- 1. What is Evolvable Hardware?
- 2. History and Motivation of Cube Calculus Machines and Logic Machines
- 3. Evolving in Hardware or Learning in Hardware?
- 4. Variants of Cube Calculus
- 5. Cube Calculus Machines
- 6. Evaluation of previous Cube Calculus Machines

Learning in Hardware using Symbolic Methods based on Multiple-Valued Logic

- This set of slides includes a general information about learning in hardware and motivation for building Cube Calculus Machines

- It is not necessary to successfully complete the project

Real time symbolic learning

- SAT, graph coloring, covering, equation solving in MV algebras
- Image processing, pattern recognition, speech processing, language understanding, sensor integration, WWW technologies, anti-terrorist biometric technologies, military and aerospace (Ulug/Bowen, GE - extended cube algebra), DNF minimization for real time learning, Ventura Quantum DNF minimization, morphological and relational algebras, data base algebras.

Plan

- 1. What is evolvable hardware?
- 2. Learning in Hardware not Evolutionary Hardware
- 3. Symbolic Learning in Hardware using MVL
- 4. Cube Calculus (binary, mv, generalized mv, simplified binary, rough sets)
- 5. Cube Calculus in Hardware (CCM, Decomposition Machine, Rough Set Machine)
- 6. Cube Calculus in Reconfigurable Hardware
  - arbitrary word length
  - pipelining and parallelism, scalable
  - selecting subset of operation from a repertoire

WHAT IS EVOLVABLE HARDWARE?

Demonstration of Learning Hardware in Robotics

- Learning is done with the human as the feedback loop.
- The set of sequences is incomplete, so the machine performs the generalization automatically. Adding or removing new rules, by the human supervisor or automatically/randomly, will change the behavior.
- Mimique the human's behaviors seen by the camera
- Like the Furby toy, but with real learning.
- Capable of building its own "world model" and internal model with unlimited behaviors.

WHAT IS EVOLVABLE HARDWARE?

- This talk reviews the in the domain of EHW in years 1989 - 1999 and points out some fundamental open research issues.
- What Is Evolvable Hardware (EHW)
- EHW as an Alternative to Electronic Circuit Design
- EHW as an Adaptive System
- Other EHW-Related Work
- Evolvable Hardware versus Learning Hardware
- Learning Multi-Valued Functions
- Universal Logic Machine - Current PSU approach to Learning Hardware
- Our Proposed Extensions: Learning Finite State Machines.
- Concluding Remarks

WHAT IS EVOLVABLE HARDWARE ? (cont)

- There are different views on what EHW is, depending on the purpose of EHW.
- EHW can be regarded as “applications of evolutionary techniques to circuit synthesis." (A. Hirst)
- EHW is hardware which is capable of on-line adaptation through reconfiguring its architecture dynamically and autonomously. (T. Higuchi et al.).
- EHW is Genetic Algorithm realized in hardware (DeGaris). (Intrinsic Evolvable Hardware).

LEARNING IS MORE GENERAL THAN EVOLVING

- Learning is more general than evolving.
- Evolving is learning by Nature: blind, random, chaotic.
- Learning is any kind of behavior that improves something.
- Learning Hardware is any kind of hardware system that can change itself and its future behavior dynamically and autonomously by interacting with its environment.
- EHW is a child of the marriage between evolutionary computation techniques and electronic hardware.
- LH is a child of the marriage of Machine Learning and hardware (so far, electronic, but see Hanyu et al for DNA and molecular computing).

EHW AS AN ALTERNATIVE TO ELECTRONIC CIRCUIT DESIGN

- Using EAs to design VLSI chips and boards has a 12 year long history.
- Used in Digital and Analog design; (mixed?).
- Few examples:
  - Evolving Hardware Description Language (HDL) programs.
  - Evolving Electronically Programmable Logic Devices (EPLDs).
  - Evolving analog circuits.
  - Unconstrained evolution of an electronic oscillator (Adrian Thompson).
  - Generalized Reed-Muller Logic using GA (Karen Dill).
  - Arbitrary Tree logic networks using GP (Karen Dill).

TWO MAJOR APPROACHES

- Early and some of the recent work related to EHW only dealt with optimization of VLSI circuits, such as cell placement, logic minimization and compaction of symbolic layout.
- Circuit functions were not designed/evolved by EAs.
- Recent work concentrates on evolving circuit architectures and thus functions.
- Two major approaches have been used:
  - Indirect Approach,
  - Direct Approach.

INDIRECT APPROACH TO EHW CIRCUIT DESIGN

- The indirect approach does not evolve hardware directly, but evolves an intermediate representation (such as trees) which species hardware circuits.
- Evolving digital circuits.
- For example, SFL (Structured Function Description Language) programs (represented by production trees) can be evolved by a genetic algorithm. A binary adder which considers all 4-bit numbers was evolved successfully.
- Evolving analog circuits.
- For example, Koza ’ s work on evolving a low-pass “brick wall" filter, an asymmetric bandpass filter, an amplifier, etc. Trees were used to represent circuits. The results were competitive with human designs.
- href="http://www-cs-faculty.stanford.edu/ koza/#anchor5384423"

DIRECT APPROACH TO EHW CIRCUIT DESIGN - GATE LEVEL

- The direct approach evolves hardware circuit's architecture bits directly. It works well only with reconfigurable hardware, such as
- FPGA (field programmable gate array) from "http://www.xilinx.com/" (Xilinx).
- The gate level evolution implies that the “atomic" hardware functional units are logical gates like AND , OR , and NOT . The evolution is used to search for different combinations of these gates.
- Typical examples include XOR, counters, FSMs (Finite State Machines), multiplexors, and an electronic oscillator.
- One argument for the direct approach is to exploit hardware resources by unconstrained hardware evolution.

DIRECT APPROACH TO EHW CIRCUIT DESIGN - FUNCTIONAL LEVEL

- The gate level evolution runs into the scalability problem quickly.
- The function level evolution uses high-level functions such as addition, multiplication, sine, cosine, etc., and thus is much more powerful.
- Typical examples: two-spiral, Iris, FSMs, image rotation.
- The work is better viewed as an attempt towards adaptive hardware , rather than as a design alternative.

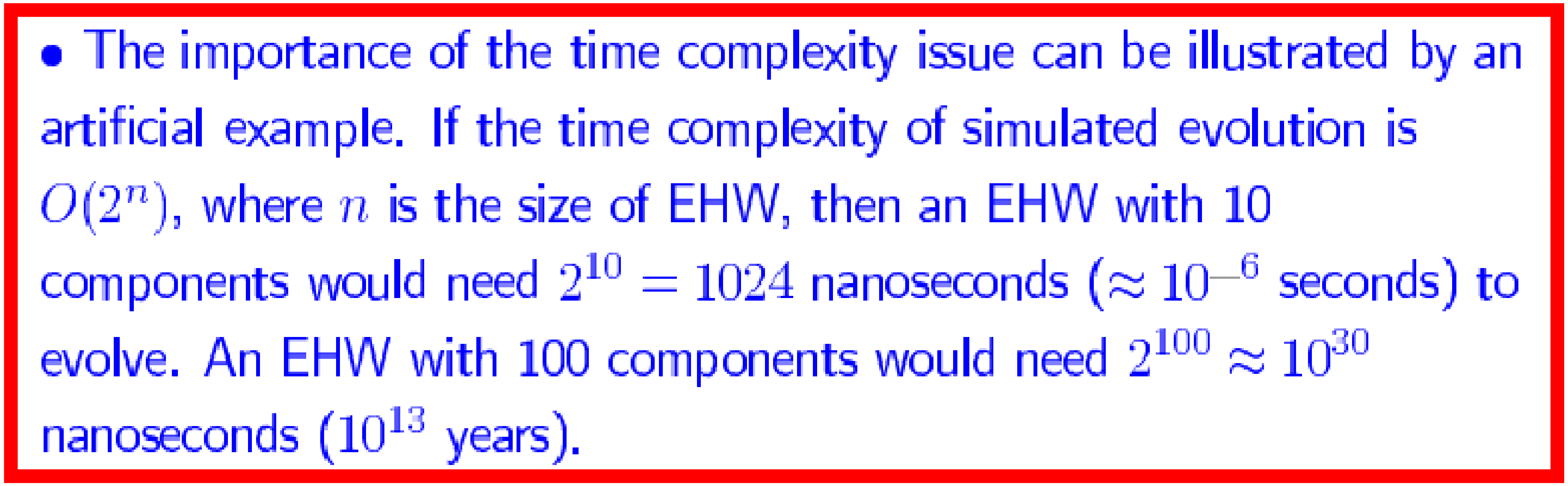
ADVANTAGES OF EVOLUTIONARY DESIGN

- Explores a larger design space and thus may be able to discover novel designs.
- Does not assume a priori knowledge and thus can be applied to various domains.
- Does not require exact specification and thus can design complex systems which cannot be handled by conventional specification-based design approach.
- However, constraints and special requirements could be imposed on the evolution if necessary through the fitness function and chromosome representation.
- Some analog circuits might be too difficult (or costly) to design by human experts.

