論 文

초대규모 집적 또는 웨이퍼규모 집적을 이용한 셀룰러 병렬 처리기의 재구현

正會員 韓 在 一*

Reconfiguration Problems in VLSI and WSI Cellular Arrays

Jae Il Han* *Regular Member*

요 약

전형적인 컴퓨터보다 훨씬 강력한 계산 능력을 얻기 위해 병렬 컴퓨터 구조에 대한 많은 연구가 진행되어 왔다. 이러한 컴퓨터들은 통상 상호 연결 네트워크(Interconnection Network)로 연결된 많은 수의 처리기들로 구성된다. 그중 중요한 한 부류가 초대규모 집적(Very Large Scale Integration) 또는 웨이퍼 규모 집적(Wafer Scale Integration)을 이용한 셀룰러 병렬 처리기로 하나의 칩이나 웨이퍼에 단지 이웃으로만 연결된 많은 수의 단순 구조를 가지는 처리기로 구성된다. 이런 셀룰러 병렬 처리기들에 반드시 수반되는 문제가 재구현(Reconfiguration)으로 세가지 유형을 정의할 수 있는데 본 논문에서는 이 세가지 재구현 문제, 즉 결함 허용 재구현(Fault-Tolerant Reconfiguration), 기능적 재구현(Functional Reconfiguration), 그리고 통합 재구현(Integrated Reconfiguration)에 대하여 논하였다. 본 논문은 결함 진단 및 검출(Fault Detection and Fault Location) 제어 방법, 구성(Configuration) 제어방법, 재구현의 수행 단계 등 결함 허용 재구현과 기능적 재구현시 필요한 여러 고려 사항을 분석 정리하고, 최근 제기된 결함 허용 재구현과 기능적 재구현의 일체화 문제 즉 통합 재구현 문제의 이해에 핵심적인 결함 허용 재구현과 기능적 재구현 사이의 관계를 밝혔으며, 통합 재구현에 적합한 결함 진단 및 검출 제어 방법과 구성 제어 방법에 대하여 논하였다.

Abstract

A significant amount of research has focused on the development of highly parallel architectures to obtain far more computational power than conventional computer systems. These architectures usually comprise of a large number of processors communicating through an interconnection network. The VLSI(Very Large Scale Integration) and WSI (Wafer Scale Integration) cellular arrays form one important class of those parallel architectures, and consist of a large number of simple processing cells, all on a single chip or wafer, each interconnected only to its neighbors. This paper studies three fundamental issues in these arrays : fault-tolerant reconfiguration, functional

*韓國電子通信研究所
Electronics And Telecommunications Research Institute
論文番號 : 93-157

reconfiguration, and their integration. The paper examines conventional techniques, and gives an in-depth discussion about fault-tolerant reconfiguration and functional reconfiguration, presenting testing control strategy, configuration control strategy, steps required for each reconfiguration, and other relevant topics. The issue of integrating fault-tolerant reconfiguration and functional reconfiguration has been addressed only recently. To tackle that problem, the paper identifies the relation between fault-tolerant reconfiguration and functional reconfiguration, and discusses appropriate testing and configuration control strategy for integrated reconfiguration on VLSI and WSI cellular arrays.

I. Introduction

The rapid progress in integrated circuit technology, and the increasing need for the design of high-performance computer, have led to the design of highly parallel architectures. Those architectures usually comprise of a large number of processors communicating through an interconnection network. It is anticipated that more advanced integration circuit technologies will introduce lower-cost, higher-density, and faster devices, encouraging further development of such architectures [1,2]. One important class of devices includes the VLSI (Very Large Scale Integration) and WSI (Wafer Scale Integration) cellular arrays consisting of a large number of simple processing cells, all on a single chip or wafer, each interconnected only to its neighbors. It will be very difficult, however, to make large structures of this kind without many defects. For brevity and ease of presentation, the abbreviation LSCA (Large Scale Cellular Array) shall be used to denote VLSI and WSI cellular arrays.

There are a variety of LSCAs proposed, such as systolic arrays [3], the wave front array [4], configurable highly parallel computer [5], and the defective processor array [6]. Two fundamental issues in these arrays are *fault-tolerant reconfiguration* and *functional reconfiguration*. Fault-tolerant (FT) reconfiguration is concerned with the process of realizing a desired target topology (such as a mesh) from the operational part of a defective physical array; functional reconfiguration deals with *polymorphism*[5], that is, how to reconfigure the topology of a parallel system to implement a different function or to run a different application. The issue of integrating FT and functional reconfigurations, here called *integrated reconfiguration*, has been addressed only recently [7], and to the best of the author's knowledge, no significant results have appeared yet. Successful integrated reconfiguration on defective LSCAs can give benefits to be gained from both FT reconfiguration and functional reconfiguration.

Although many reconfigurable LSCAs have been proposed in the past, most of them are concerned with FT reconfiguration [6,8-25], and very few proposals deal with functional reconfiguration [5,26,27]. Moreover, in combination of FT reconfigurability with functional reconfigurability in highly defective LSCAs, the conventional approaches break down, and a different approach is needed. The study of conventional reconfiguration techniques shows that the design of a reconfigurable LSCA requires co-development of a reconfiguration strategy, a proper architecture, and a configuration algorithm.

This paper anatomizes fault-tolerant and functional reconfigurations, and deals with the feasibilty of their integration in an LSCA. Conventional fault-tolerant and functional reconfiguration techniques are discussed in section 2 and 3. In section 4, the relation between FT and functional reconfigurations is identified. In section 5, the paper talks about basic requirements for the integration of FT and functional reconfigurations,

shows that their difference vanishes in integrated reconfiguration, and discusses an appropriate control strategy for integrated reconfiguration. Many terms from graph theory are used extensively in this paper. For discussion of the terms, and their definition, the reader is referred to the standard literature, such as [28,29,30].

Ⅱ. Fault-tolerant reconfiguration

Two types of faults exist within LSCA chips: *production defects* that occur during manufacturing, and *operational faults* that occur during the lifetime of the working chip. Therefore, fault-tolerance in an LSCA stems form two fundamental key factors: *yield enhancement* and *reliability* [7,18, 31-34]. It is desirable to increase yield when manufacturing so as to offset production defects and to increase reliability during the life of the array by offsetting operational faults.

Fault-tolerant techniques can be broadly classified into three approaches [18]: defect avoidance, module replacement and dynamic fault recovery. The defect avoidance approach tries to reduce the sensitivity of a device to processing defects by adapting the circuit layout in such a way that it is susceptible to failure in the presence of known defect mechanisms. In the module replacement approach, the device architecture is divided into a number of module types, each of which is replicated two, three, or more times. Non defective modules are harvested and configured into a working device after fabrication. As with the module replacement approach, the aim of the dynamic fault recovery approach is to enable correct device functionality in spite of the presence of faults within the device. In the dynamic fault recovery approach, all signal I/O is checked and corrected (if necessary) at the module and/or device interfaces instead of attempting to identify and isolate the faulty elements within the device. The most common approach by far, and the one that is considered herein, is the module replacement scheme.

