \le l \le y$-1. Unit (i,j,l) is connected to units $((i \pm 1) \bmod x, j,l)$, $(i, (j \pm 1) \bmod y ,l)$, $(i, j,(l \pm 1) \bmod z)$. Each connection is assumed to be bi-directional. Units (i,j,l) and (i',j',l') which are interconnected are said to be *adjacent,* denoted $(i, j,l) \leftrightarrow (i',j',l')$.

We assume that any two adjacent units test each other. The test is executed by comparing the output sequences of the units undergoing the test, when they are assigned the same job. We assume that any fault affecting a single unit causes at least one mismatch, while mismatches may fail to occur when both units are faulty. For the sake of simplicity, the comparator is assumed to be fault-free. This hypothesis will be removed when deriving the algorithms exploited by APEmille. Given $(i,j,l) \leftrightarrow (i',j',l')$, we use notation $(i,j,l) \xleftrightarrow{\gamma} (i',j',l')$ to denote the comparison of (i',j',l') and (i,j,l) with outcome γ (γ is 0 or 1). Using this notation, the test invalidation rule (which is closely related to the PMC model) is reported in Table 1.

(i,j,l)	(i',j',l')	γ
G	G	0
G	F	1
F	G	1
F	F	0 or 1

Table 1: Invalidation rule in the PMC model (F= faulty; G= non-faulty)

In order to perform self-diagnosis we need the outcomes of all the possible comparisons, collectively denoted as the *syndrome*. The syndrome is collected by the *diagnoser,* a reliable external processor which executes the diagnosis algorithm.

The objective of the algorithm is to identify the state (faulty or non-faulty) of the largest possible number of units. Given any syndrome σ, the diagnosing algorithm proceeds through three steps. The first step (*local diagnosis*), performs preliminary classification of units. The second step identifies a non-empty *Fault-Free Core;* i.e., a set of units which are known to be non-faulty. The third step (*augmentation*) exploits the reliable outcomes of comparisons between units in the Fault-Free Core and adjacent units not in this set, to augment the Fault-Free Core itself and the define a set of units identified as faulty.

Local diagnosis is based on comparison of adjacent units: every unit is compared with, its six neighbours. Every unit is classified either D or Z. Units D (*dual* units) are defined in disjoint pair with the property that, for every pair, at least one unit must be faulty. All units which cannot be classified D are labelled Z and, at this stage of the algorithm, their state remains unidentified.

Classification of units is performed according to the following rules, which derive immediately from the invalidation rule defined in Table 1:

a) for each unit pairs, (i,j,l) and (i',j',l') such that $(i,j,l) \xleftrightarrow{1} (i',j',l')$, at least one unit in the pair (i,j,l), (i',j',l') must be faulty. Provided that neither units has been labelled as yet, both units (i,j,l) and (i',j',l') are classified D. Observe that this rule guarantees that D-units be introduced in

[1] The diagnosis is said to be correct if every unit diagnosed as faulty is actually faulty and every unit diagnosed as non-faulty is actually non-faulty. It is said to be complete if every unit is diagnosed as either faulty or non-faulty. A correct diagnosis might be incomplete and, conversely, a complete diagnosis might be not correct.

disjoint pairs.

b) All units which remain not labelled are classified Z. Their state cannot be identified at this stage of the algorithm; however, if (i,j,l) and (i',j',l') are adjacent Z-units, then necessarily $(i,j,l) \xleftrightarrow{0} (i',j',l')$, and both units must be in the same state (faulty or non-faulty).

In the second step, all pairwise adjacent Z-units are combined in contiguous sets, called *aggregates*. Z-units which do not combine are considered to be singleton aggregates. Letting α be the maximum of the aggregate cardinalities, the *Fault-Free Core* (denoted $\mathcal{G}$) is defined as the union set of all the aggregates of cardinality α. The Fault-Free Core is a non-empty set of non-faulty units under the conditions to be stated in the following of this section.

As the Fault-Free Core has been identified, the set $\mathcal{N}$ of all units is partitioned into set $\mathcal{G}$, sets $\mathcal{A}'_1,\mathcal{A}'_2,\ldots,\mathcal{A}'_h$, with $h \geq 0$ (each an aggregate of cardinality less than α) and set $\mathcal{D}$ of all units classified D. Observe that all units in sets $\mathcal{A}'_1,\mathcal{A}'_2,\ldots,\mathcal{A}'_h$ and $\mathcal{G}$ must be classified Z. Units in set $\mathcal{G}$ are known to be non-faulty. For every $\mathcal{A}'_i$, all units are in the same state which, however, remains unknown. The state of units in set $\mathcal{D}$ is also unidentified, although it is known that at least half of them are faulty.

The third step (*augmentation*) of the algorithm augments set $\mathcal{G}$ and defines a set $\mathcal{F}$ of units which are known to be faulty, by identifying the state of the largest possible number of units in sets $\mathcal{A}'_1,\mathcal{A}'_2,\ldots,\mathcal{A}'_h$ and $\mathcal{D}$. State identification exploits the reliable outcomes of comparisons between units in the Fault-Free Core and adjacent units not in this set, as well as the possible existence of *zero-ancestors* of faulty units. A *zero-ancestors* of a faulty unit (i,j,l) is a unit (i',j',l'), adjacent to (i,j,l), such that $(i,j,l) \xleftrightarrow{0} (i',j',l')$,. From the invalidation rule of Table 1, it is immediate that unit (i',j',l') must also be faulty. Units comparing with outcome 0 with units in the Fault-Free Core are added to the Fault-Free Core itself. Units which are tested with outcome 1 by units in the Fault-Free Core, as well as units (not in the Fault-Free Core) which are zero-ancestors of units in set $\mathcal{F}$, are added to set $\mathcal{F}$ itself.