SCALABILITY OF EHW

- Scalability of the algorithm: Time complexity of the EA for EHW?
- Scalability of the representation: Size of chromosomes vs. Size of EHW?
- Time is more crucial since the size of chromosome (space) is usually polynomial in the size of EHW circuits.

WILL ELECTRONIC SPEED SOLVE THE SCALABILITY PROBLEM?

- There have been some expectations that the speed of simulated evolution would not be a problem in a few years as faster VLSI chips come out.
- This statement can be misleading. Electronic speed is not a solution to the scalability problem. The scalability problem has to be addressed at the fundamental level.
- 

• The importance of the time complexity issue can be illustrated by an artificial example. If the time complexity of simulated evolution is $O(2^n)$, where n is the size of EHW, then an EHW with 10 components would need $2^{10} = 1024$ nanoseconds ($\approx 10^{-6}$ seconds) to evolve. An EHW with 100 components would need $2^{100} \approx 10^{30}$ nanoseconds (10^{13} years).

CIRCUIT VERIFICATION/TEST AND FITNESS FUNCTION

- How to verify the correctness of EHW? How to find a fitness function which guarantee the correctness of EHW?
- For example, if all 4-bit numbers have been correctly added, would all 5-bit, 6-bit, etc., numbers be added correctly by the same circuit?
- Exploiting hardware resources is attractive. Has an EHW exploit something totally irrelevant, such as room temperature or minor Earth movement?
- Is it practical to test all possible situations in which an EHW might be used?
- How robust is EHW to minor environmental changes? Does it degrade gracefully?
- When to stop simulated evolution? How to know whether a correct circuit has been evolved?

EHW AS AN ADAPTIVE SYSTEM

- Current work on adaptive EHW can be classified into two major categories:
  - EHW controllers.
  - EHW recognizers and classifiers.

EHW CONTROLLERS

- A number of control tasks can be performed by EHW, e.g., ATM control and robot control among others.
- Some examples:
  - Evolving an artificial ant to follow the John Muir Trail in simulation.
  - Evolving a wall following robot in a simulated environment, “virtual reality". "http://www.cogs.susx.ac.uk/users/adrianth/" .
  - Evolving an ATM traffic shaper.
  - Evolving an adaptive equalizer.

EHW RECOGNIZERS AND CLASSIFIERS

- Evolving FPGA to perform learning tasks, such as letter recognition, the comparator in a V-shape ditch tracer, two-spiral, Iris, FSMs, etc.
- Unlike most other studies, generalization is explicitly emphasized here.
- A complexity (regularization) term was included in the fitness evaluation function.

OTHER EHW-RELATED WORK

- Self-reproduction and self-repair hardware at Logic Systems Laboratory (LSL), Computer Science Department, Swiss Federal Institute of Technology - Lausanne. http://lslsun5.ep.ch/" .
- Artificial brains.
- CAM-BRAIN (CBM) from ATR's Department 6 (Evolutionary Systems) "http://www.hip.atr.co.jp/x/ATRCAM8" .
- Artificial Brain Systems at RIKEN. (No hardware implementation.) "http://www.bip.riken.go.jp/absl/Welcome.html" .

SOME CHALLENGES TO ADAPTIVE EHW

- Scalability: Efficiency of simulated evolution.
- Generalization: Dealing with new environments.
- Disaster prevention in fitness evaluation during on-line adaptation.
- On-line adaptation: incremental evolution/learning.

A BEHAVIORAL VIEW TOWARDS EHW

- What is being evolved? A circuit or the circuit's behaviors? In other words, what is actually being evaluated by a fitness function?
- Is it genetic evolution or behavioral evolution?
- Claim: It is EHW behavior, not its circuitry, that is being evolved.
- Some consequences of taking the behavioral view towards EHW:
  - 1. The environment is crucial. Generalization should be discussed with respect to environments.
  - 2. The role of crossover needs to be re-evaluated.

CONCLUDING REMARKS ON EVOLVABLE HARDWARE

- Population-based learning (simulated evolution) is good at following slow environmental changes, but not at real-time on-line adaptation.
- Individual learning should be introduced.
- There is some existing work on EANNs and GP which may be useful for function-level EHW, e.g., mutations and other techniques for maintaining behavioral links between parents and their offspring.
- Co-evolution is a very promising approach to deal with the problem of fitness evaluation. That is, co-evolution can be used to generate changing and challenging environments.

FURTHER REMARKS

- Evolutionary design of digital circuits would not be able to compete with the conventional approach.
- Evolutionary design of analog circuits needs to address the issues of circuit verification and robustness.
- Adaptive EHW has most potentials, but would need individual learning to implement on-line learning.
- The most profitable application domains for EHW would be those which are very complex but highly specialized.

WWW RESOURCES

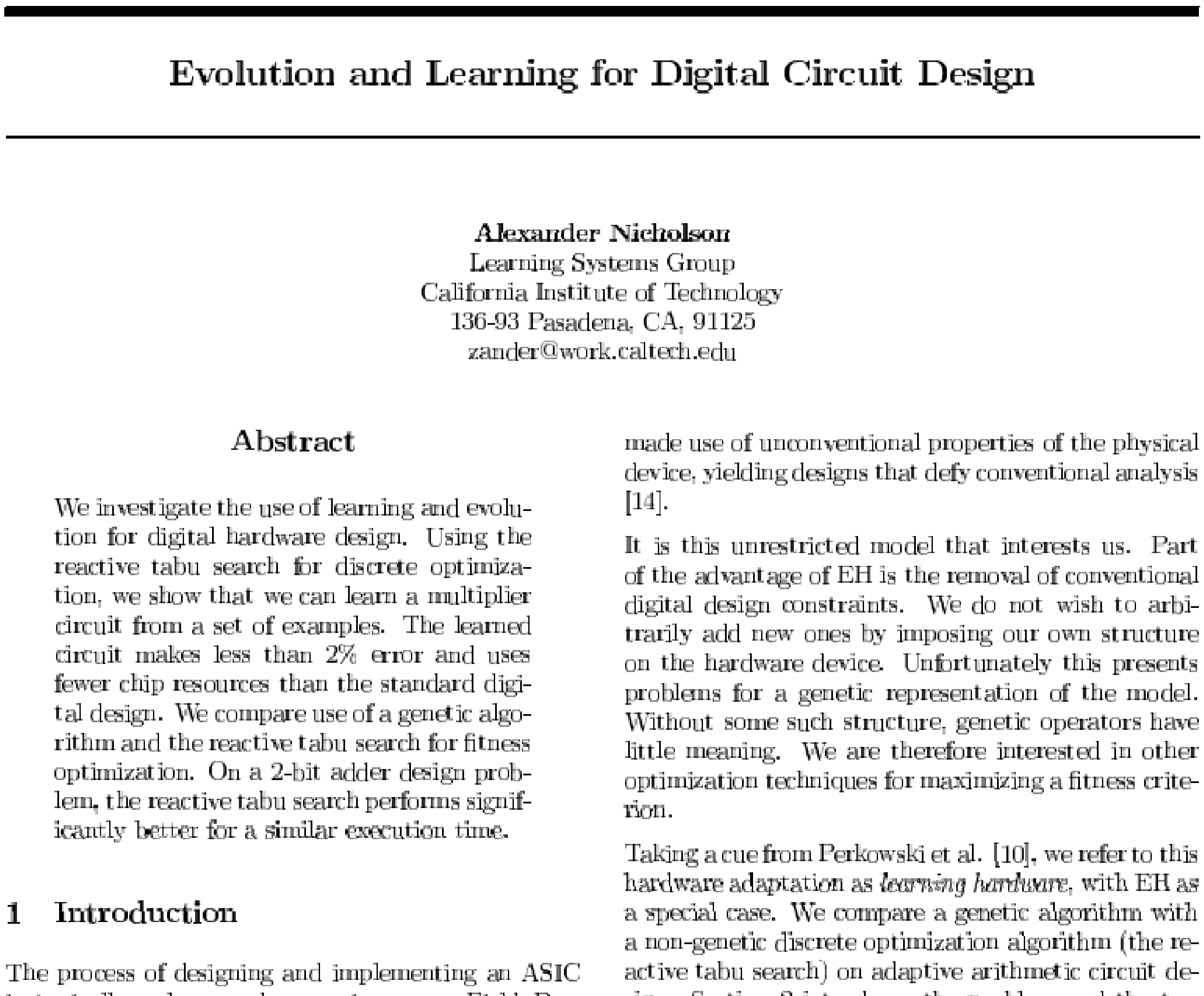
- The following papers are available on-line.
- X. Yao and T. Higuchi, “Promises and Challenges of Evolvable Hardware," Submitted to ICES'96. (Available as "ftp://www.cs.adfa.oz.au/pub/xin/ices96-challenge.ps.gz" .