Let us introduce our nomenclature. A physical array, interchangeably represented as a graph, defines a *physical graph*, which represents the topological structure and status of the physical array (Fig. 1). An edge of the physical graph is defined as an *external edge* if it represents a wire communicating with the external world and is defined as an *internal edge* otherwise. In the physical graph, each external edge is incident to two ends. One corresponds to a physically real cell and the other represents a *dummy cell*. The additional vertices of a physical graph representing dummy cells are necessary not only conform to the definition of a graph, but also to identify external communi-

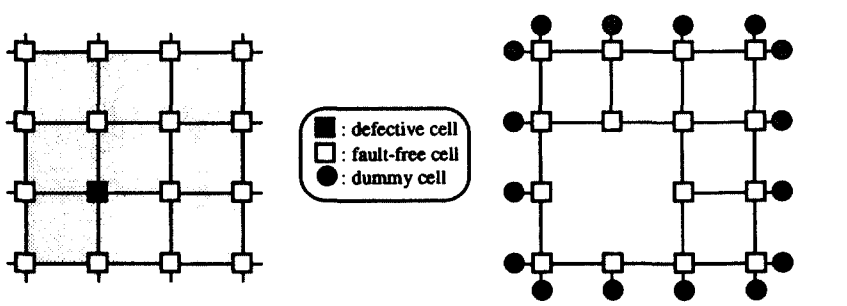

Fig 1. A Physical Graph Representing the Physical Array

cation paths during reconfiguration. These vertices, however, will generally not be shown in our figures.

2.1 Four measures of FT reconfiguration techniques

Production defects and operational faults are usually handled by introducing additional hardware (architectural requirement) and restructuring the array by means of that extra hardware (algorithmic requirement) [33,34]. The hardware added can be redundant cells or processing elements, external switching elements, communication links, an internal switching element in the cell, or any combination of these. Using this extra hardware, a physical array which has defective elements is reconfigured into a desired target array of fixed interconnection pattern and size, which we call a *virtual array*, by isolating them and connecting good elements (Fig.2). Here, a virtual array is formally defined as a graph whose vertices represent programmable or dummy cells and whose edges represent bidirectional interconnections between adjacent cells. External and internal edges of the virtual array are defined identically to those of the physical graph. Virtual arrays required for the majority of computation-intensive applications tend to be regular graphs such as a linear array, a mesh, a hexagonal array, or a binary tree [1].

And FT reconfiguration technique is implemented by avoiding defective elements and connecting functional elements, thus it usually increase the *communication delay* between two cells participating in computation. Whereas communication delay is a critical factor in the performance of a synchronous cellular array due to clock skew [14,15,19,24,34,35], the delay is less crucial in an asynchronous cellular array because of the data-driven character of the computation. It is clear, however, that long communication delays have a negative effect on the throughput [1].

Utilization and *survivability* are other important factors that affect the design of FT reconfi-

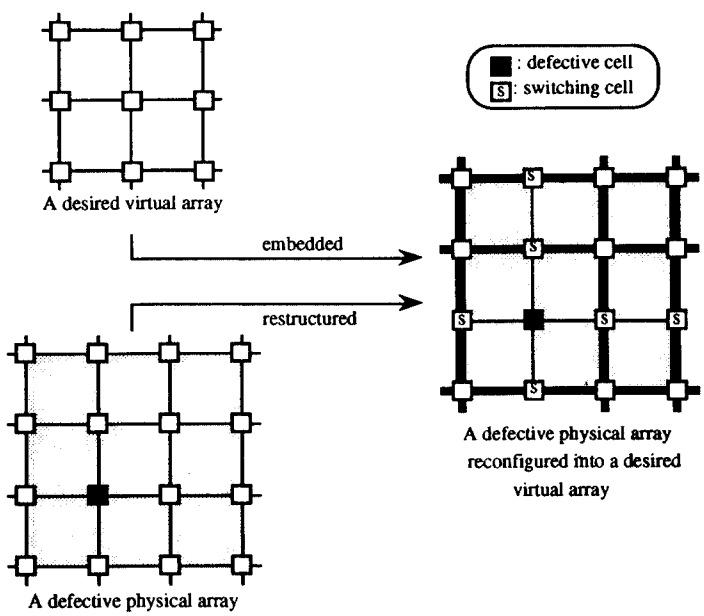

Fig 2. Fault-Tolerant Reconfiguration

gurable arrays [7,22,33]. The term *utilization* refers to the ratio of fault-free cells used to construct the virtual array to the total fault-free cells in the defective physical array. The probability of survival $S(x)$, called *survivability*, is the conditional probability of success in reconfiguring a target virtual array, given that it has x working cells. Survivability and utilization have a close relationship. A reconfiguration scheme that achieves good utilization by wasting none, or very few cells, will provide high survivability.

Finally, a configuration algorithm in FT reconfiguration should be simple and efficient so that its time complexity is polynomially bound as a function of the number of cells in an LSCA [7]. The above factors-time complexity of a configuration algorithm, utilization/survivability, area overhead, and communication delay-should be considered in the design and evaluation of an FT reconfiguration technique [7,33,34].

The FT reconfiguration requires three phases : a *testing phase* to detect and locate faults, an *FT configuration phase* to generate a configuration mapping, and a *loading phase* to actually restructure a physical LSCA into a target virtual array either by hard-wiring or soft-wiring. In this paper, *configuration* is defined as the process of generating a configuration mapping from a source graph to a target graph. *Embedding* is defined as the process of performing configuration and actual loading. The configuration and embedding of a target virtual array in FT reconfiguration are called by us, respectively, *fault-tolerant* (FT) *configuration* and *fault-tolerant* (FT) *embedding*. The time spent in configuration and loading phases is called *array-embedding overhead*.

2.2 Fault-tolerant reconfiguration techniques

Testing and FT configuration may be controlled either by an external machine or the array itslf, and as such, there are four possible reconfiguration strategies :

1. Controller-driven Testing and Controller-driven Configuration (CTCC),
2. Self-Testing and Self-Configuration (STSC),
3. Self-Testing and Controller-driven Configuration (STCC),
4. Controller-driven Testing and Self-Configuration (CTSC).

The conventional FT reconfiguration techniques proposed for LSCAs are classified into two groups [17,18,22,32] : *static (or physical) FT reconfiguration* and *dynamic (or logical) FT reconfiguration*. Static FT reconfiguration deals with production defects and always adopts the CTCC strategy (e.g., [14,19,36,37,38]). The technologies proposed for static FT reconfiguration (following manufacturing and testing) include electrical and laser programmed fuses, laser welding, E-beam programming of electrical switches, and discretionary wiring [31,32,39,40]. In static reconfiguration, the connections in the physical array are hard-wired to form a target virtual array of specific interconnection pattern and size, and they cannot be altered once the physical array is reconfigured. In contrast, dynamic FT reconfiguration can deal with both production defects and operational faults (e.g., [6,8,12,23,40,41]). It may be performed and unlimited number of times at system power-up and at run time, as may be required during the life of the physical array. This technique requires a form of electrically programmable switches which are built into the physical array.