The diagnosis correctness relies on the existence of non empty set of non-faulty units. In fact, the diagnosis algorithm is actually able to identify a Fault-Free Core under the conditions stated below. In the following statement, set $\mathcal{N}_f \subseteq \mathcal{N}$ denotes the set of units in the system which are actually faulty. It should be recalled that, after the Fault-Free Core has been identified, the array is partitioned in the fault-free core $\mathcal{G}$, a number of aggregates $\mathcal{A}'_1,\ldots,\mathcal{A}'_h$, each of cardinality less than α, and set $\mathcal{D}$ of the dual units.

Theorem 1. *Given syndrome σ, the units in the Fault-Free Core are non-faulty, provided $\#\mathcal{N}_f < T_\sigma$ where: $T_\sigma = \frac{1}{2}\#\mathcal{D} + \alpha$, and α is the maximum of the aggregate cardinalities.*

Proof. Preliminary, observe that:

- the number of faulty units in set $\mathcal{D}$ is at least $\frac{1}{2}\#\mathcal{D}$;
- assuming $\#\mathcal{N}_f < n/2$, there must exist at least one aggregate,
- for every aggregate, all units must be in the same state (faulty or non-faulty).

This considered, let $\mathcal{A}$ be any aggregate with $\#\mathcal{A}=\alpha$ and assume that units in $\mathcal{A}$ are faulty: it follows that $\#\mathcal{N}_f$ is at least $\frac{1}{2}\#\mathcal{D} + \alpha$, and this contradicts the hypothesis. ❑

The bound T_σ defined in Theorem 1 depends on the cardinalities α and $\#\mathcal{D}$, which, in turn, are dependent on the actual syndrome σ. A lower bound to T_σ; that is, a bound holding for any possible syndrome, is stated by Theorem 2.

Theorem 2. *For any syndrome, all units in the Fault-Free Core are non-faulty, provided $\#\mathcal{N}_f \leq T(n)$*

$$= \min_{0<\alpha\leq n} n\frac{w/6}{2(\alpha + w/6)} + \alpha, \text{ with } w = 2h^2 - 2\left\lfloor \frac{h^3 - h^2}{\sqrt[3]{n} - 1} \right\rfloor + 2 + 2\sqrt{2h^2 - 1} \text{ and } h = \sqrt[3]{\alpha}.$$

The proof to this theorem is omitted for the sake of brevity. The technique used to derive the bound $T(n)$ is similar to the one which was presented in [10,11] for the case of two-dimensional arrays. The *boundary* of any aggregate $\mathcal{A}$ is defined as the set of those units not in $\mathcal{A}$, which are adjacent to at least one unit in $\mathcal{A}$. All units in the boundary of $\mathcal{A}$ must be classified D, since they failed to aggregate. It follows that a lower bound to $\#\mathcal{D}$ can be evaluated by determining the union set of the boundaries of all the aggregates. In turn, a lower bound to the cardinality of this set can be found using geometrical techniques similar to those exploited in [10,11], although the three dimensional (rather than two-dimensional) structure of the array implies analytical difficulties that could only be solved by introducing rougher approximations. In theorem 2, bound $T(n)$ is expressed as the minimum of a function, which cannot be expressed analytically. However, $T(n)$ is easily evaluated using numerical techniques, as reported in Table 2 for a few values of the array size. Table 2 also reports the aggregate cardinality α corresponding to the minimum $T(n)$.

n	α	$T(n)$
64	2	14
125	8	21
216	10	31
343	11	44
512	22	57
729	25	73
1000	28	92
1331	43	112
1728	46	134
2197	50	159
2744	69	187
3375	74	216
4096	79	248

Table 2: Function $T(n)$

Being the result of a worst case analysis, the bound stated by theorem 2 is very pessimistic. Conversely, the syndrome dependent bound T_σ given by theorem 1, beside being much more accurate, is easily available. In fact, given any syndrome, this bound is asserted by the diagnosis algorithm itself, since cardinalities $\#\mathcal{D}$ and α are byproducts of the diagnosis.

4. DIAGNOSIS STRATEGY

As it is seen from Figure 1, APEmille can be viewed as consisting of three separate subsystems: the Tmille subsystem, the Cmille subsystem and the Jmille subsystem. The Tmille and the Cmille subsystems are three-dimensional arrays of Tmille instruction processors or of Cmille communication units, respectively. The Jmille subsystem is a three-dimensional arrays of clusters, each cluster consisting of 8 Jmille processors. For every cluster K_x in the Jmille subsystem, there exists a corresponding Tmille instruction processor T_x and a corresponding Cmille communication unit C_x, which share a board B_x with K_x. T_x distributes an instruction sequence to all the Jmille processors in K_x, which it fetches from its private program memory. As the APEmille is a SIMD machine, all Tmille processors work synchronously, each one distributing the same instruction sequence to the Jmille processors in its own cluster. Jmille processors in cluster K_x communicate through Cmille communication unit C_x. Jmille processors in adjacent clusters K_x and K_y communicate through Cmille communication units C_x and C_y.

The three subsystems are diagnosed separately, using the general algorithm presented in the preceding section, up to some details depending on the test invalidation rules, which must take into account the effects of faulty comparators. As it will be clarified in the following, the diagnosis of the Tmille and of the Cmille subsystems proceeds concurrently with the execution of the job assigned to the machine. In fact, pairs of units are tested by comparing their output sequences, and the test outcomes are defined by detecting the occurrence of at least one mismatch over long output sequences. The test strategy requires that all the units be controlled by the same instruction sequence, although different processors may work on different data sets. This requirement is automatically satisfied, since APEmille is a SIMD machine. Conversely, in the diagnosis of the

Jmille subsystem, processor pairs are tested by comparing the output sequences which they write in the bus. This requires that both the instruction sequence and the data sets be the same for all the processors. To this purpose, the system needs to be assigned a special job, called a *diagnosis session*. The self-diagnosis of the Tmille and of the Cmille subsystems is activated periodically, using the syndromes obtained concurrently with the job execution. If this leads to identification of at least one faulty unit, the job execution is aborted and a diagnosis session is immediately started to diagnose the Jmille subsystem as well. The diagnosis session of the Jmille subsystem is also activated following any job successfully completed. As a global results of self-diagnosis of the three subsystems, all those boards which were not recognized as non-faulty (if any) are replaced, to reestablish the correct system behaviour.

