

A DEFECT TOLERANT SELF-ORGANIZING NANOSCALE SIMD ARCHITECTURE

Discussion Leader: Yiwen Shi

Scribe: Elif Alpaslan

MOTIVATION:

The decrease in transistor sizes due to either CMOS scaling or emerging nanotechnologies resulted to an increase in defect densities, contributing to the exponentially increasing cost of top-down lithography. But regardless of fabrication methodology, defect tolerant architectures are essential for future increased device densities. The need for the defect tolerant architectures is the motivation for the paper where they have developed a defect tolerant SIMD architecture.

INTRODUCTION:

As transistor sizes get smaller and smaller with CMOS scaling technologies it introduced many problems like: manufacturing defect rate increase, power density, process variability, transient faults, bulk silicon limits, rising test cost etc. . Some short-term solutions have been introduced at architectural level, but the need to explore long term alternatives to CMOS devices and fabrication techniques is necessary. DNA-based self-assembly of nanoscale components is one promising alternative. One important advantage of this technique is the precise control within small area which allows the construction of large number of small nodes. On the other hand, with lack of control over placement and orientation of nodes we might end up with a random network of nodes that contain defective nodes and links. The challenge for a computer architects is to efficiently exploit the computational power of the large number of nodes while beating loss of precise control over the entire fabrication process and high defect rates.

The work described in this paper explores a defect tolerant SIMD architecture. The key feature of their design is the ability of large number of limited capability nodes

with high defect rates (up to 30%) to self-organize into a set of SIMD processing elements. The fundamental building block of this approach is a relatively small node which operates asynchronously. Initially there is a configuration phase that isolates defective nodes from not defective ones and allows groups of non defective nodes to self-organize into SIMD processing elements (PEs).

SELF-ASSEMBLED NANOSCALE SYSTEMS

Self assembly of nano-electronic devices is a lower-cost alternative to top-down manufacturing. DNA-based self assembly is one example for self-assembled nanoscale systems, where precise binding rules of DNA with nanoscale devices help to build computing systems. Electronic circuits are placed on a DNA grid and a simple unit cell is replicated on a large scale to build a circuit. Ability to control the placement of the electric devices (e.g., carbon nanotubes or silicon nanowires at specific points on DNA scaffold to form the circuit.) is a key requirement for this process.

The example in Figure 1 the patterned DNA (letter A representation with lights) does not represent anything but the precise control of writing letter A with 20 nm pitch. We can think the lights as the places where we can put transistors. The paper does not explain how the letter A is formed on DNA grid and in the class some questions came about the formation of the letter A. One question that class talked about was, if we think DNA as grid and carbon nanotubes as the dots, what happens when we remove DNA scaffold? Will the carbon nanotubes stay in their same places? Or does this structure operate at high Temperature? We also don't about the medium that they used to form the letter A. All of these questions come up in the discussion of class time.

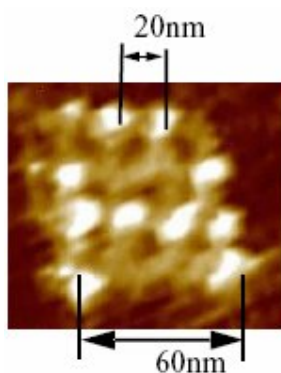


FIGURE 1.
Patterned DNA [26]

The placement and routing of the circuits are limited because of the current assumption of limited (only 2 layers) number of metal interconnects within the lattice. If you look to cross-sectional view of the lattice depicted in Figure 2, the conducting metallic planes separated by insulating layers provide power and ground to the circuit.

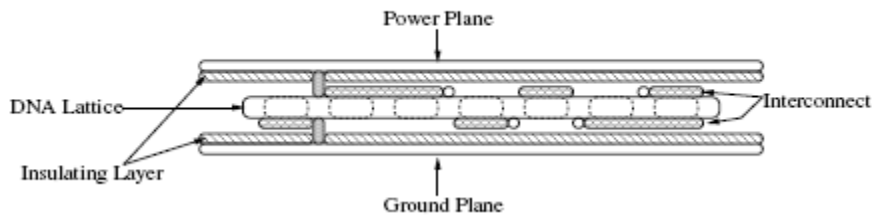


Figure 2: Lattice with two levels of interconnects, and connections to Vdd and Gnd

Current self-assembly processes produce limited size DNA grid which results in a limited circuit size. But we can construct many nodes by parallel-nature of assembly process and the nodes then might be linked together by self-assembled conducting nanowires. The proposed self-assembly method will result in a random network of nodes that contains defective nodes and links. The communication with external CMOS circuitry occurs through vias and anchor nodes.