History and Motivation for Cube Calculus Machines and Logic Machines

History and Motivation

- ICCAD 85 - our paper about hardware Logic Design Machine that was solving the following problems:
  - satisfiability
  - graph coloring
  - set covering
  - tautology
- ISMVL 1992 - our paper that generalized cube calculus for MVL and showed many other operations and applications as well. Reconfigurable
- ISCAS 92 and about 6 other conferences - variants
- ULSI 97 - ESOP and SAT - reconfigurable

Tabu search for learning in reconfigurable hardware



Evolution and Learning for Digital Circuit Design

Alexander Nicholson
Learning Systems Group
California Institute of Technology
136-93 Pasadena, CA, 91125
zander@work.caltech.edu

Abstract

We investigate the use of learning and evolution for digital hardware design. Using the reactive tabu search for discrete optimization, we show that we can learn a multiplier circuit from a set of examples. The learned circuit makes less than 2% error and uses fewer chip resources than the standard digital design. We compare use of a genetic algorithm and the reactive tabu search for fitness optimization. On a 2-bit adder design problem, the reactive tabu search performs significantly better for a similar execution time.

1 Introduction

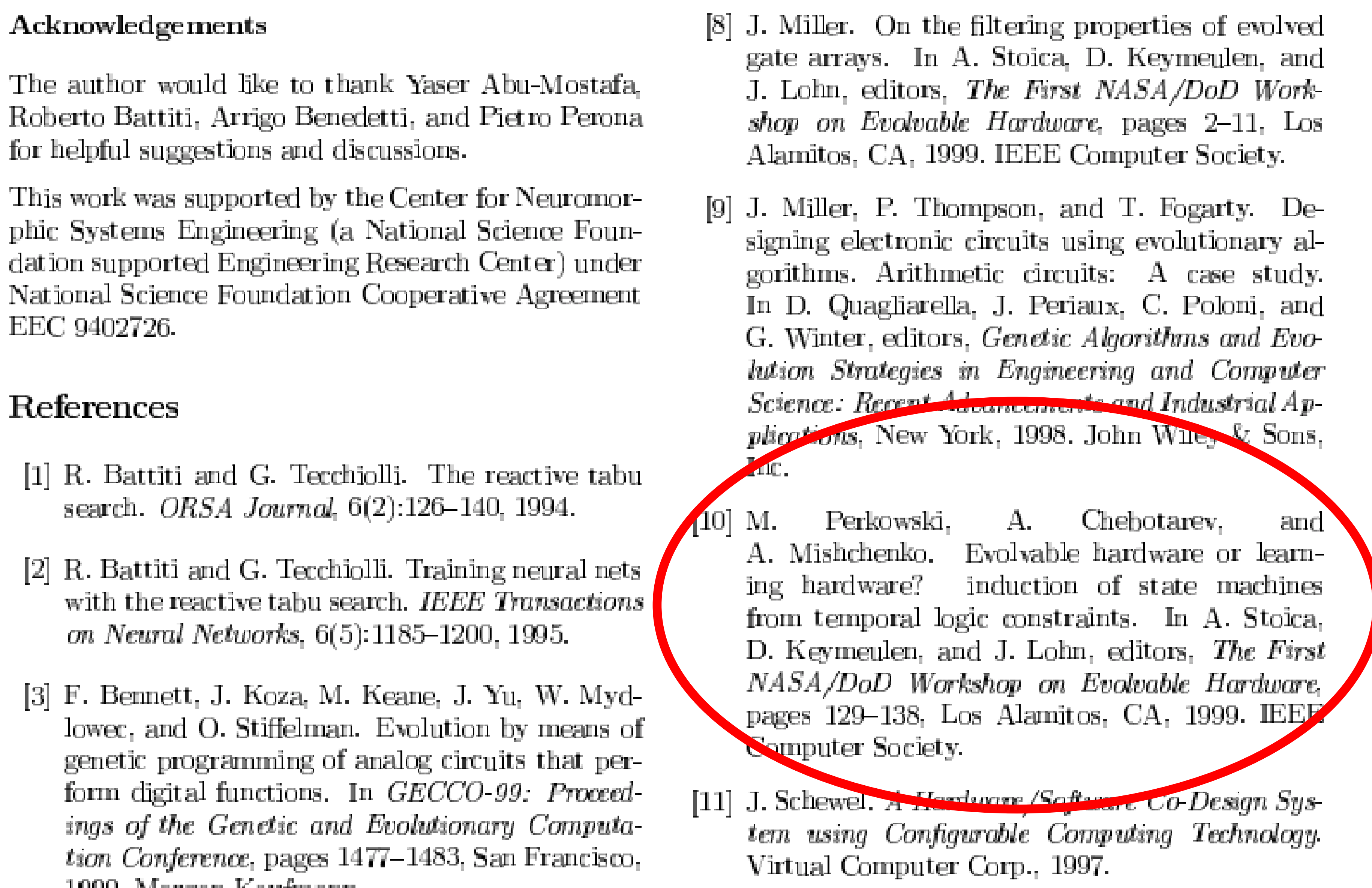
The process of designing and implementing an ASIC

made use of unconventional properties of the physical device, yielding designs that defy conventional analysis [14].

It is this unrestricted model that interests us. Part of the advantage of EH is the removal of conventional digital design constraints. We do not wish to arbitrarily add new ones by imposing our own structure on the hardware device. Unfortunately this presents problems for a genetic representation of the model. Without some such structure, genetic operators have little meaning. We are therefore interested in other optimization techniques for maximizing a fitness criterion.

Taking a cue from Perkowski et al. [10], we refer to this hardware adaptation as *learning hardware*, with EH as a special case. We compare a genetic algorithm with a non-genetic discrete optimization algorithm (the reactive tabu search) on adaptive arithmetic circuit de-

Student of Abu-Mostafa



Acknowledgements

The author would like to thank Yaser Abu-Mostafa, Roberto Battiti, Arrigo Benedetti, and Pietro Perona for helpful suggestions and discussions.

This work was supported by the Center for Neuromorphic Systems Engineering (a National Science Foundation supported Engineering Research Center) under National Science Foundation Cooperative Agreement EEC 9402726.

References

[1] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal*, 6(2):126–140, 1994.

[2] R. Battiti and G. Tecchiolli. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6(5):1185–1200, 1995.

[3] F. Bennett, J. Koza, M. Keane, J. Yu, W. Mydlowec, and O. Stiffelman. Evolution by means of genetic programming of analog circuits that perform digital functions. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1477–1483, San Francisco, 1999. Morgan Kaufmann.

[8] J. Miller. On the filtering properties of evolved gate arrays. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *The First NASA/DoD Workshop on Evolvable Hardware*, pages 2–11, Los Alamitos, CA, 1999. IEEE Computer Society.

[9] J. Miller, P. Thompson, and T. Fogarty. Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study. In D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications*, New York, 1998. John Wiley & Sons, Inc.

[10] M. Perkowski, A. Chebotarev, and A. Mishchenko. Evolvable hardware or learning hardware? induction of state machines from temporal logic constraints. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *The First NASA/DoD Workshop on Evolvable Hardware*, pages 129–138, Los Alamitos, CA, 1999. IEEE Computer Society.

[11] J. Schewel. *A Hardware/Software Co-Design System using Configurable Computing Technology*. Virtual Computer Corp., 1997.