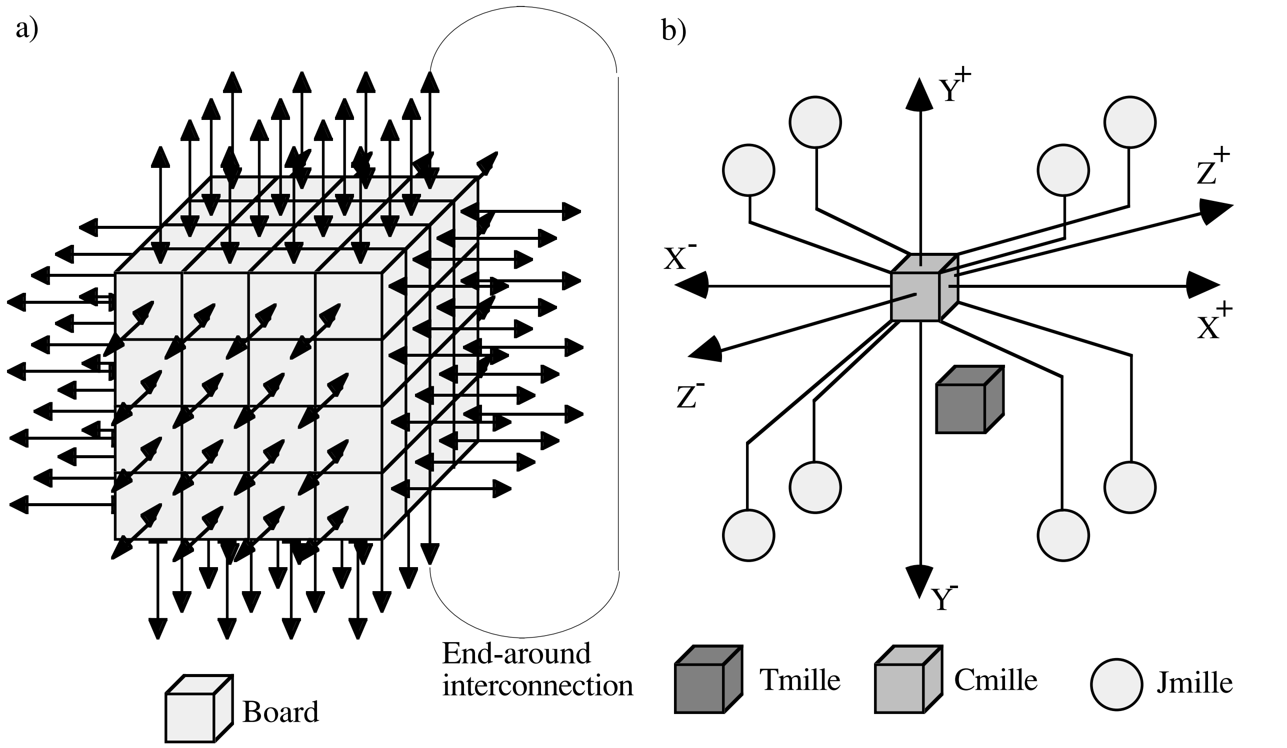

Figure 1: Structure of a) APEmille b) an individual board

The diagnosis of each subsystem will identify a (possibly empty) set of faulty units (in the case of the Tmille and Cmille subsystems) or faulty clusters (in the case of the Jmille subsystem), and a set of units or clusters which are known to be non-faulty. The state of some units or clusters may remain unidentified. However, the number of such units or clusters, which will be called *suspect,* can be expected to be very small, or possibly zero, given the simulation results reported in the forthcoming section. Under the condition stated in the preceding section, the diagnosis is certainly correct: this statement means that units will be declared to be faulty only if they behave incorrectly. However, the incorrect behaviour could be determined by the influence of some other faulty unit (which in turn will be identified as such with high probability), rather than by faults affecting its hardware. For example, the erroneous behaviour of a Cmille communication unit C_X might depend from Tmille processor T_X being faulty, which in fact could distribute wrong instructions. Similarly, a Tmille instruction processor might display an erroneous behaviour, which however, might depend on a fault in the Cmille communication units providing the comparator used to test the Tmille processor itself. Furthermore, the diagnosis of the Jmille subsystem is generally unable to identify those Jmille processors in a given cluster which are actually faulty, although the cluster is correctly diagnosed as affected by at least one fault. However, by combining the results of the diagnoses of the three subsystems, the diagnostic resolution can be generally improved to some extent.

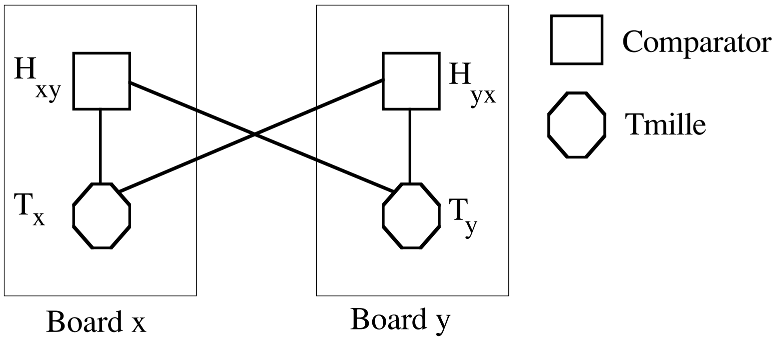

Figure 2: Comparison of Tmille processors

The test of Tmille processors consists in a comparison of checksums. For any pair T_x and T_y of Tmille processors to be compared, T_x and T_y define, independently from each other, a checksum of the instruction code being fetched. The checksums (each a double-rail encoding of the instruction code parity) are collected by a pair of comparators, $H_{x,y}$ and $H_{y,x}$ (Figure 2) which are located in the Cmille units of boards B_x and B_y, respectively. Each comparator is designed to be self-checking [12]; that is, the comparator outputs two variables which are assigned the same binary values (00 or 11) if and only if there is a parity mismatch, or the comparator itself is faulty. The self-checking design covers single, stuck-at-X faults [12]. For every board B_x, the Cmille communication unit contains 6 comparator, each comparing the checksum of T_x with the checksum of a distinct T_y which is a neighbour of T_x in any one of the three possible axes.