SYSTEM OVERVIEW

The goal of this work is to build a defect tolerant computing system with a random network of nodes using new solutions of known techniques and at the same time still achieve same performance comparable to CMOS based system. The system proposed in this paper is called “Self-Organizing SIMD Architecture” (SOSA). SOSA supports a three operand register-based ISA with predicated execution and explicit PE-Shift Instructions to move data between PEs and communicate with external controller.

One key feature of this approach is that each self-assembled node is a fully asynchronous circuit and there is no global clock to synchronize data transfers between or within the nodes. The asynchronous feature of the SOSA approach is discussed in the class. The class discussed whether there is a possibility to make a synchronous design with this approach. For a synchronous design we need to route clock to each node in your design, but routing of the clock is not easy. We need to also remember that with this

approach we are limited to use only two metal layers, maybe with increased number of metal layers can make routing problem easier and lead to synchronous designs.

Another feature of this approach is that each link supports low bandwidth asynchronous communication that transfers 1 data bit per handshake. So this tells us basically SIMD is bandwidth limited. In addition to these feature, support for three virtual channels are added so deadlock free routing can be supported.

Configuration process is initiated from an anchor, records defective nodes and connects functional nodes in a broadcast tree. There are total of two ways to configure the system:

a) monolithic way = all nodes in one logical ring

An advantage of this implementation is that the anchors can be used to speed up PE configuration and data input/output by serving taps into logical ring, on the other hand the disadvantage of this implementation is that an anchor can not partition a PE.

b) multiple independent logical rings

This implementation achieves space partitioning by running the configuration algorithm from multiple anchors to create independent cells.

In the configuration logic part, the class questioned about whether there is a logic to pick up the anchor node. Because the configurations stage begins with a anchor node, meaning that the node that is going to be an anchor node must be a non-defective node. "Reverse path forwarding" algorithm is an answer for deciding the anchor node. In the paper, a generic node is chosen somehow (not explained) to be anchor node and everything starts with that node. It might be the case that the node that is chosen as an anchor might not be optimal for arriving the live nodes.

NODE MICROARCHOTECTURE

If you want to increase your performance the design of the nodes are critical and important. The tradeoff between circuit size and functionality has to be made clearly when designing nodes of your circuits. Instruction execution and sequencing within a node are asynchronous so we can reduce the area of total circuit and power consumption which are results of routing clock signal. Each node is a small circuit

that can communicate between at most 4 neighbors, stores small amount of state and performs simple operations.

Data Path

Each node has a simple data path which comprises of 1 bit ALU, 32 bit register file and a data buffer. The decision about the width of the registers has to be made in a smart way, because with the increase in register width we can reduce the total number of registers available to program but on the other hand the amount of work that has been done by a register will increase.

Control

Control logic has basically 2 parts: Configuration part and run-time control logic part. The first part is configuration part where control registers are responsible for defect isolation and defect testing. The second part is the run-time control logic which is used to decode and execute instructions. In order to reduce the design complexity they use micro-coded control logic with each instruction divided into multiple microinstructions, but with this approach it takes longer to read the whole instruction so the latency will be increased. All arriving micro-instructions are first sent to an instruction buffer where they wait for previous instructions to be ready then they are executed. So basically they form two stage pipeline. With this approach execution is serialized, because going from one instruction to another one is done serially, but on the other hand you are executing the instructions on several nodes. The size of the instruction buffer size is a design tradeoff because with the increase in instruction buffer size you can improve your performance but on the other hand you can cause greater contention on network because instruction and data have to share link bandwidth. Results from simulations show that the single entry instruction buffer is the best tradeoff between improving the performance and minimizing the design complexity.

CIRCUIT SIZE AND POWER ESTIMATES

The used the energy*delay product for carbon nanotube field effect transistor circuits to estimate the power consumption. Results show that the estimated power consumption is much less the ones that correspond to current processors. But the

paper does not give to much information about their estimation on calculating power consumed by circuit.

EVALUATION

In their evaluation part of their design they compare SOSA performance to other four architectures.

- A Pentium 4 (P4) (3 GHz, 1MB L2, 1GB RAM)
- An ideal out-of-order superscalar (I-SS)
- An ideal 16-way CMP (16-CMP)
- An ideal implementation of SOSA (I-SOSA) (unit instruction execution latencies, no communication overhead)

They run these different architectures on six different programs and the general results is that SOSA achieves a good performance on benchmarks that have data parallelism and it can tolerate high defect rate on nodes.