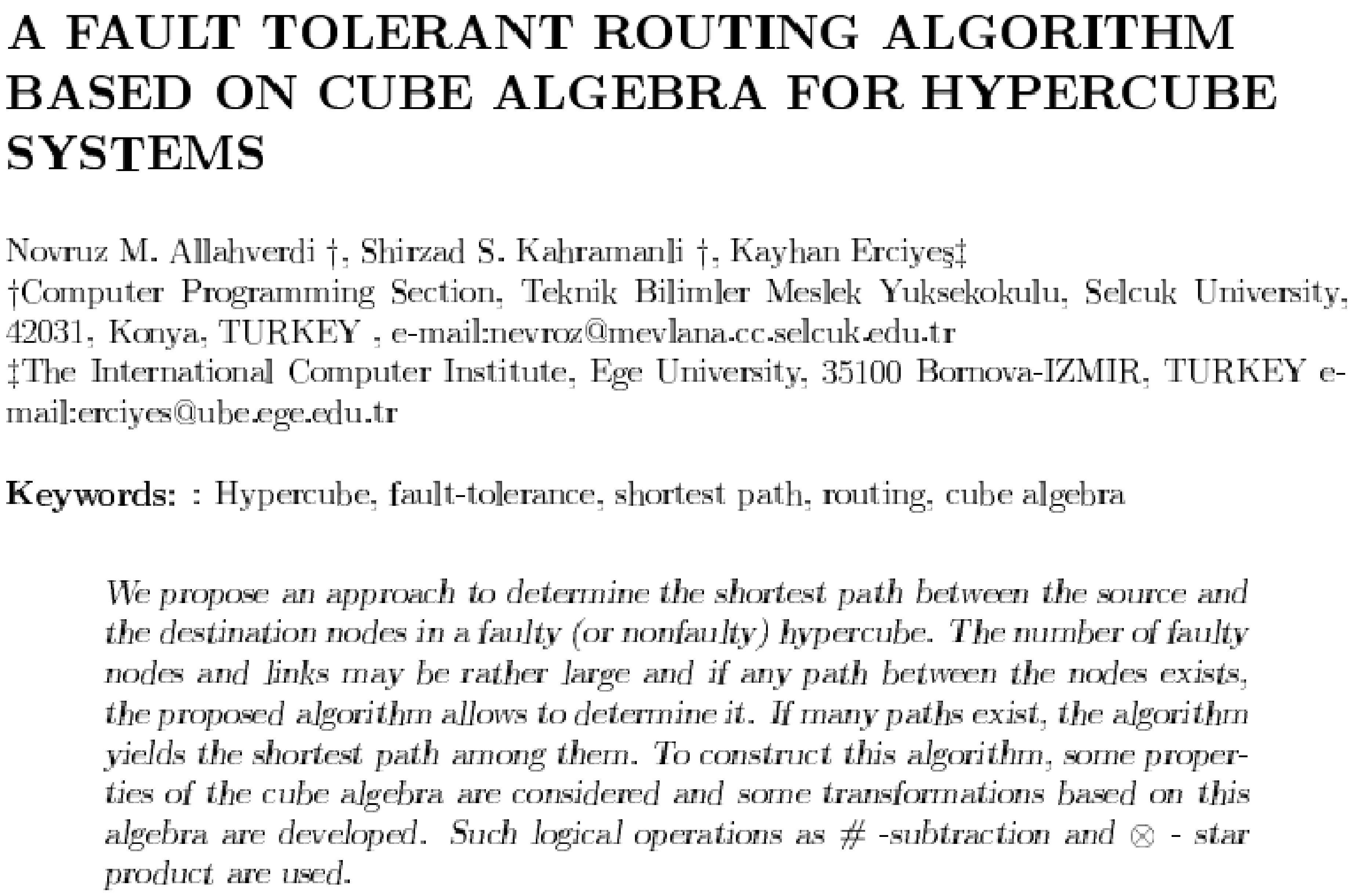
- We compare use of a genetic algorithm and the reactive tabu search for fitness
- optimization. On a 2-bit adder design problem, the reactive tabu search performs significantly better for a similar execution time.
- 1 Introduction
- The process of designing and implementing an ASIC
- is typically a long and expensive one. Field Pro-
- grammable Gate Arrays (FPGAs) are available as an
- alternative to reduce the concept-to-product time and
- the cost of making modications. Recent FPGAs have
- It is this unrestricted model that interests us. Part
- of the advantage of EH is the removal of conventional
- digital design constraints. We do not wish to arbi-
- trarily add new ones by imposing our own structure
- on the hardware device. Unfortunately this presents
- problems for a genetic representation of the model.
- Without some such structure, genetic operators have
- little meaning. We are therefore interested in other
- optimization techniques for maximizing a tness crite-
- rion.
- Taking a cue from Perkowski et al. [10], we refer to this
- hardware adaptation as learning hardware, withEHas
- a special case. We compare a genetic algorithm with
- a non-genetic discrete optimization algorithm (the re-
- active tabu search) on adaptive arithmetic circuit de-
- sign. Section 2 introduces the problem and the two
- optimization algorithms. The physical system imple-
- mentation is described in section 3, and experimental
- results are given in section 4.

1.3 Hardware Learning

In order to improve expected generalization, it is desirable to obtain as many examples as possible. With a very large data set, however, carrying out a learning algorithm may take a very long time. A major issue in learning from examples is the computation time required to carry out a learning algorithm. One direct way to improve the speed of learning is by moving from software to hardware.

For some time, there have been hardware implementations of learning systems, and in particular neural network models [Mead, 1989] [Ramacher and Ruckert, 1991]. More recently, a different approach to hardware machine learning has arisen. Instead of designing a hardware representation of an established learning system, a learning algorithm can be designed for use with existing reconfigurable hardware devices. Most of the results along these lines have involved genetic or evolutionary algorithms, and the field is generally known as *evolvable hardware* (EH). Some success has been shown in evolving analog and mixed-signal circuits [Thompson, 1998] [Bennett *et al.*, 1999], digital filters [Miller, 1999], control circuitry [Keymeulen *et al.*, 1998] and a variety of components for practical applications [Higuchi *et al.*, 1999]. Like Perkowski et al. [Perkowski *et al.*, 1999], we refer to this hardware adaptation as *learning hardware*, with EH as a special case.

Cube Algebra in software



A FAULT TOLERANT ROUTING ALGORITHM BASED ON CUBE ALGEBRA FOR HYPERCUBE SYSTEMS

Novruz M. Allahverdi †, Shirzad S. Kahramanli †, Kayhan Erciyeş‡
†Computer Programming Section, Teknik Bilimler Meslek Yuksekokulu, Selcuk University, 42031, Konya, TURKEY , e-mail:nevroz@mevlana.cc.selcuk.edu.tr
‡The International Computer Institute, Ege University, 35100 Bornova-IZMIR, TURKEY e-mail:erciyes@ube.ege.edu.tr

Keywords: : Hypercube, fault-tolerance, shortest path, routing, cube algebra

We propose an approach to determine the shortest path between the source and the destination nodes in a faulty (or nonfaulty) hypercube. The number of faulty nodes and links may be rather large and if any path between the nodes exists, the proposed algorithm allows to determine it. If many paths exist, the algorithm yields the shortest path among them. To construct this algorithm, some properties of the cube algebra are considered and some transformations based on this algebra are developed. Such logical operations as # -subtraction and $\otimes$ - star product are used.

Cube algebra in hardware accelerator

Mapping switch-level simulation onto gate-level hardware accelerators

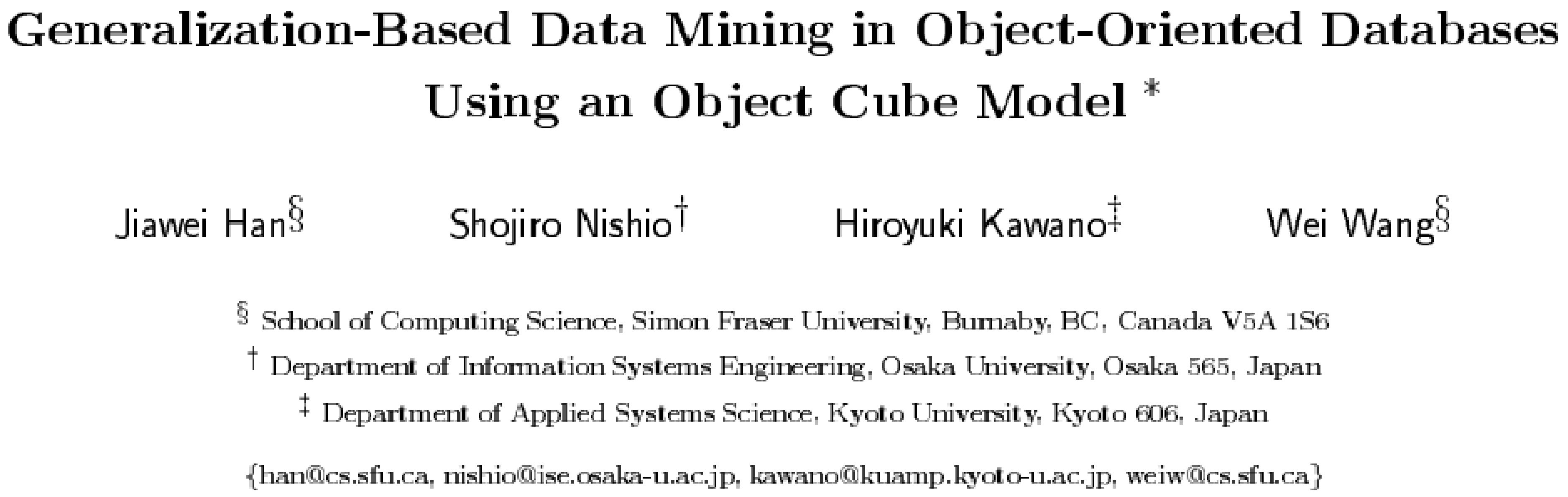
Alok Jain

Dept of ECE
Carnegie Mel l on
Pittsburgh, PA15213

Randal E. Bryant

School of Computer Sci ence
Car negi e Mel l on
Pittsburgh, PA15213

Generalized Cube Calculus for Data Bases and Data Mining



Generalization-Based Data Mining in Object-Oriented Databases Using an Object Cube Model *