A comparison of these two reconfiguration techniques shows that the static FT reconfiguration techniques tend to have low area overhead (hardware) and good performance but are expensive in terms of the initial equipment cost and/or the fabrication-customization throughput rate. Dynamic FT reconfiguration techniques offer multiple reconfiguration capability from faults, but the area overhead (extra circuitry) tends to be higher, and testing and configuration control software must be provided either on or off silicon, so as to allow multiple reconfigurations [18,32,33]. This paper will focus on dynamic FT reconfiguration since the integration of FT and

functional reconfigurations requires recovery from operational faults.

2.3 Testing and configuration in dynamic ault-tolerant reconfiguration

Operational faults may be detected either on system power-up or in the middle of computation at run time. In power-up-time dynamic FT reconfiguration, one test of a physical array is sufficient to detect both operational faults and production defects, and either of the controller-driven or self-testing strategies can be employed. Detection of operational faults at run time raises the problem of performance degradation regardless of the testing strategy. Since the time of fault occurrence cannot be predicted, each cell should be tested while it performs computation. The testing in rum-time dynamic FT reconfiguration involves much higher overhead than the testing in power-up-time dynamic FT reconfiguration.

In controller-driven testing techniques, no additional and software support needs to be provided within each cell, as it is assumed that there is an external unit which is responsible for initiating the test. Performance degradation is inevitable in controller-driven testing schemes because the entire computation performed in the physical array is interrupted whenever the external unit initiates testing. Self-testing techniques require additional hardware and software support within each cell. However, self-testing schemes can examine the array with low performance degradation, because self-testing schemes are implemented by making the cells themselves responsible for performing periodic testing concurrently with their normal operation. In this way, testing can be interleaved with computation [33,42,43]. Moreover, the self-testing strategy has the potential for better fault coverage because of the proximity of the testing unit and the unit under test [33]. Hence, for run-time dynamic FT reconfiguration, the self-testing strategy is more appealing than the controller-driven one, as it offers better fault coverage and better performance. The price is the requirement of extra hardware and software support at each cell. Among self-testing techniques, the *Concurrent Built-In Self-Testing* (CBIST) technique, which tests combinational logic circuits concurrently with normal operation (e.g., [44]), is most attractive. It has no periodic testing inside of a cell, and the reduced *detection time* between fault occurrence and fault detection at run-time may result in even better performance of a cellular array.

The only task of the FT configuration is to configure a physical graph into a target virtual array. The task has nothing to do with ongoing computation in the array, and imposes no additional overhead regardless of when reconfiguration is performed. Thus, the array-embedding overhead remains stable at all times for any given configuration strategy. Either of the controller-driven and self-configuration strategies can be employed in power-up-time and run-time dynamic FT reconfigurations. The selection between the two strategies entirely depends on a particular implementation of dynamic FT reconfiguration. The controller-driven configuration strategy requires an external host that is responsible for FT configuration. In the self-configuration strategy, additional hardware and software support is required at each cell to perform FT configuration locally. The advantage of the self-configuration strategy is that FT configuration can be made transparent to most of system since the FT configuration task is made local to each node [33].

2.4 Control strategy in dynamic fault-tolerant reconfiguration

The conventional dynamic FT reconfiguration techniques are classified into two groups : *centralized* and *distributed* approach [33]. A centralized scheme adopts the CTCC strategy, thus, testing and FT configuration of the array are controlled by the external unit, called a *controller* (e.g., [10, 16,22,24,45]). In a distributed dynamic FT reconfiguration scheme, the STSC strategy is adopted and the cells themselves are responsible for

performing testing and FT configuration in the asynchronous mode (e.g., [8,12,13,41]).

The STCC and CTSC strategies have not been adopted yet for dynamic FT reconfiguration. However, the CTSC strategy is clearly of no use. It suffers from performance degradation caused by controller-driven testing, and requires additional hardware and software support at each cell for self-configuration. In contrast, the STCC strategy can be viewed as the compromise between the CTCC and the STSC strategy. A successful implementation of the STCC strategy will not involve testing overhead as does the CTCC strategy and will have less area overhead than the STSC strategy. Hence, the STCC strategy is a viable technique for dynamic FT reconfiguration.

2.5 Temporal classification of dynamic fault-tolerant reconfiguration

The FT reconfiguration can be performed at three distinct instances : static FT reconfiguration right after fabrication, dynamic FT reconfiguration on system power-up, or dynamic FT reconfiguration at run time. When FT reconfiguration is performed, and no user computation is involved, the only task is to embed a target virtual array into a defective physical array. Hence, static or dynamic FT reconfigurations on system powe-up are straightforward and can be accomplished in three steps : a *testing phase*, an *FT configuration phase*, and a *loading phase*.

In contrast, there is an ongoing computation in the physical array at run time. Dynamic FT reconfiguration performed at run time has to recover the topology of a target virtual array as well as the functionality assigned to the virtual array. If physical cells are homogeneous, that is, the same function is built into every physical cell, FT reconfiguration can be completed in the above three steps. In order to compensate for the cost of extra-hardware, however, the dynamically FT reconfigurable arrays often employ programmable cells and are utilized for a broader problem domain. If physical cells are programmed to execute different functions, it is necessary to reassign their functions when faults occur at run time [1].

Thus, dynamic FT reconfiguration at run time is more complex than FT reconfiguration at other instances, and requires the above three steps *plus* one more step ; it is termed a *reinitialization phase*. The reinitialization phase has been considered partially in [1,13]. It should be handled carefully so that it be transparent to the external world [22]. The reinitialization phase always imposes an overhead time to recover the computing structure of an algorithm at the time of fault occurrence [9].

Ⅲ. Functional Reconfiguration

The computing structure of each algorithm can be represented as a directed graph, called a *computation graph* [6], whose arce represent the communication structure of an algorithm and whose vertices are associated with the appropriate arithmetic and logic operations for the algorithm. An arc of a computation graph is defined an *external arc* if it is used for communication with the external world. Otherwise, an arc is called an *internal arc*·Either the tail or the head of an external arc is a dummy vertex. In the paper, computation graphs and virtual arrays are collectively called *logical computing structures*. Since computation graphs are derived from different application algorithms, each computation graph exhibits its own interconnection patterns (i.e., the communication structure) and size (the number of vertices). When a physical array is not big enough to embed a computation graph, a partitioning technique has to be employed by which a large computation graph can be transformed into a smaller computation graph or partitioned into small computation graphs that can fit into the size of a physical array [1]. In the paper, our interest in functional reconfiguration does not lie in such partitioning techniques regarding size but in its capability of identifying whether computation graphs of differ-

ent interconnection patterns can be embedded into a physical array. Functional reconfiguration is generally considered to fail when a given computation graph cannot be embedded into a physical array. In most of our discussion, it is implicitly assumed that the size of a computation graph is small enough to be embedded into a physical array.

The design and manufacturing cost of LSCAs led few researchers to consider the use of LSCAs for arbitrary algorithms (e.g., [5,26]). Even in algorithmically specialized processors such as systolic arrays, programmable processor modules are more favored than dedicated modules (e.g., [4,46, 47,48]). For the economic viability of LSCAs, it is desirable to enable them to embed and thus execute arbitrary computation graphs. Such capability is of importace independent of its economic value, since it may be possible to attain the efficiency of special-purpose devices with standardized devices [22].