Since the test is performed concurrently with the computation assigned to the machine, the comparison covers a very long sequence of instructions, which are supposed to be the same for all Tmille processors. For every comparator, the outputs of the comparisons are accumulated in a two bit register R_T, whose contents will result the same (that is, 00 or 11) if at least one mismatch is observed or if a single, stuck-at-X fault affects the register or its control logic. It is reasonable to assume that any fault in a Tmille processor will produce at least one mismatch, and that the mismatch will be observable at the register outputs. Mismatches may depend on errors in the instruction fetching, or on errors occurring in the instruction transmission, or on the fact that the synchronism of two or more Tmille processors was lost for some reason. Faults affecting the comparator and/or register R_T will also produce observable mismatches with very high probability. All the registers will be eventually read by the external diagnoser, to perform the diagnosis algorithm.

For any pair of Tmille processors being compared, the test outcomes depend on the state (faulty or non-faulty) of T_x, T_y, $H_{x,y}$ and $H_{y,x}$. We denote by $T_x \xrightarrow{\gamma} T_y$ (and, similarly, by $T_x \xleftarrow{\delta} T_y$) the comparison of T_x and T_y, performed by $H_{x,y}$, (the comparison of T_x and T_y, performed by $H_{y,x}$), where γ and δ are the binary outcomes. More concisely, we use notation $T_x \overset{\delta \quad \gamma}{\longleftrightarrow} T_y$ to denote both tests and their outcomes. The assumed test invalidation rule is reported in Table 3. Contrary to the rule in Table 1, this rule takes into account the effect of faults in the comparators $H_{x,y}$ and $H_{y,x}$. It is worth observing that the test outcomes $T_x \overset{1 \quad 0}{\longleftrightarrow} T_y$ which have been assumed when T_x, T_y and $H_{x,y}$ are non faulty and $H_{y,x}$ is faulty (and, similarly, the test outcomes $T_x \overset{0 \quad 1}{\longleftrightarrow} T_y$ which have been assumed when T_x, T_y and $H_{y,x}$ are non faulty and $H_{x,y}$ is faulty, as well as the test outcomes $T_x \overset{1 \quad 1}{\longleftrightarrow} T_y$ which have been assumed when T_x and T_y are non-faulty, while $H_{x,y}$ and $H_{y,x}$ are faulty) depend on the self-checking design of the comparator. In fact, with very high probability, the faulty comparator will erroneously signal a large number of mismatches, by producing test outcomes 00 and 11. Observation of such mismatches could only be impaired by a double stuck-at-X fault affecting the comparator outputs or the register, which is very unlikely to occur. Similarly, the self-checking design of the comparator implies that a pair of Tmille in different states, when compared by one or two faulty comparators, produces at least one mismatch.

In conclusion, the test invalidation rules of Table 3 can be summarized as follows:

- If $T_x \overset{0 \quad 0}{\longleftrightarrow} T_y$, then either T_x and T_y are non-faulty, or both T_x and T_y are faulty. In the latter case, any one of the comparators or both of them could also be faulty.
- If $T_x \overset{0 \quad 1}{\longleftrightarrow} T_y$, or $T_x \overset{1 \quad 0}{\longleftrightarrow} T_y$, or $T_x \overset{1 \quad 1}{\longleftrightarrow} T_y$, then at least one between $T_x \cup H_{x,y}$ and

$T_y \cup H_{y,x}$ must be faulty, where $T_x \cup H_{x,y}$ (and, similarly, $T_y \cup H_{y,x}$) denote the combination of the Tmille processor T_x and of the comparator $H_{x,y}$ (the combination of the Tmille processor T_y and of the comparator $H_{y,x}$). Observe that $T_x \overset{0\quad 1}{\longleftrightarrow} T_y$ does not imply $T_y \cup H_{y,x}$ being faulty, as it is proved by the example of Figure 3, where T_x , T_y and $H_{y,x}$ are non-faulty and $H_{x,y}$ is faulty. Since the comparators used to test the Tmille processors belong to the Cmille subsystem, this is an example of a fault in the Cmille subsystem which causes the Tmille subsystem to display an erroneous behaviour.

T_x	T_y	$H_{X,Y}$	$H_{y,x}$	$T_x \rightarrow T_y$	$T_y \rightarrow T_x$
G	G	G	G	0	0
G	G	G	F	0	1
G	G	F	G	1	0
G	G	F	F	1	1
G	F	G	G	1	1
G	F	G	F	1	1
G	F	F	G	1	1
G	F	F	F	1	1
F	G	G	G	1	1
F	G	G	F	1	1
F	G	F	G	1	1
F	G	F	F	1	1
F	F	G	G	0 or 1	0 or 1
F	F	G	F	0 or 1	0 or 1
F	F	F	G	0 or 1	0 or 1
F	F	F	F	0 or 1	0 or 1