Jiawei Han[§] Shojiro Nishio[†] Hiroyuki Kawano[‡] Wei Wang[§]

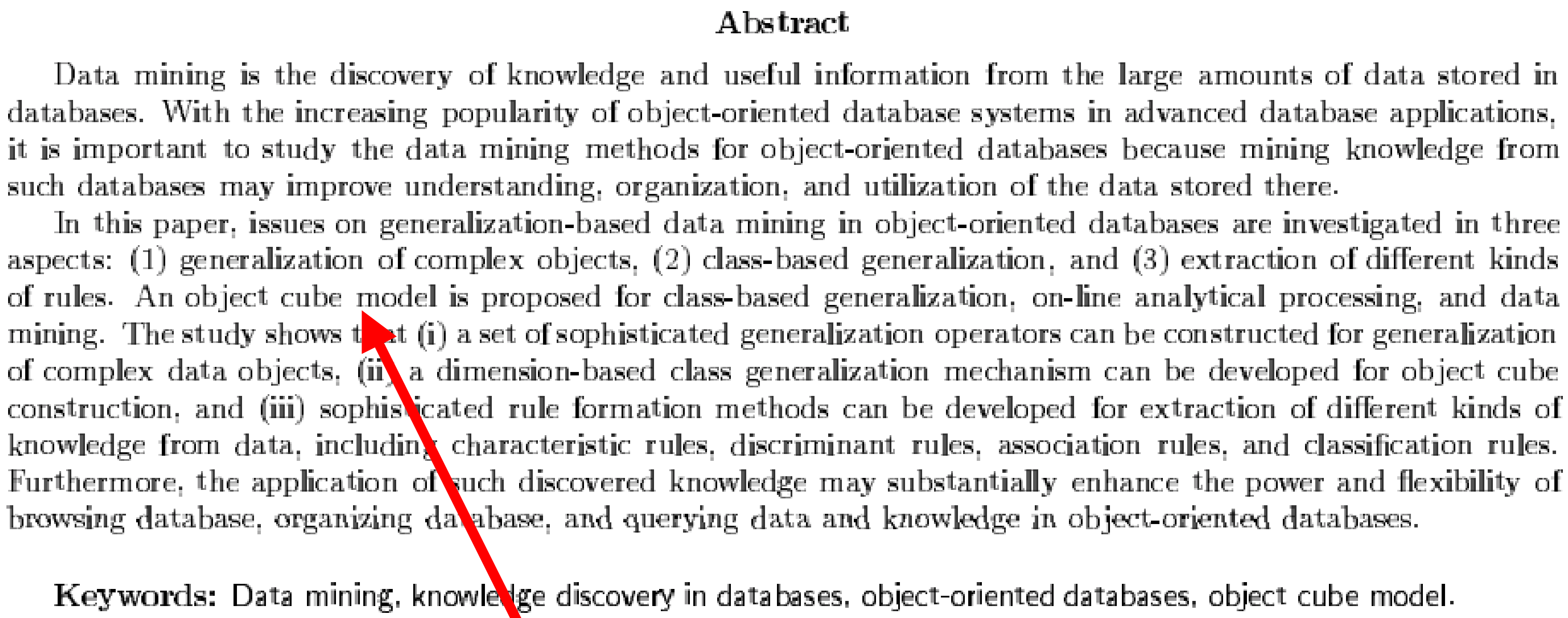
[§] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6

[†] Department of Information Systems Engineering, Osaka University, Osaka 565, Japan

[‡] Department of Applied Systems Science, Kyoto University, Kyoto 606, Japan

{han@cs.sfu.ca, nishio@ise.osaka-u.ac.jp, kawano@kuamp.kyoto-u.ac.jp, weiw@cs.sfu.ca}

Data mining



Abstract

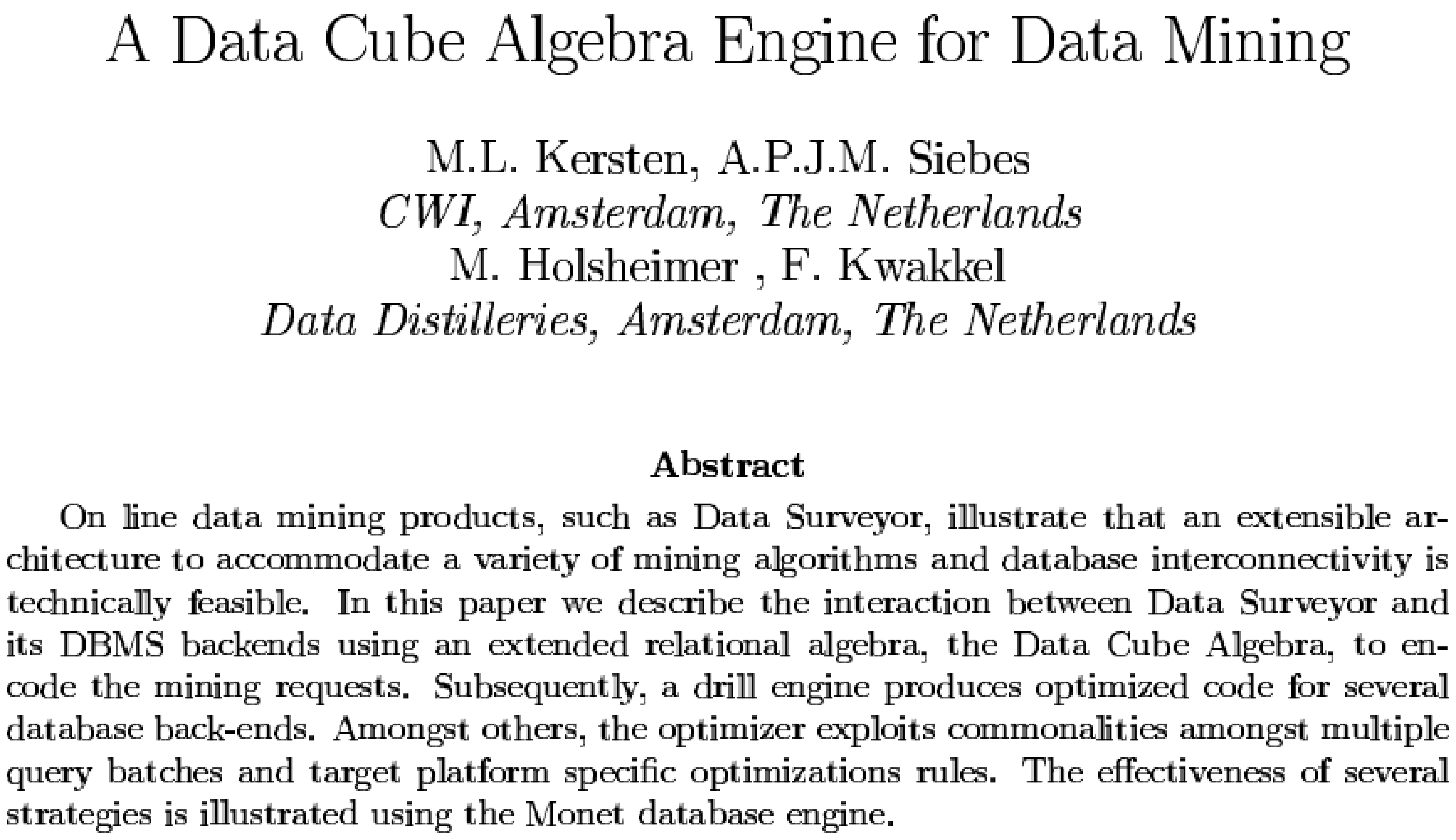
Data mining is the discovery of knowledge and useful information from the large amounts of data stored in databases. With the increasing popularity of object-oriented database systems in advanced database applications, it is important to study the data mining methods for object-oriented databases because mining knowledge from such databases may improve understanding, organization, and utilization of the data stored there.

In this paper, issues on generalization-based data mining in object-oriented databases are investigated in three aspects: (1) generalization of complex objects, (2) class-based generalization, and (3) extraction of different kinds of rules. An object cube model is proposed for class-based generalization, on-line analytical processing, and data mining. The study shows that (i) a set of sophisticated generalization operators can be constructed for generalization of complex data objects, (ii) a dimension-based class generalization mechanism can be developed for object cube construction, and (iii) sophisticated rule formation methods can be developed for extraction of different kinds of knowledge from data, including characteristic rules, discriminant rules, association rules, and classification rules. Furthermore, the application of such discovered knowledge may substantially enhance the power and flexibility of browsing database, organizing database, and querying data and knowledge in object-oriented databases.

Keywords: Data mining, knowledge discovery in databases, object-oriented databases, object cube model.