The paper defines two kinds of functional reconfigurations, termed *dynamic functional reconfiguration* and *static functional reconfiguration*. A physical cellular array is said to be capable of dynamic functional reconfiguration when successive configurations of the physical array into different computation graphs are possible at run time. If an interconnection topology is replaceable only at power-up time and just arithmetic and logic operations can be reassigned to cells at run time, or if very few computation graphs could be embedded at run-time, the array is said to have static functional reconfigurability. Clearly, dynamic functional reconfiguration is a necessary feature for integrated reconfiguration. The paper shall focus on dynamic functional reconfiguration, and that is what the paper means when the term *functional reconfiguration* is used. The paper shall not discuss the question of how to derive comptation graphs, such as signal-flow graphs and data-flow graphs, from application algorithms. Many methods for such derivation are mentioned in the literature (e.g., [1,49,50]). The paper concentrates on the discussion of how to embed computation graphs into an LSCA, assuming the existence of appropriate computation graphs for application algorithms.

3.1 Functional embedding and functional configuration

The embedding of a computation graph into a physical array, which we call *functional embedding*, basically requires two steps : *functional configuration* to compute a configuration mapping from a computation graph onto a physical graph, and *functional loading* to load a computation graph into a physical array according to the configuration mapping [26,51] (Fig.3). Since computation graphs derived from different application algorithms will have different communication patterns and different assignment of arithmetic and logic operations to vertices, the main question in functional reconfiguration is how to accomplish the above two task with as little overhead as possible. One solution to this problem is to design a polymorphic LSCA [5] that allows simple and efficient functional embedding for a variety of computation graphs. Such a polymorphic LSCA can then be used either as an accelerator attached to a general-purpose host machine or as a stand-alone machine equipped with a controller, and will be utilized to execute many computation-intensive algorithms by frequently changing and executing them.

A configuration mapping from a computation graph onto a physical graph, which we call a *physical mapping*, can be obtained directly or indirectly. Note that in conventional functional reconfiguration approaches (e.g., [5,26]), underlying physical arrays are considered fault-free. Direct computation of a physical mapping is performed by mapping a computation graph directly onto a physical graph ; indirect computation of the physical mapping is performed by indirectly mapping a computation graph onto a physical

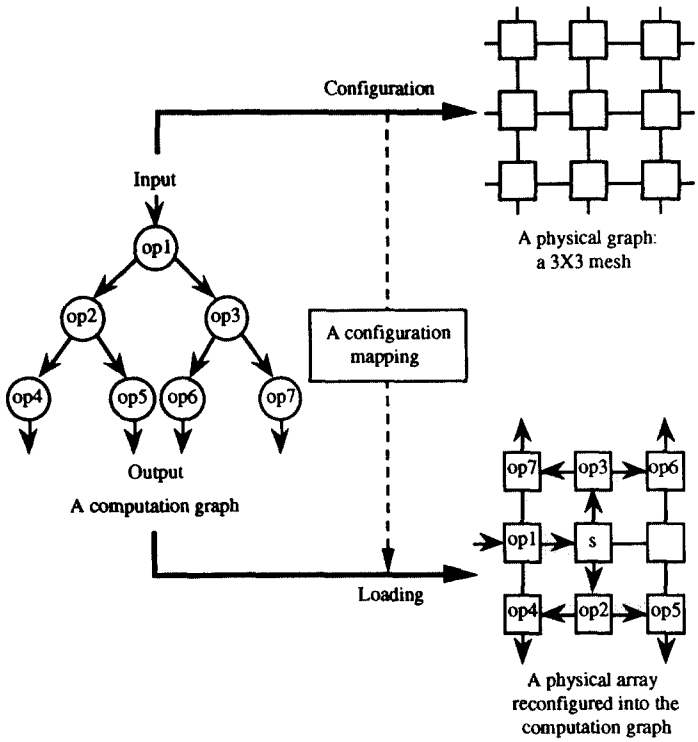

Fig 3. Functional Reconfiguration

graph by measns of an intermediate logical array of predefined interconnection pattern and size. In section 2, the paper introduced the term *virtual array* for such an intermediate logical array.

In indirect functional configuration, a configuration mapping from a virtual array to physical graph, which we call an *array mapping*, is generated and stored by the controller. Later, when a computation graph is mapped onto a virtual array, a configuration mapping from the computation graph to the virtual array, which we call a *logical mapping*, is obtained, and a physical mapping is computed by combining a logical mapping and an array mapping. The advantage of indirect functional configuration is that physical mappings for many computation graphs can be obtained easily and efficiently by using virtual arrays as intermediate means. Since most computation-intensive algorithms exhibit regular or near-regular computation graphs [1], it is easier and more efficient to map each of these computation graphs onto an appropriate regular virtual array than onto a physical graph. The direct configuration strategy may be more difficult to implement and more inefficient than the indirect configuration strategy, since it has to map all (regular and irregular) computation graphs onto a physical graph. The direct configuration strategy, however, can provide more powerful mapping capability than the indirect configuration strategy. In the paper, functional reconfiguration with the direct configuration strategy is considered. A functional reconfiguration technique with the indirect configuration strategy is nothing but a combination of pre-existing functional and FT reconfiguration techniques.

After physical mappings of computation graphs have been obtained through functional configur-

ation, each of these computation graphs has to be loaded into the physical array for its execution. Functional loading, however, involves overhead, here called *graph-switching overhead*. The controller has to load arithmetic/logic and switching instructions assigned to each vertex of a computation graph into the memory of each cell in the physical array. However, it has been observed that interleaving computation and instruction loading with a single interconnection network in the physical array may contaminate information processed in the array [8], and may affect turnaround time. Hence, those instructions have to be loaded into each cell's memory between the time the physical array completes execution of an old algorithm and the time the array starts execution of a new algorithm, unless there exists an extra interconnection network that enables communication between an external host and the cells without interrupting computation in the physical array (e.g., [5,26]). The physical array cannot perform computation during the functional loading. It is a reasonable assumption that graph-switching overhead is tolerable, since each algorithm is computation-intensive and will need a sufficient amount of execution time, but it is desirable to reduce graph-switching overhead.

Functional reconfiguration requires three steps : an *analysis phase* to find a computation graph of an algorithm, a *configuration phase* to obtain a physical mapping, and a *loading phase* to actually load a computation graph into a physical array. Controller-driven configuration strategy is our choice for functional reconfiguration, as it allows for a practically unlimited number of computation graphs. The analysis phase is far beyond the scope of this paper. Only functional configuration and embedding are discussed here.

Three representative approaches proposed for dynamic functional reconfiguration are [5,26,27]. There are proposals that may be considered for static functional reconfiguration (e.g., [6,18,41]). However, they mention FT reconfiguration only and never consider how to handle computation graphs. In those approaches, once a physical array has been reconfigured into a target virtual array, computation graphs of all algorithms should be first mapped onto the virtual array. For algorithms whose communication patterns do not match well the interconnection topology of the virtual array, it is necessary to reconfigure the physical array into another virtual array suitable for those algorithms. Such a requirement imposes non-negligible array-embedding overhead, because a physical array cannot perform useful computation during the reconfiguration process. Even when the communication pattern of an algorithm matches well the interconnection pattern of the virtual array residing in a physical array, there is still high graph-switching overhead. Functional loading in those proposals requires complete interruption of a physical array and its controller.