Table 3: Test invalidation rule applying to the Tmille processors

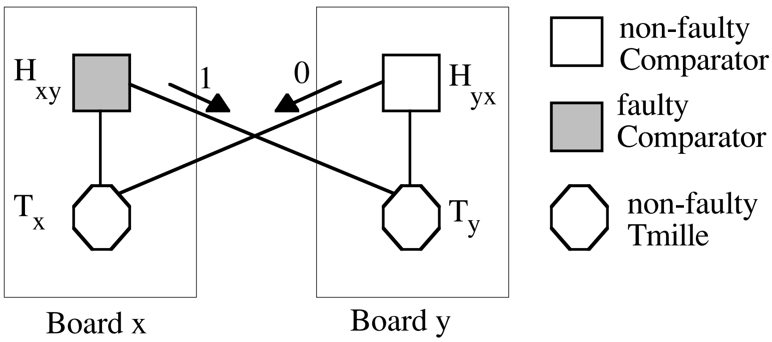

Figure 3: Wrong outcome of a faulty comparator

Given this test invalidation rule, the diagnosis algorithm of the Tmille subsystem becomes as follows:

t1) in the first step *(local diagnosis)*, every Tmille processor T_x tests, and is tested by, its six neighbours, and it is classified accordingly. Precisely, any two T_x and T_y which test each other are classified D if $T_x \overset{1\quad 1}{\longleftrightarrow} T_y$, or $T_x \overset{1\quad 0}{\longleftrightarrow} T_y$, or $T_x \overset{0\quad 1}{\longleftrightarrow} T_y$ and neither T_x nor T_y was already labelled D. All the remaining Tmille processors are classified Z . Adjacent Tmille processors T_x and T_y being classified Z must test each other with $T_x \overset{0\quad 0}{\longleftrightarrow} T_y$

t2) In the second step *(Fault-Free Core identification)*, all pairwise adjacent Tmille processors which are classified Z are combined into aggregates. Tmille processors which do no combine become singleton aggregates. Letting α be the maximum of the aggregate cardinalities, the *Fault-Free Core* $\mathcal{G}$ is defined as the union set of all the aggregates of cardinality α.

t3) In the third step *(augmentation)*, Tmille processors which are tested with outcome 0 by units in the Fault-Free Core are added to the Fault-Free Core itself, while those which are tested with outcome 1 by Tmille processors in the Fault-Free Core are identified as faulty and are added to set $\mathcal{F}$ (initially an empty set). Tmille processors (necessarily not in the Fault-Free Core) which are zero-ancestors of units in set $\mathcal{F}$, are also added to set $\mathcal{F}$.

The test of Cmilles also consists in a comparison of checksums, which proceeds concurrently with the computation assigned to the APEmille. Every 32 data block to be transmitted is encoded in a double error detecting, single error correcting code [13] by the transmitting Cmille C_x. This adds a 8 bit checksum to the data block. In turn, a decoding unit $H_{y,x}$ in the receiving Cmille C_y (Figure 4) corrects all single errors and detects all single or double errors by decoding the received words . Every Cmille unit C_y contains 6 distinct decoding units, each decoding the 40 bit words received from a different Cmille C_x which is a neighbour of C_y in any one of the three possible axes. The decoder outputs are encoded in a two-rail code and are accumulated in a two-bit register R_c, again with the property that at least one mismatch will be observable at the register outputs whenever a single or a double transmission error occurs, or when an encoder or a decoder is faulty, or when a single, stuck-at-X fault affects the register or its control logic. Mismatches may also depend on the fact that the Cmille unit which is supposed to receive (or to send) information was not instructed to do so by its Tmille instruction processor, due to a fault in this processor.

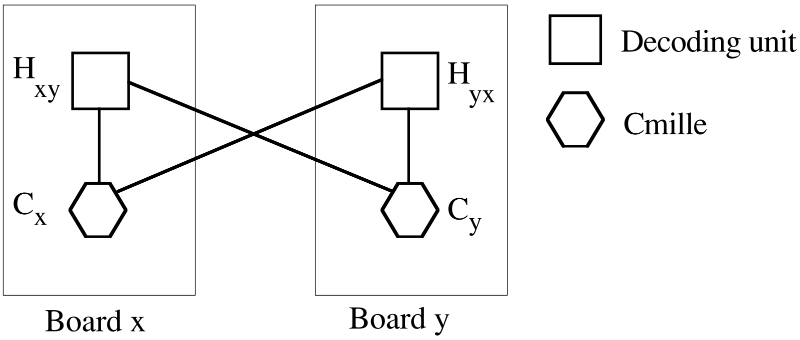

Figure 4: Comparison of Cmille communication units

The test invalidation rule is the same as Table 3, up to a renaming of units undergoing test (Cmille units C_x and C_y instead of Tmille processors T_x and T_y). This rule can be summarized as follows:

- If $C_x \xleftrightarrow{0\quad 0} C_y$, then either C_x and C_y are non-faulty, or both C_x and C_y are faulty. In the latter case, any one of the decoders or both of them could also be faulty.
- If $C_x \xleftrightarrow{0\quad 1} C_y$, or $C_x \xleftrightarrow{1\quad 0} C_y$, or $C_x \xleftrightarrow{1\quad 1} C_y$, then at least one between $C_x \cup H_{x,y}$ and $C_y \cup H_{y,x}$ must be faulty, where $C_x \cup H_{x,y}$ (and, similarly, $C_y \cup H_{y,x}$) denote the combination of a Cmille unit and of a decoder $H_{x,y}$.

Considering the test invalidation rule, the diagnosis algorithm of the Cmille subsystem, which is essentially the same as the one of the Tmille subsystem, is reported below:

c1) in the first step *(local diagnosis)*, every Cmille unit C_x tests, and is tested by, its six neighbours, and it is classified accordingly. Precisely, any two C_x and C_y which test each other are classified D if $C_x \xleftrightarrow{1\quad 1} C_y$, or $C_x \xleftrightarrow{1\quad 0} C_y$, or $C_x \xleftrightarrow{1\quad 1} C_y$, and neither C_x nor C_y was already labelled D. All the remaining Cmille units are classified Z . Adjacent Cmille units C_x and C_y being classified Z must test each other with $C_x \xleftrightarrow{0\quad 0} C_y$

c2) In the second step *(Fault-Free Core identification)*, all pairwise adjacent Cmille units which are classified Z are combined into aggregates. Cmille units which do no combine become singleton aggregates. Letting α be the maximum of the aggregate cardinalities, the *Fault-Free Core* $\mathcal{G}$ is defined as the union set of all the aggregates of cardinality α.

c3) In the third step *(augmentation)*, Cmille units which are tested with outcome 0 by units in the Fault-Free Core are added to the Fault-Free Core itself, while those which are tested with outcome 1 by Cmille units in the Fault-Free Core are identified as faulty and are added to set $\mathcal{F}$ (initially an empty set). Cmille units which are zero-ancestors of units in set $\mathcal{F}$, are also added to set $\mathcal{F}$.