Cubical model but still for software applications

Cube Calculus for Data Mining



A Data Cube Algebra Engine for Data Mining

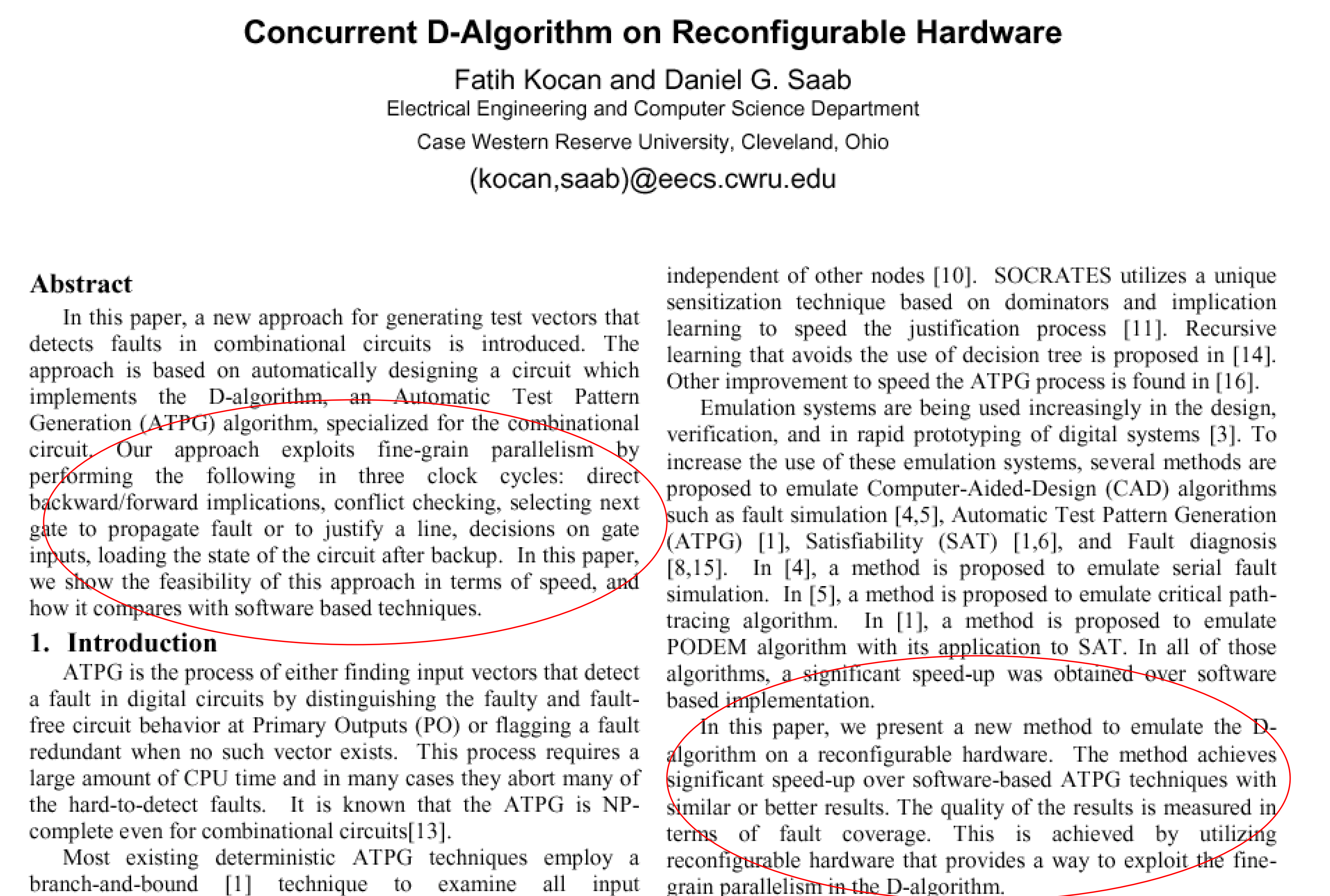
M.L. Kersten, A.P.J.M. Siebes
CWI, Amsterdam, The Netherlands
M. Holsheimer , F. Kwakkel
Data Distilleries, Amsterdam, The Netherlands

Abstract

On line data mining products, such as Data Surveyor, illustrate that an extensible architecture to accommodate a variety of mining algorithms and database interconnectivity is technically feasible. In this paper we describe the interaction between Data Surveyor and its DBMS backends using an extended relational algebra, the Data Cube Algebra, to encode the mining requests. Subsequently, a drill engine produces optimized code for several database back-ends. Amongst others, the optimizer exploits commonalities amongst multiple query batches and target platform specific optimizations rules. The effectiveness of several strategies is illustrated using the Monet database engine.

This is still software engine, not hardware….

D algorithm reconfigurable hardware



Concurrent D-Algorithm on Reconfigurable Hardware

Fatih Kocan and Daniel G. Saab
Electrical Engineering and Computer Science Department
Case Western Reserve University, Cleveland, Ohio
(kocan,saab)@eecs.cwru.edu

Abstract

In this paper, a new approach for generating test vectors that detects faults in combinational circuits is introduced. The approach is based on automatically designing a circuit which implements the D-algorithm, an Automatic Test Pattern Generation (ATPG) algorithm, specialized for the combinational circuit. Our approach exploits fine-grain parallelism by performing the following in three clock cycles: direct backward/forward implications, conflict checking, selecting next gate to propagate fault or to justify a line, decisions on gate inputs, loading the state of the circuit after backup. In this paper, we show the feasibility of this approach in terms of speed, and how it compares with software based techniques.

1. Introduction

ATPG is the process of either finding input vectors that detect a fault in digital circuits by distinguishing the faulty and fault-free circuit behavior at Primary Outputs (PO) or flagging a fault redundant when no such vector exists. This process requires a large amount of CPU time and in many cases they abort many of the hard-to-detect faults. It is known that the ATPG is NP-complete even for combinational circuits[13].

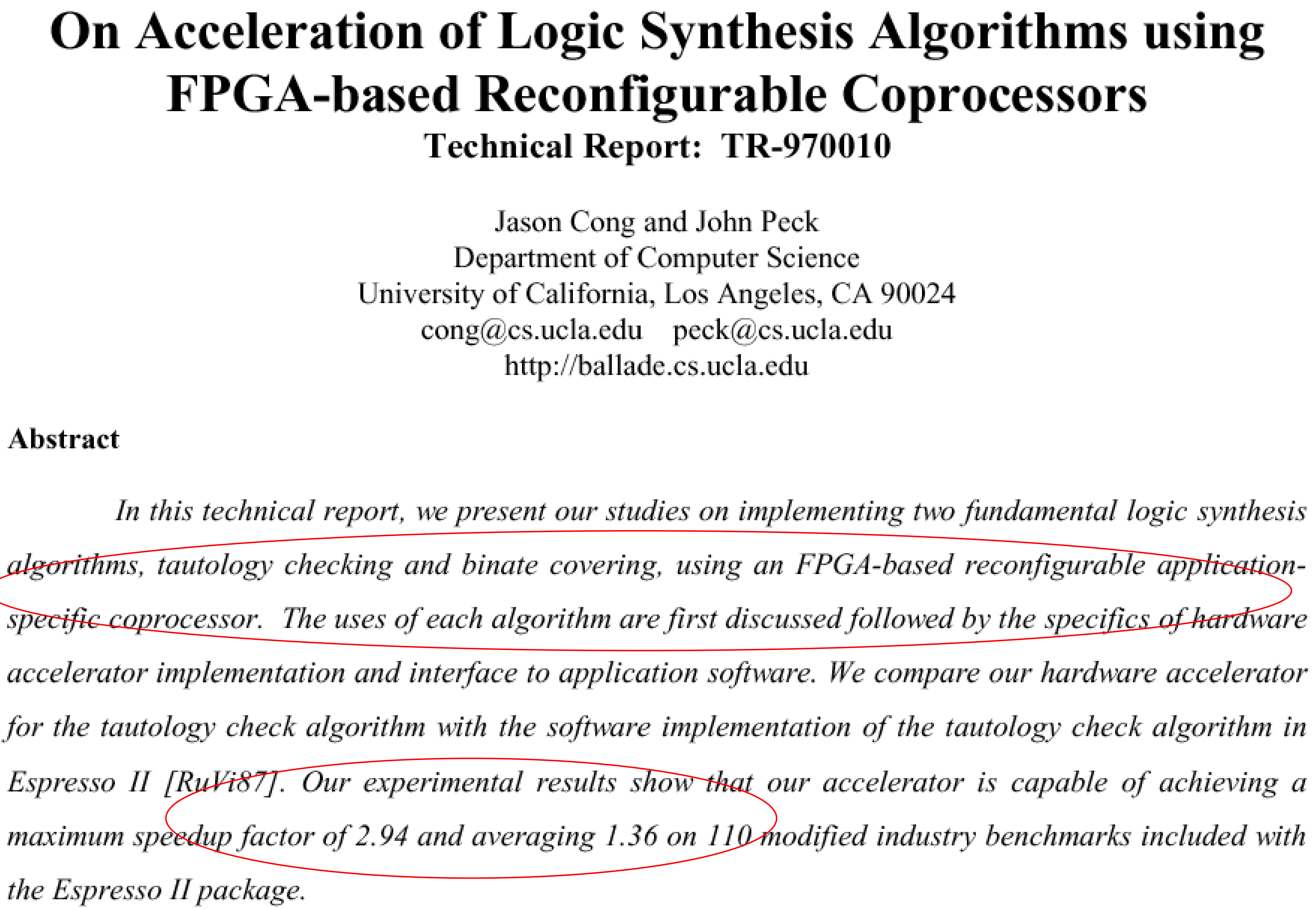
Most existing deterministic ATPG techniques employ a branch-and-bound [1] technique to examine all input

independent of other nodes [10]. SOCRATES utilizes a unique sensitization technique based on dominators and implication learning to speed the justification process [11]. Recursive learning that avoids the use of decision tree is proposed in [14]. Other improvement to speed the ATPG process is found in [16].