Ⅳ. The relation between fault-tolerant and functional reconfigurations

The analysis of the FT and functional configuration tasks reveals an important relationship between FT and functional reconfigurations. The task of the FT configuration is to map the virtual array at the time of fault occurrence onto a physical graph. The task of the functional configuration is to map arbitrary computation graphs of various topologies onto a physical graph. We notice that FT and functional configuration tasks are not different in terms of their functionality. The difference between them is only in the objects to be handled (Fig.4).

In FT reconfiguration, a single virtual array of predefined interconnection topology and fixed size is allowed ; thus, an FT configuration algorithm is customized for the fixed virtual array and is often implemented as built-in hardware so that it can be performed in real time. Since an arbitrary number of computation graphs should be handled in functional reconfiguration, a functional configuration algorithm tends to be complex (e.g., [26]).

The configuration problems in FT and func-

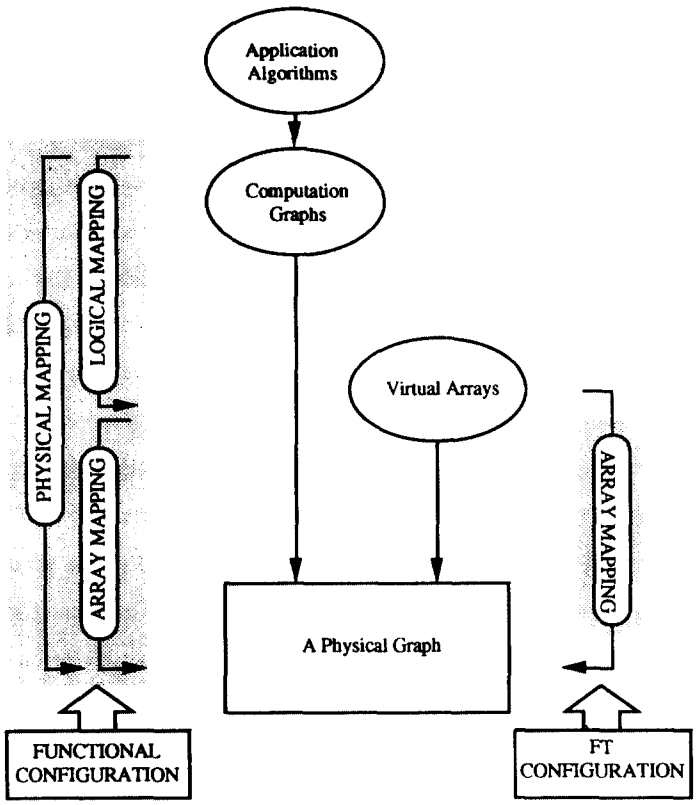

Fig 4. Functional Configuration versus FT Configuration

tional reconfigurations can be thoroughly understood by means of the *subgraph homeomorphism problem* [52-57]. Here we briefly outline a subgraph homeomorphism and relevant results. A more complete and detailed discussion of the SHP can be found in [58]. Let G and H be two graphs, both of which are multigraphs (or multidigraphs). In a *multigraph*, no loops are allowed but more than one edge can join two vertices. (In a *multidigraph*, no arc whose head and tail are the same vertex is allowed, but more than one arc can join two vertices.) A subgraph homeomorphism is a pair of 1-1 mappings (ϕ, θ), the first from vertices of H to vertices of G and the second from edges (arcs) of H to paths (directed paths) of G, requiring that a path (directed path) in G corresponding to an edge (arc) (x, y) in H go from $\phi(x)$ to $\phi(y)$ in G. The graph H is called an *input graph* and the graph G is called a *base graph*. We call ϕ a *vertex mapping* and θ an *edge mapping*. If the image of the edges of H are a set of paths which are vertex-disjoint up to end-vertices, the homeomorphism is a *vertex-disjoint homeomorphism* and we say that H is *vertex-disjoint homeomorphic* to a subgraph of G. If the set of paths is edge-disjoint, then H is *edge-disjoint homeomorphic* to a subgraph of G. The subgraph homeomorphism problem (SHP) is defined by :

Instance : An input graph H and a base graph G.
Question : Does G contain a subgraph homeomorphic to H?

An important class of problems related to the

SHP is the SHP for a fixed input graph [59]. Both the vertex-disjoint SHP and the edge-disjoint SHP are NP-complete for undirected and directed graphs [55]. Robertson and Seymour [60,61] showed that the SHP for a fixed input graph is in P-class for undirected graphs : however, as the authors themselves note, their results are of little practical significance, and that problem is still considered intractable. The SHP for a fixed input graph is NP-complete for directed graphs [59,62]. With the above background, we can define the FT configuration problem as the SHP for a fixed input graph and the functional configuration problem as the SHP. Thus, we can see that the FT configuration problem is in P-class though it is still intractable, and the functional configuration problem is NP-complete in its most general form.

V. Integrating FT and functional reconfiguration

The integrated reconfiguration on highly defective LSCAs requires powerful dynamic FT configuration capability. Any logical computing structure in the system, not just a single target virtual array, has to be reembedded into the physical array on the occurrence of operational faults. In addition, a functional configuration algorithm has to be capable of mapping an arbitrary computation graph onto a physical graph of a highly defective physical array. To obtain the above dynamic FT and functional reconfigurability, the basic requirement is that a cellular array architecture and an accompanying configuration method provide powerul connecting capability so that communication paths between non adjacent processors can be established by detouring many defective processors. Also, the time complexity of the configuration algorithm has to be polynomially bound, while achieving short communication delay, good survivability, high utilzation, and low area overhead.

To this goal, the nature of integrated reconfiguration on highly defective arrays is discussed in subsection 5.1. Testing and configuration strategies for integrated reconfiguration on highly defective arrays are covered in subsection 5.2, along with some simulation results.

5.1 Integrated reconfiguration on highly defective arrays

For integrated reconfiguration on LSCAs with many faulty processors, the object to be handled by an FT configuration algorithm should be any logical computing structure, while a functional configuration algorithm has to deal with a physical array that is no longer a fault-free perfect array. The difference between FT and functional configuration tasks vanishes, opening the integrated reconfiguration. In this paper, the configuration task in integrated reconfiguration is called a *unified configuration problem*. The unified configuration problem is the SHP, thus its computational complexity is NP-complete [58]. However, a general solution of the unified configuration problem is required since functional reconfiguration (and thus integrated reconfiguration) cannot be achieved otherwise. We need to develop an approximation algorithm with polynomial-time bound using techniques such as suggested in [63]. Having a general solution, nevertheless, does not prevent us from having better individual solutions for special cases of that problem. For example, individually-customized configuration algorithms can be written for some selected input graphs and can coexist with the general solution, so as to achieve better results for those selected graphs than the general solution can. This is useful when handling the typical topologies that are most commonly used, such as a linear array, a mesh, a binary tree, and a hexagonal array.

5.2 Control strategy for integrated reconfiguration

While the author shall continue to pursue self-

configuration strategies, the author is eager to see if other approaches can provide an immediate solution to the exponential complixity of the configuration, as it was introduced in [6]. Let us analyze the situation.