The Jmille processors undergo testing both in a preliminary step of the algorithm (called *intracluster diagnosis)* and in the first step (the *local diagnosis,* which will be renamed *intercluster diagnosis).* Every test is implemented by comparing the output sequences produced by pairs of Jmille processors during a diagnosis session, in which they are assigned the same job with the same data set. As the output sequences are very long, any fault will cause at least one mismatch with very high probability. The comparison outcomes, which are encoded in a two-rail code, are accumulated in a two-bit register R_J, whose contents will eventually detect the occurrence of at least one mismatch.

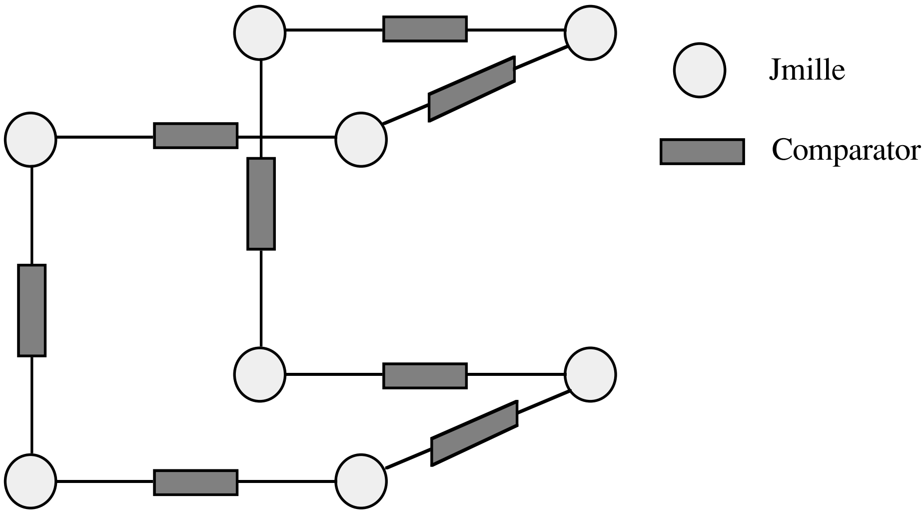

Figure 5: Comparison of Jmille processors in the intracluster diagnosis

For any cluster, the goal of intracluster diagnosis is to ascertain if there exists at least one fault in the cluster. To this purpose, the Jmille processors in each cluster are tested by using the scheme of Figure 5. The Jmille processors in the cluster are identified by indices in the range 0 .. 7 and, for every i in this range, J_i is compared with $J_{(i+1) \bmod 8}$ using a self-checking comparator, denoted $H_{i, (i+1) \bmod 8}$. The invalidation rule is reported in Table 4, where it has been assumed that, due to the self-checking design, any faulty comparator is able to signal at least one mismatch with very high probability, when comparing the outputs of Jmille processors of which at least one is non-faulty. With this rule, if at least two Jmille processors test each other with outcome 1, there must be at least one fault (affecting a processor or a comparator) in the cluster. Conversely, if all test outcomes are 0, either the cluster is completely fault-free or all of its Jmille processors must be faulty. In the latter hypothesis, one or more comparators might also be faulty. If all test outcomes are 0, the cluster is classified Z; otherwise it is classified NZ.

J_X	J_Y	$H_{X,Y}$	$J_X \longleftrightarrow J_Y$
G	G	G	0
G	G	F	1
G	F	G	1
G	F	F	1
F	G	G	1
F	G	F	1
F	F	G	0 or 1
F	F	F	0 or 1

Table 4: Test invalidation rule applying to the Jmille processors in the intracluster diagnosis

In the intercluster diagnosis, for any pair of adjacent clusters K_x and K_y which have been classified Z, a Jmille processor J_x in K_x is compared with a Jmille processor J_y in K_y, using a pair of comparators $H_{x,y}$ and $H_{y,x}$ which are located in clusters K_x and K_y, respectively. J_x and J_y are called the *witnesses* of K_x and K_y, respectively. As the comparisons are executed in a diagnosis session, the Jmille processors to be compared are assigned the same job with the same data set. This scheme of comparison is shown in Figure 6 and the invalidation rule, which is the same as that of Table 3, up to a renaming of units (Jmille processors J_x and J_y instead of Tmille processors T_x and T_y), can be summarized as follows:

- If $J_x \xleftrightarrow{0\quad 0} J_y$, then either J_x and J_y are non-faulty, or both J_x and J_y are faulty. In the latter hypothesis, any one of the comparators or both of them might also be faulty.

- If $J_x \xleftrightarrow{0\quad 1} J_y$, or $J_x \xleftrightarrow{1\quad 0} J_y$, or $J_x \xleftrightarrow{1\quad 1} J_y$, then at least one between $J_x \cup C_{x,y}$ and $J_y \cup C_{y,x}$ must be faulty. As it was shown in the example of Figure 3, the occurrence of test outcomes $J_x \xleftrightarrow{0\quad 1} J_y$ does not imply that J_y is faulty.