Emulation systems are being used increasingly in the design, verification, and in rapid prototyping of digital systems [3]. To increase the use of these emulation systems, several methods are proposed to emulate Computer-Aided-Design (CAD) algorithms such as fault simulation [4,5], Automatic Test Pattern Generation (ATPG) [1], Satisfiability (SAT) [1,6], and Fault diagnosis [8,15]. In [4], a method is proposed to emulate serial fault simulation. In [5], a method is proposed to emulate critical path-tracing algorithm. In [1], a method is proposed to emulate PODEM algorithm with its application to SAT. In all of those algorithms, a significant speed-up was obtained over software based implementation.

In this paper, we present a new method to emulate the D-algorithm on a reconfigurable hardware. The method achieves significant speed-up over software-based ATPG techniques with similar or better results. The quality of the results is measured in terms of fault coverage. This is achieved by utilizing reconfigurable hardware that provides a way to exploit the fine-grain parallelism in the D-algorithm.

Tautology and Binate Covering



On Acceleration of Logic Synthesis Algorithms using FPGA-based Reconfigurable Coprocessors

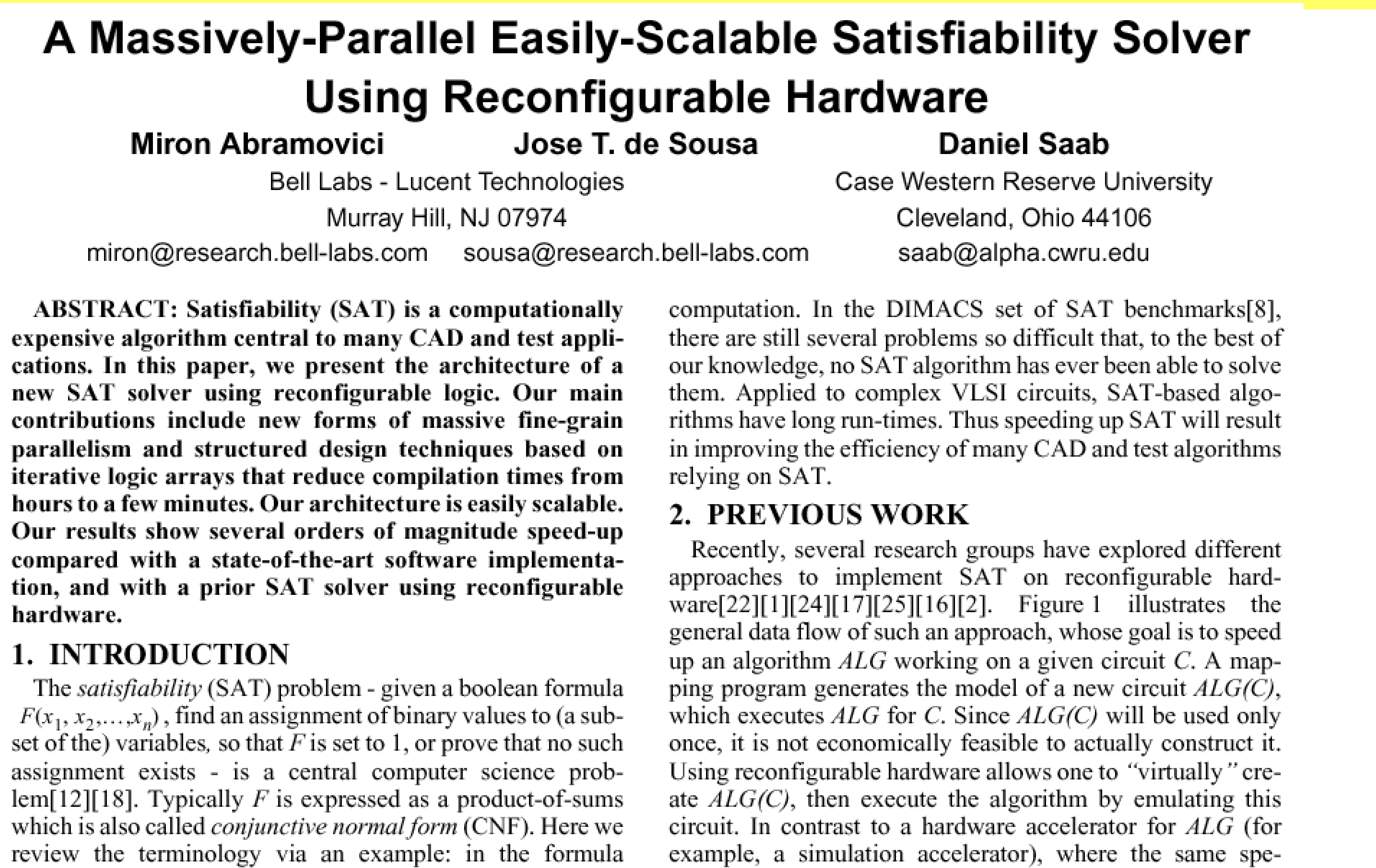
Technical Report: TR-970010

Jason Cong and John Peck
Department of Computer Science
University of California, Los Angeles, CA 90024
cong@cs.ucla.edu peck@cs.ucla.edu
http://ballade.cs.ucla.edu

Abstract

In this technical report, we present our studies on implementing two fundamental logic synthesis algorithms, tautology checking and binate covering, using an FPGA-based reconfigurable application-specific coprocessor. The uses of each algorithm are first discussed followed by the specifics of hardware accelerator implementation and interface to application software. We compare our hardware accelerator for the tautology check algorithm with the software implementation of the tautology check algorithm in Espresso II [RuVi87]. Our experimental results show that our accelerator is capable of achieving a maximum speedup factor of 2.94 and averaging 1.36 on 110 modified industry benchmarks included with the Espresso II package.

Satisfiability Reconfigurable



A Massively-Parallel Easily-Scalable Satisfiability Solver Using Reconfigurable Hardware

Miron Abramovici **Jose T. de Sousa** **Daniel Saab**
Bell Labs - Lucent Technologies, Murray Hill, NJ 07974
Case Western Reserve University, Cleveland, Ohio 44106
miron@research.bell-labs.com sousa@research.bell-labs.com saab@alpha.cwru.edu

ABSTRACT: Satisfiability (SAT) is a computationally expensive algorithm central to many CAD and test applications. In this paper, we present the architecture of a new SAT solver using reconfigurable logic. Our main contributions include new forms of massive fine-grain parallelism and structured design techniques based on iterative logic arrays that reduce compilation times from hours to a few minutes. Our architecture is easily scalable. Our results show several orders of magnitude speed-up compared with a state-of-the-art software implementation, and with a prior SAT solver using reconfigurable hardware.

1. INTRODUCTION

The *satisfiability* (SAT) problem - given a boolean formula $F(x_1, x_2,\ldots,x_n)$, find an assignment of binary values to (a subset of the) variables, so that F is set to 1, or prove that no such assignment exists - is a central computer science problem[12][18]. Typically F is expressed as a product-of-sums which is also called *conjunctive normal form* (CNF). Here we review the terminology via an example: in the formula

computation. In the DIMACS set of SAT benchmarks[8], there are still several problems so difficult that, to the best of our knowledge, no SAT algorithm has ever been able to solve them. Applied to complex VLSI circuits, SAT-based algorithms have long run-times. Thus speeding up SAT will result in improving the efficiency of many CAD and test algorithms relying on SAT.