Since integrated reconfiguration requires that an arbitrary logical computing structure be embedded into a cellular array, an LSCA designed with the self-configuration strategy has to contain *programmable* self-configuration hardware(e.g., [6]). In this case, an individually customized self-configuration algorithm needs to be prepared for each logical computing structure so as to program self-configuration hardware. Preparing an individual self-configuration algorithm for each logical computing structure is very costly, because a virtually unlimited number of logical computing structures are allowed in integrated reconfiguration.

A more difficult problem is to keep a polynomial-time bound for each self-configuration algorithm. The paper mentioned that the unified configuration problem is NP-complete. In order to keep polynomial-time bound, each self-configuration algorithm should incorporate an ad hoc technique suitable for the corresponding logical computing structure. This task is not easy because a decision to try another configuration mapping has to be made at a cell based on local knowledge. Also, complex self-configuration algorithms have to be avoided, as they create a potentially prohibitive area overhead. The indirect configuration through a few, fixed number of intermediate virtual arrays may help overcome such difficulties. This technique, however, causes frequent exchange of intermediate virtual arrays, because a computation graph can be embedded into a physical array *only after* a suitable intermediate virtual array has been embedded into the physical array (e.g., [6,41]). Thus, the self-configuration strategy combined with the indirect configuration technique suffers from high array-embedding overhead.

The configuration and loading phases are inherently indistinguishable in self-configuration. A target logical computing structure is formed in a physical array while self-configuration is in progress. The self-configuration process cannot overlap computation with configuration, and wastes computing resources whenever self-configuration is performed. Another problem is that the controller can know only whether or not self-configuration succeeds (e.g., [18,41]), since self-configuration is an autonomous operation within a physical array. The controller has no way to obtain information on the physical mapping of a logical computing structure, essential for the assignment of functions to cells in programmable cellular arrays, except by polling the array.

The controller-driven configuration strategy does not impose a limit on the number of logical computing structures, and needs no extra hardware support. In addition, computation can be overlapped with configuration since configuration is done by the controller. Whereas the self-configuration strategy does not allow complex configuration algorithms, the controller-driven configuration strategy allows algorithms of high complexity [22]. Furthermore, it allows the development of a general configuration algorithm with polynomial-time bound that can handle arbitrary logical computing structures, which is, at this time, unattainable with the self-configuration strategy.

The above discussion shows that at least initially, the controller-driven configuration strategy is more feasible than the self-configuration strategy. However, it has the disadvantage of requiring global knowledge of a physical array, so that effective and efficient configuration be attained. The only way to obtain such information is to perform run-time tests [33]. The controller-driven testing strategy also involves serious overheads such as periodic testing long detection time. To *preserve* the benefits provided by the controller-driven configuration strategy, we need to combine it with the self-testing strategy in order to acquire the global knowledge as little

run-time tesing overhead as possible. Therefore, in the long range, the STCC control strategy is considered most suitable for integrated reconfiguration, provided a cellular array can be designed to support it.

A new cellular array architecture designed with the STCC strategy, called the ALFA processor array, has been proposed in [64], as the first attempt to tackle integrated reconfiguration. The ALFA array allows massive defects, and is reconfigured based on a heuristic, polynomially-bound configuration algorithm. Extensive simulations show that the ALFA array and the accompanying configuration algorithm achieve very good results overall, and demonstrate the feasibility of integrating FT and functional reconfigurations in defective LSCAs. Fig.5 shows the average size and utilization obtained on a 40×40 ALFA array for a linear array, a binary tree, a mesh, and a hexagonal array. Since no results are available for integrated reconfiguration at this time, the results have been compared with the results of the octally-connected massively fault-tolerant cellular array (MFCA) [6], which is by far the only approach proposed to handle massive defects. The MFCA adopts the STSC strategy, and the complexity of its self-configuration algorithms is exponential. No results are given for a hexagonal array in [6]. The comparison in Fig.5 (a), (b) and (c) shows the new results are clearly superior to the results of the MFCA. The reader is referred to [64] for a detailed discussion of the new approach and various simulation results

Ⅵ. Summary and conclusion

Many approaches have been proposed for FT reconfiguration. With the exception of the Massively Fault-Tolerant Cellular Array [6], their capability is limited to handle a small number of defects, despite the fact that LSCAs are very hard to make without many defects. The time complexity of configuration in the Massively Fault-Tolerant Cellular Array is currently exponential, and utilization and survivability is low (with the exception of linear computing structures). Production cost of manufacturing LSCAs favors the consideration of functional reconfiguration, however, very few realistic approaches have have been developed to date. A natural and challenging question is whether it is possible to integrate FT and functional reconfigurations in highly defective LSCAs, and if so, what are the requirements for integrated reconfiguration.

To answer this question, this paper took the first step by identifying the characteristics of FT and functional reconfigurations. The paper discussed control strategies, and distinguished different types of FT and functional reconfigurations. Among the tasks involved in reconfiguration, *configuration* is an essential task in FT and functional reconfigurations. This process maps one topology onto another topology. The paper has shown that the difference between FT and functional configuration tasks vanishes in integrated reconfiguration. The paper also discussed which control strategy is useful for integrated reconfiguration.

When designing an LSCA, we have to understand what type of reconfiguration is desired, and have to consider the issues of appropriate control strategy, locality (local communication), regularity (regular interconnection), modularity (identical components), massive fault-tolerance, polymorphism (customizability for a variety of computations), complexity of the configuration algorithm, survivability/utilization, communication delay, and scalability. An open question is which celluar array architecture is suitable for integrated reconfiguration in the presence of many defects, and how one can achieve low area overhead, short communication delay, high survivability/utilization, and polynomially-bound time complexity of configuration.