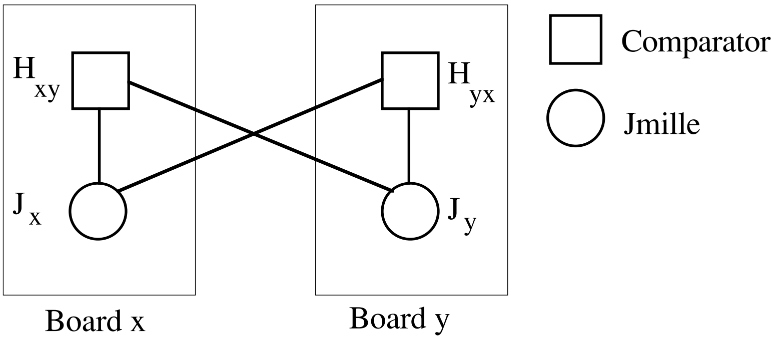

Figure 6: Comparison of Jmille processors in the intercluster diagnosis

As a result of the intercluster tests, the classification of those clusters that were classified Z is modified as follows:

- given cluster K_x and K_y, both classified Z, if witnesses J_x and J_y test each other with $J_x \xleftrightarrow{0\quad 1} J_y$, or $J_x \xleftrightarrow{1\quad 0} J_y$, or $J_x \xleftrightarrow{1\quad 1} J_y$, then classification of both clusters K_x and K_y becomes D;

- all those clusters K_x and K_y, whose witnesses J_x and J_y test each other with $J_x \xleftrightarrow{0\quad 0} J_y$, retain their classification of Z and are aggregated. It is immediate that all the Jmille processors in the same aggregate must be in the same state (faulty or non-faulty).

In conclusion, the diagnosis algorithm of the Jmille subsystem is as follows:

j0) in the *intracluster diagnosis*, for each cluster K_x, Jmille processor J_i tests, and is tested by, Jmille processor $J_{i+1 \bmod 8}$ for every i, i= 0 .. 7. If $J_i \xleftrightarrow{1} J_{i+1 \bmod 8}$ for at least one i, cluster K_x is classified NZ, otherwise it is classified Z. Observe that intracluster diagnosis can proceed independently for all clusters.

j1) in the *intercluster diagnosis,* for any two adjacent clusters K_x and K_y which are classified Z, the respective witnesses J_x and J_y test each other. If test outcomes are $J_x \xleftrightarrow{0\quad 1} J_y$, or $J_x \xleftrightarrow{1\quad 0} J_y$, or $J_x \xleftrightarrow{1\quad 1} J_y$, then classification of both clusters K_x and K_y becomes D. Any other cluster retains its classification.

j2) In the *Fault-Free Core identification*, all pairwise adjacent clusters which are classified Z are combined into aggregates. Clusters which do no combine become singleton aggregates. Letting α be the maximum of the aggregate cardinalities, the *Fault-Free Core* $\mathcal{G}$ is defined as the union set of all the aggregates of cardinality α.

j3) In the last step *(augmentation)*, for every pair of clusters K_x and K_y, with K_x in the Fault-Free Core and K_y not belonging to this set, K_x tests K_y. The test is performed by the respective witnesses J_x and J_y. If $J_x \xrightarrow{0} J_y$, then K_y is identified as non-faulty and added to the Fault-Free Core. If $J_x \xrightarrow{1} J_y$, then K_y is identified as faulty and added to set $\mathcal{F}$ (initially an empty set). Furthermore, for every pair of clusters K_x and K_y, with K_x not belonging to the Fault-Free Core and K_y identified as faulty, K_x tests K_y. Again the test is performed by the respective witnesses J_x and J_y.If $J_x \xrightarrow{0} J_y$, then K_x is identified as faulty and added to set $\mathcal{F}$.

The diagnosis algorithm of the Jmille subsystem yields a set of clusters (those in the Fault-Free Core) which are completely fault-free and a set of clusters (those classified NZ and those in set $\mathcal{F}$)

which are understood to contain at least one faulty Jmille processor or (inclusively or exclusively) at least one faulty comparator. The clusters which cannot be identified as either fault-free or faulty are declared to be suspect. Combining this information with the diagnosis of the Tmille and of the Cmille subsystems, which identify sets of non-faulty, faulty and suspect Tmille processors (or, respectively, of Cmille communication units), most system boards (or, possibly, all of them) can be identified as either fault-free, faulty, or suspect. Board B_x is fault-free if all its components (T_x, J_x and K_x) are non-faulty; it is faulty if at least one component among T_x, J_x and K_x is faulty and it is suspect if at least one component among T_x, J_x and K_x is suspect and any other component is non faulty. This result fulfills the requirements of self-diagnosis, since the board is the smallest replaceable unit in the system: replacing all faulty or suspect boards will be sufficient to remove all faults from the system.

Although the self-diagnosis may be considered complete after identifying the faulty boards, it is generally able to provide additional and more detailed information. In fact, extending the diagnosis session of the Jmille subsystem with a few additional tests may enable identification of some Jmille processors which are faulty (or non-faulty) within those clusters which were identified as containing at least one fault. In fact, if K_x and K_y are adjacent clusters with K_x fault-free and K_y containing at least one fault, assume that J_x tests J_y with $J_x \xrightarrow{0} J_y$ (where J_x and J_y are the witnesses of K_x and K_y, respectively). Then J_y is identified as non-faulty; similarly, if J_x tests J_y with $J_x \xrightarrow{1} J_y$, J_y is identified as faulty.

Additional information may be gathered by combining the diagnoses of the Tmille, the Cmille and the Jmille subsystem, which were performed independently. If T_x, C_x and K_x were diagnosed faulty, but K_x was classified Z in the intracluster diagnosis, K_x could actually be fault-free and its erroneous behaviour could depend on T_x, which distributed the same sequence of wrong instructions to all the Jmille processors in K_x. Similarly, considered that any faulty Tmille processor T_x would necessarily cause incorrect behaviours of both C_x and K_x, identification of T_x as faulty is invalidated if at least one between C_x and K_x is diagnosed non-faulty. In fact, T_x is likely to be non-faulty and its identification as faulty is likely to depend on a fault affecting comparator $H_{x,y}$, which resides in C_x.

5. DIAGNOSIS EVALUATION

Although it was proved to be correct for any syndrome deriving from at most $T(n)$ faults, the diagnosis provided by the algorithms presented in this paper may be incomplete [10]. In fact, one or more boards might be completely surrounded by faulty boards and, in this hypothesis, the reliable comparisons propagating from the Fault-Free Core could not identify their state.

The probability of complete or almost complete diagnosis has been evaluated by means of simulation. This section reports some preliminary results, which refer to the largest possible configuration of the APEmille (i.e. n=512 boards and 4096 processing elements). Every row of Table 5 corresponds to a simulation run, in which a large number of syndromes were generated. Every syndrome resulted from a certain number of faults injected in each subsystem, this number being the same for all syndromes in each run, while the numbers were different in different runs. The faults were distributed uniformly among the units of each subsystem, at most one fault for every unit (Tmille processor, Cmille unit or Jmille processor). The number of faults ranged from 20 to 65 for Tmille and Cmille subsystems, and from 50 to 140 for Jmille subsystem. Given any fault set, the syndrome was generated assuming the invalidation rules presented in section 4, and equal probability of either outcome when faulty units were compared.

The meaning of column heading in Table 5 is as follows:

- *Tmille Faults*, *Cmille Faults* and *Jmille Faults* are the number of faults injected in the respective subsystems, for each simulation run;
- *Board Faults* is the average number of boards which were affected by at least one of those faults which were injected in the three subsystems;
- *Correct Diagnoses* is the average of percentages of diagnoses which resulted correct in each simulation run;
- *Complete Diagnoses* is the average of percentages of diagnoses which resulted complete in each simulation run;
- *Suspect Boards* is the average number of boards that could not be identified as either faulty or non-faulty, considering only those syndromes in the run leading to incomplete diagnoses.

As it is seen from Table 5, the diagnosis was always found to be correct in all simulation runs where the number of injected faults was below $n/2$; that is, a threshold much above $T(n)$. Such result is not surprising since the bound $T(n)$ is very pessimistic and it was obtained using rough approximations. Whenever the number of faulty boards was not above $T(n)$, the diagnosis was also found to be complete, and incomplete diagnosis were obtained only if this number was further increased. However, the average number of suspect boards remained very small and almost independent of the number of faults.

Tmille Faults	Cmille Faults	Jmille Faults	Board faults (average)	Correct diagnoses (percentage)	Complete diagnoses (percentage)	Suspect boards (number)
20	20	50	83.5	100	100	0
25	25	60	100.9	100	98.4	1.00
30	30	70	117.1	100	89.1	1.16
35	35	80	132.3	100	89.7	1.14
40	40	90	148.7	100	85.9	1.12
45	45	100	163.5	100	64.8	1.30
50	50	110	177.6	100	46.2	1.44
55	55	120	191.5	100	36.7	1.82
60	60	130	204.9	100	24.2	1.76
65	65	140	217.2	100	12.3	2.24

Table 5: Diagnosis correctness and completeness: simulation results

REFERENCES

[1] Tripiccione, R., "Ape100 and beyond", *Int. Journ on Modern Phisics*, sec.C vol.4, 1993, pp.13-23.

[2] Bartoloni, A., Battista, C., Cabasino, S., Cosimi, M., D'Agostini, U., Marzano, F., Panizzi, E., Paolucci, P.S., Rapuano, F., Rinaldi, W., Sarno, R., Todesco, G.M., Torelli, M., Vicini, P., Cabibbo, N., Fucci, A. and Tripiccione, R., "APEmille: a Parallel Processor in the Teraflops Range", INFN preprint, March 1995.

[3] Hwang, K. and Briggs, F.A., Computer Architecture and Parallel Processing, New York, McGraw-Hill, 1984.

[4] Wilson, K.G., Physical Review vol. D10, 1974, pp. 2445.

[5] Shigemitsu, J., "Lattice Gauge Theory: A Status Report", *Proceedings of the XXVII Int'l Conference on High Energy Physics* (edited by P. J. Bussey and I. G. Knowles). Institute of Physics Publishing, 1995, pp. 135- 156.

[6] Preparata, F.P, Metze, G. and Chien, R.T., "On the Connection Assignment Problem of Diagnosable Systems". *IEEE Trans. Comput.,* vol. EC-16, Dec. 1967, pp. 848-854.

[7] Barborak, M., Malek, M. and Dahbura, A.T., "The Consensus Problem in Fault Tolerant Computing". *ACM Surveys* , vol.25 n.2, June 1993, pp. 173 - 220.

[8] LaForge, L. E., Huang, K. and Agarwal, V., "Almost Sure Diagnosis of Almost Every Good Element". *IEEE Trans. Comput*., vol. 43, Mar. 1994, pp. 295-305.

[9] Huang, K., Agarwal, V., LaForge L. and Thulasiraman K., "A Diagnosis Algorithm for Constant Degree Structures and Its Application to VLSI Circuit Testing". *IEEE Trans. Parallel Distrib. Syst*., vol. 6, Apr. 1995, pp. 363-372.

[10] Chessa, S. and Maestrini, P., "Correct and Almost Complete Diagnosis of Processor Arrays" Istituto di Elaborazione dell'Informazione del CNR, Technical Report B4.27, Pisa - Italy, June 1995

[11] Maestrini, P. and Santi, P., "Self Diagnosis of Processor Arrays Using a Comparison Model". *Proc. 14.th Symposium on Reliable Distributed Systems,* IEEE Comput. Soc. Publ., September 1995, pp. 218-228.

[12] Siewiorek, D. P. and Swarz, R. S., The Theory and Practice of Reliable System Design, Bedford, MS, Digital Press, 1982.

[13] Peterson, W. W. and Weldon, E. J., Error Correcting Codes, Boston, MIT Press, 1972.