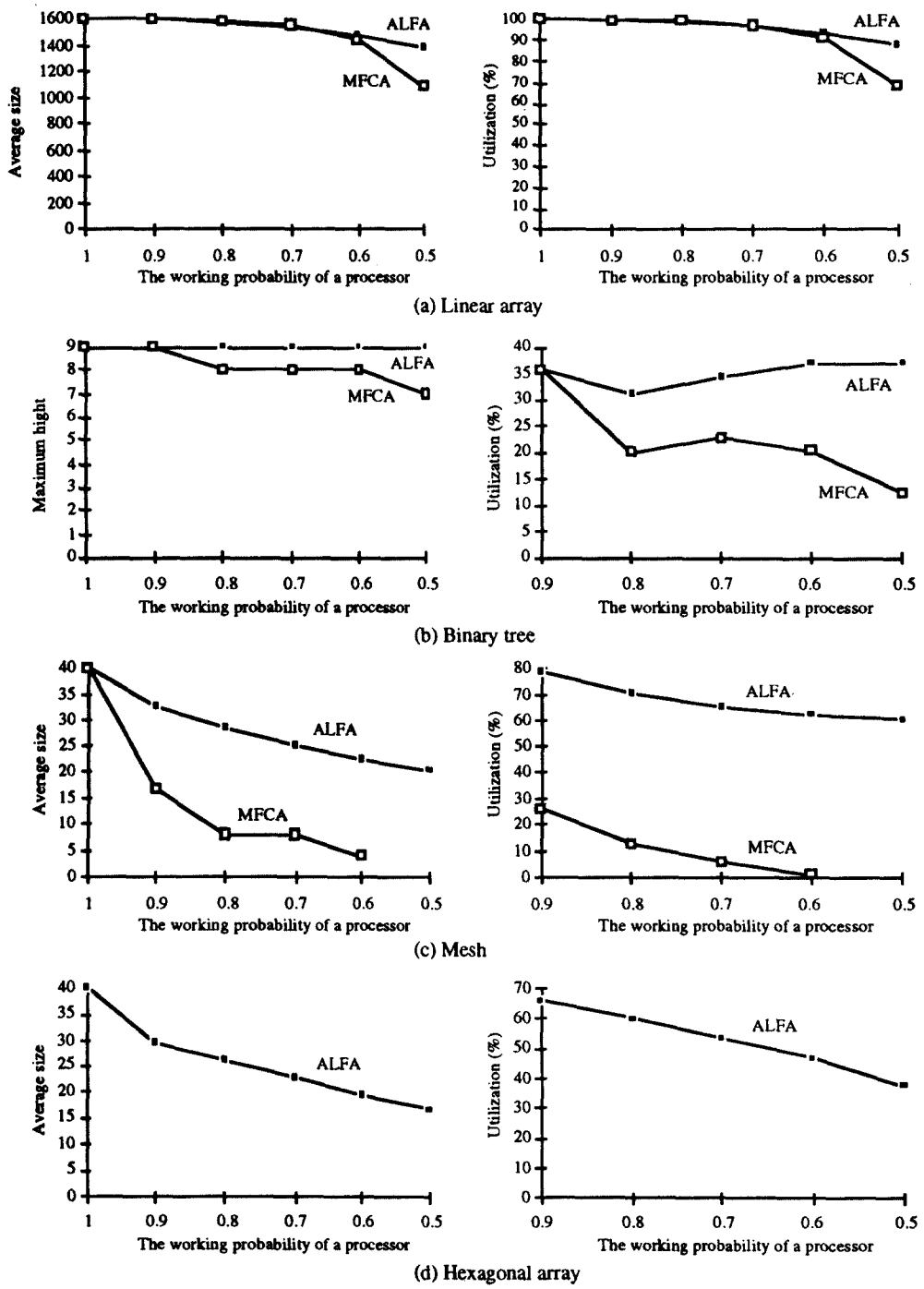

Fig 5. Simulation Results. In (c) and (d), the number n at the tick mark of the y axis implies $n \times n$.

References

1. S. Y. Kung, VLSI Array Processors, Prentice Hall, 1988
2. W. Maly, "Prospects for WSI : A Manufacturing Perspective," IEEE Computer, Vol.25, No.4, pp.58-65, April 1992
3. H. T. Kung, "Why Systolic Architectures?," IEEE Computer, Vol.15, No.1, pp.37-46, January 1982
4. S. Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors," IEEE Proceedings, Vol.72, No.7, pp.867-884, July 1984
5. L. Snyder, "Introduction to the Configurable Highly Parallel Computer," IEEE Computer, Vol.15, No.1, pp.47-56, January 1982
6. M. S. Lee and G. Frieder, "Self-configuration of Defective Cellular Arrays," Complex Systems, Vol.1, No.1, pp.81-105, February 1987
7. M. Chean and J. A. B. Fortes, "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," IEEE Computer, pp. 55-69, January 1990
8. R. C. Aubusson and I. Catt, "Wafer-Scale Integration-A Fault-Tolerant Procedure," IEEE Journal of Solid-State Circuits, Vol. SC-13, No.3, pp.339-344, June 1978
9. M. Berg and I. Koren, "On Switching Policies for Modular Redundancy Fault-Tolerant Computing Systems," IEEE Transactions on Computers, Vol.C-36, No.9, pp.1052-1062, September 1987
10. A. Boubekeur, et al, "Configuring a Wafer-Scale Two Dimensional Array of Single-Bit Processors," IEEE Computer, Vol.25, No.4, pp.29-40, April 1992
11. V. N. Doniants, V. G. Lazarev and R. Stefanelli, "Fault Tolerance of Cellular Processing Arrays : Algorithmic Methods for Yield Enhancement and Reliability," Microprocessing and Microprogramming 25, pp.113-118, 1989
12. D. Fussell and P., Varmann, "Fault-Tolerant Wafer-Scale Architectures for VLSI," Proc. 9th Annual Symposium on Computer Architecture, pp.190-198, 1982
13. D. Gordon, I. Koren and G. M. Silberman, "Restructuring Hexagonal Arrays of Processors in the Presence of Faults," Journal of VLSI and Computer Systems, Vol.2, No.1-2, pp.23-35, 1987
14. J. Greene and A. E. Gamal, "Configuration of VLSI Arrays in the Presence of Defects," JACM, Vol.31, No.4, pp.694-717, August 1984
15. K. S. Hedlund, "WASP-A Wafer-Scale Systolic Processor," ICCD 85, pp.665-671, October 1985
16. J. H. Kim and S. M. Reddy, "On the Design of Fault-Tolerant Two-Dimensional Systolic Arrays for Yield Enhancement," IEEE Transactions on Computers, Vol.38, No.4, pp.515-525, April 1989
17. I. Koren and D. K. Pradhan, "Modeling the Effect of Redundancy on Yield and Performance of VLSI Systems," IEEE Transactions on Computers, Vol.C-36, No.3, pp.344-355, March 1987
18. R. M. Lea and H. S. Bolouri, "Fault Tolerance : Step towards WSI," IEEE Proceedings, Vol.135, Part E. No.6, pp.289-297, November 1988
19. T. Leighton and C. E. Leiserson, "Wafer-Scale Integration of Systolic Arrays," IEEE Transactions on Computers, Vol. C-34, No.5, pp.448-461, May 1985
20. H. F. Li, R. Jayakumar and C. Lam, "Restructuring for Fault-Tolerant Systolic Arrays," IEEE Transactions on Computers, Vol.38, No.2, pp.307-311, February 1989
21. F. Lombardi, "Reconfiguration of hexagonal arrays by diagonal deletion," Integration, the VLSI Journal, Vol.6, pp.263-290, 1988
22. R. Negrini, M. Sami and R. Stefanelli, "Fault Tolerant Techniques for Array Structures used in Supercomputing," IEEE Computer, Vol.19, No.2, pp.78-87, February 1986

23. A. L. Rosenberg, "The Diogenes Approach to Testable Fault-Tolerant Arrays of Processors," IEEE Transactions on Computers, Vol. C-32, No.10, pp.902-910, October 1983
24. P. J. Varman and I. V. Ramakrishnan, "Optimal Matrix Multiplication on Fault-Tolerant VLSI Arrays," IEEE Transactions on Computers, Vol.38, No.2, pp.278-283, February 1989
25. M. Wang, M. Cutler and S. Y. H. Su, "Reconfiguration of VLSI/WSI Mesh Array Processors with Two-Level Redundancy," IEEE Transactions on Computers, Vol.38, No.4, pp. 547-554, April 1989
26. I. Koren, et al, "A Data-Driven VLSI Array for Arbitrary Algorithms," IEEE Computer, Vol.21, No.10, pp.30-43, October 1988
27. H. T. Kung, "iWarp multicomputer with an embedded switching network," Microprocessors and Microsystems, Vol.14, No.1, pp. 59-60, 1990
28. J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, Elsevier Science, 1976
29. N. Deo, Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall, 1974
30. F. Harary, Graph Theory, Addison-Wesley, 1969
31. J. B. Butcher and K. K. Johnstone, "Wafer Scale Integration," IEE Proceedings, Vol.135, part E, No.6, pp.281-288, November 1988
32. R. O. Carlson and C. A. Neugebauer, "Future Trends in Wafer Scale Integration," Proceedings of the IEEE, Vol.74, No.12, pp.1741-1752, December 1986
33. I. Koren and D. K. Pradhan, "Yield and Performance Enhancement through Redundancy in VLSI and WSI Multiprocessor Systems," Proceedings of IEEE, Vol.74, No.5, pp. 699-711, May 1986
34. R. Negrini and M. Sami, "Redundancy and Fault-Tolerance Aspects in VLSI Processing Architectures," COMPEURO 87, pp.3-7, May 1987
35. I. Koren and A. D. Singh, "Fault Tolerance in VLSI circuits," IEEE Computer, pp.73-83, July 1990
36. S. Y. Kung, S. N. Jean and C. W. Chang, "Fault-Tolerant Array Processors using single-track Switches," IEEE Transactions on Computers, Vol.38, No.4, pp.501-514, April 1989
37. K. Yamashita, et al, "A Wafer-Scale 170, 000-Gate FFT Processor with BUILT-IN TEST Circuits," IEEE Journal of Solid-State Circuits, Vol.23, No.2, pp.336-342, April 1988
38. H. Y. Youn and A. D. Singh, "On Implementing Large Binary Tree Architectures in VLSI and WSI," IEEE Transactions on Computers, Vol.38, No.4, pp.526-537, 1989
39. A. H. Anderson, J. I. Raffel, and P. W. Wyatt, "Wafer-Scale Integration Using Restructurable VLSI," IEEE Computer, Vol.25, No.4, pp.41-47, April 1992
40. W. R. Moore, "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield," Proceedings of the IEEE, Vol.74, No.5, pp.684-698, May 1986
41. I. Koren, "A Reconfigurable and Fault-Tolerant VLSI Multiprocessor Array," Proc. 8th Annual Sysmposium on Computer Architecture, pp.425-442, May 1981
42. S. Chau and D. Rennels, "Design Techniques for a Self-Checking Self-Exercising Processor," In Defect and Fault Tolerance in VLSI Systems, I. Koren ed., Vol.1, pp.191-202, 1989
43. M. C. Howells, R. Aitken and V. K. Agarwal, "Defect Tolerant Interconnects for VLSI," In Defect and Fault Tolerance in VLSI Systems, I. Koren ed., Vol.1, pp.65-76, 1989
44. R. Sharma and K. K. Saluja, "An Implementation and Analysis of a concurrent built-in self-test technique," IEEE International Symposium on Fault-Tolerant Computing, pp. 164-169, 1988
45. K. S. Hedlund and L. Snyder, "Systolic Architectures-A Wafer Scale Approach," ICCD 84, pp.604-610, October 1984

46. M. Annaratone, et al, "Architecture of Warp," COMPCON Spring 87, pp.264-267, February 1987
47. A. L. Fisher, et al, "The Architecture of a Programmable Systolic Chip," Journal of VLSI and Computer Systems, Vo.1, No.2, pp.153-169, 1984
48. D. L. Landis, N. Nigam, and J. W. Yoder, "Wafer-Scale Optimization Using Computational Availability," IEEE Computer, Vol.25, No.4, pp.66-71, April 1992
49. S. K. Rao, "Regular Iterative Algorithms and Their Implementations on Processor Arrays," Ph. D. thesis, Stanford University, Stanford, California, 1986
50. J. D. Ullman, Computational Aspects of VLSI, Computer Science Press, 1984
51. L. Snyder, "Parallel Programming and the Poker Programming Environment," IEEE Computer, Vol.17, No.7, pp.27-36, July 1984
52. T. Asano, "An approach to the Subgraph Homeomorphism Problem," Theoretical Computer Science 38, pp.249-267, 1985
53. S. Fortune, J. Hopcroft and J. Wyllie, "The Directed Subgraph Homeomorphism Problem," Theoretical Computer Science 10, pp. 111-121, 1980
54. S. Khuller, "Parallel Algorithms for the Subgraph Homeomorphism Problem," Algorithms and Data Structures, F. Dehne, J. R. Sack and N. Santoro (Eds), Lecture Notes in Computer Science 382, pp.303-315, 1989
55. A. S. LaPaugh and R. L. Rivest, "The Subgraph Homeomorphism Problem," Jouranl of Computer and System Sciences 20, pp. 133-149, 1980
56. A. Lingas and A. Proskurowski, "Fast Parallel Algorithms for the Subgraph Homeomorphism and the Subgraph Isomorphism Problem for Classes of Planar Graphs," Foundations of Software Technology and Theoretical Computer Science, K. V. Nori (Ed.), Lecture Notes in Computer Science 287, pp. 79-94, 1987
57. K. E. Strange and B. Toft, "An Introduction to the Subgraph Homeomorphism Problem," Proceedings of the third Czechoslovak Symposium on Graph Theory, Graphs and Other Combinatorial Topics, Prague, pp.296-301, 1982
58. J. Han, "Three Types of Reconfiguration in Large Scale Cellular Arrays : Their Mathematical Characterization and Computational Complexity," submitted
59. D. S. Johnson, "The NP-Completeness Column : An Ongoing Guide," Journal of Algorithms, Vol.8, pp.285-303, 1987
60. N. Robertson and P. D. Seymour, "Graph Minors-A Survey," Survey in Combinatorics, I. Anderson (Ed.), Cambridge University Press, pp.153-171, 1985
61. N. Robertson and P. D. Seymour, "Graph Minors. VII. Disjoint Paths on a Surface," Journal of Combinatorial Theory, Series B 45, pp.212-254, 1988
62. N. Robertson and P. D. Seymour, "Disjoint Paths-A Survey," SIAM J. on Algebraic and Discrete Methods, Vol.6, pp.300-305, 1985
63. M. R. Garey and D. S. Johnson, "Computers and Intractability : A Guide to the Theory of NP-Completeness," Bell Laboratories, 1979
64. J. Han and G. Frieder, "Integrating Fault-Tolerant And Functional Reconfigurations On Defective Processor Arrays," to appear in the InfoScience '93, Seoul, Korea, October 1993

韓 在 一(Jae-Il Han) 정회원
1958년 2월 14일생
1980년 : 연세대학교 수학과 졸업 (이학사)
1986년 : 미국 Syracuse University 전산학과 졸업(석사)
1992년 : 미국 Syracuse University 전산학과 졸업(박사)
1993년 3월~현재 : 한국전자통신연구소 분산처리 연구실 선임연구원
※주관심분야 : 병렬및 분산 처리, 고장 감내 시스템, 분산 시스템 프로그래밍 언어, Concurrent Programming