2. PREVIOUS WORK

Recently, several research groups have explored different approaches to implement SAT on reconfigurable hardware[22][1][24][17][25][16][2]. Figure 1 illustrates the general data flow of such an approach, whose goal is to speed up an algorithm *ALG* working on a given circuit *C*. A mapping program generates the model of a new circuit *ALG(C)*, which executes *ALG* for *C*. Since *ALG(C)* will be used only once, it is not economically feasible to actually construct it. Using reconfigurable hardware allows one to "virtually" create *ALG(C)*, then execute the algorithm by emulating this circuit. In contrast to a hardware accelerator for *ALG* (for example, a simulation accelerator), where the same spe-

MV cubes used in MVSIS of Brayton

MVSIS group

Minxi Gao

Yinghua Li

Jie-Hong Jiang

Yunjian Jiang

Alan Mishchenko, (PSU, Portland OR)

Subarnareka Sinha

Tiziano Villa

Robert K. Brayton

Publications on MVSIS

EVOLVABLE HARDWARE OR LEARNING HARDWARE?

- Evolvable Hardware is Genetic Algorithm (GA) plus reconfigurable hardware.
- One may ask: *"Why Genetic Algorithm"?*
- We question the usefulness of GA as a sole learning method to reconfigure binary FPGAs.

EVOLVABLE HARDWARE OR LEARNING HARDWARE?

- We propose the "Learning Hardware" approach.
- Creating a sequential/combinational network based on feedback from the environment (for instance, positive and negative examples from the trainer), and realizing this network in an array of Field Programmable Gate Arrays (FPGAs).

Symbolic Learning from binary and MVL data

- DNF minimization
- Problems reducible to exorlink (ESOP, etc)
- Factorization, problems reducible to covering and binate covering
- Problems reducible to graph coloring
- Problems reducible to maximum clique (robotics, image processing)
- Constraints solving.
- Finite State Machine (FSM) minimization.
- FSM assignment and encoding
- Functional decomposition of multi-valued logic functions and relations

LEARNING ON A HIGHER LEVEL

- Learning on the level of constraints acquisition and functional decomposition rather than on the low level of programming binary switches.
- Occam's Razor learning that allows for generalization and discovery.
- Fast operations on complex logic expressions and solving NP-complete problems such as satisfiability .
- Algorithms realized in hardware to obtain the necessary speed-ups.
- Fast prototyping tool, the DEC-PERLE-1 board based on an array of Xilinx FPGAs.
- Now we have better boards - Dr. Greenwood

SOFT COMPUTING AND MACHINE LEARNING VERSUS HARDWARE DESIGN

- Artificial Neural Nets (ANNs), Cellular Neural Nets (CNN), Fuzzy Logic, Rough Sets, Genetic Algorithms (GA), Genetic and Evolutionary Programming, Artificial Life, solving problems by analogy to Nature, decision making, knowledge acquisition, new approaches to intelligent robotics.
- Learning, adapting, modifying, evolving or emerging.
- Mixed approaches.
- The computer is taught on examples rather than completely programmed (instructed) what to do.
- Machine Learning becomes a new and most general system design paradigm unifying these previously disconnected research areas.
- It starts to become a new hardware construction paradigm as well.

EVOLVABLE HARDWARE

- **DeGaris -** Evolvable Hardware is realization of genetic algorithm (GA) in reconfigurable hardware.
- Brain Builder CBM (DeGaris), ROBOKONEKO.
- Neural Nets PLUS Genetic Algorithm.
- The Genetic Algorithm is a simple and practically blind mechanism of Nature.
- It is easily realizable in hardware.
- Although it is relatively easy to do crossover and mutation in hardware, the fitness function evaluation is difficult.

UNIVERSAL LOGIC MACHINE

- Started in Poland, 1977. Logic Design Machine. (TTL logic model):
- Satisfiability, Petrick Function (ICCAD'85).
- Tsutomu Sasao, 1985: Tautology Engine in EPLDs (ICCD'85).
- Cube Calculus Machine, since 1990. (realization in FPGAs). (Sendai'92).
- Decomposition Machine, since 1997, (DEC-PERLE-1), (Lousanne'98, ICCD'98, Sendai'99).
- Temporal Constraints Machine - new ideas presented here for the rst time (reduce to Satisfiability, Tautology, Decision Functions, and Boolean/Multi-Valued Logic Equations.

LOGIC ALGORITHMS IN HARDWARE

- Logic algorithms draw upon human knowledge.
- Logic algorithms are optimal and mathematically sophisticated.
- Logic algorithms lead to high quality learning results:
  - knowledge generalization,
  - discovery,
  - no overfitting,
  - small learning errors (Ross, Abu Mostafa, DFC, COLT).
- Their software realizations use very complex data structures and controls.
- It is difficult to realize them in hardware.

LEARNING HARDWARE

- Learning understood very broadly, as any mechanism that leads to the improvement of operation.
- Evolution-based learning is thus included in it.
- Combinational or sequential **network is constructed that stores the knowledge acquired in the learning phase.**
- The learned network is next run on old or new data.
- The responses may be correct or erroneous. The network's behavior is then evaluated by some fitness (cost) functions and the learning and running phases are alternating.

WHY TO USE HARDWARE INSTEAD OF SOFTWARE?

- Supervised inductive learning algorithms require fast operations on complex logic expressions and solving some NP-complete problems.
- Satisfiability, Tautology, Solving Boolean Equations, Graph Coloring, Set Covering, Maximum Cliques.
- These algorithms should be realized in hardware to obtain the necessary speed-ups.
- Fast prototyping tool, DEC-PERLE-1 board is based on an array of Xilinx FPGAs.
- We are developing virtual processors that accelerate the design and optimization of decomposed networks of arbitrary logic blocks.

EVOLVING IN HARDWARE VERSUS LEARNING IN HARDWARE

- Soft Computing: Artificial Neural Nets (ANNs), Cellular Neural Nets (CNN), Fuzzy Logic, Rough Sets, Genetic Algorithms (GA), Genetic and Evolutionary Programming, Artificial Life, Solving Problems by Analogy to Nature, decision making, knowledge acquisition, new approaches to intelligent robotics (Brooks).
- Learning, adapting, modifying, evolving or emerging.
- Mixed approaches combine elements of these areas with the goal of solving very complex and poorly defined problems that could not be tackled by previous, analytic models.

EVOLVING IN HARDWARE VERSUS LEARNING IN HARDWARE (cont)

- What is common to all these approaches is that they propose a way of automatic learning by the system.
- The computer is taught on examples rather completely programmed (instructed) what to do.
- Machine Learning (ML) becomes then now a new and most general system design paradigm unifying many previously disconnected research areas.
- ML starts to become a new hardware construction paradigm as well.

EVOLVABLE HARDWARE VERSUS LOGIC METHODS.

- Evolvable Hardware (EHW) (De Garis, Higuchi) is a realization of genetic algorithm (GA) in reconfigurable hardware.
- Our approach of Universal Logic Machine (ICCAD '85, Sendai '92, Jozwiak'98), proposes to build a learning machine based on logic principles .
- Constructive Induction (Michalski) and Rough Set Theory (Pawlak).
- Genetic Algorithm is a very simple and practically blind mechanism of Nature, it can be easily realizable in hardware.
- We do not believe that this mechanism alone cannot produce good results.

EVOLVABLE HARDWARE VERSUS LOGIC METHODS.

- TRADE-OFFS
- The logic algorithms that use previous human knowledge are optimal and mathematically sophisticated. They lead to high quality learning results.
- Their software realizations use so complex data structures and controls that it is very difficult to realize them in hardware.
- Software/hardware realizations may suffer from the consequences of the Amdahl's Law.
- Interesting software-hardware design trade-offs must be resolved to realize optimally the learning algorithms based on logic.

LEARNING HARDWARE

- "Learning Hardware" is any mechanism that leads to the improvement of operation, evolution-based learning is thus included.
- The process of learning some kind of network. It stores the knowledge acquired in the learning phase (the network can become equivalent to a state machine or fuzzy automaton by adding some discrete or continuous memory elements).
- The learned network is next run (executed, evaluated, etc.) for old or new data given to it, thus producing its responses - expected behaviors(decisions, controls) in unfamiliar situations (new data sets).
- The responses may be correct or erroneous, the network's behavior is then evaluated by some fitness (cost) functions and the learning and running phases are interspersed.

TWO PHASES OF LEARNING IN HARDWARE

- **The phase of learning** , which is, constructing and tuning the network.
- **The phase of acting** . Using knowledge, running the network for data sets.
- The first stage could be compared to the entire process of conceptualizing, designing, and optimizing a computer, and the second stage to using this computer to perform calculations.
- You cannot redesign standard computer hardware, however, when it cannot solve the problem correctly.
- **The Learning Hardware will redesign itself automatically using new learning examples given to it**.

Logic rather than evolutionary methods for learning

- Michie makes distinction between black-box and knowledge-oriented concept learning systems by introducing concepts of weak and strong criteria.
- The system satisfies a weak criterium if it uses sample data to generate an updated basis for improved performance on subsequent data.
- A strong criterion is satisfied if the system communicates concepts learned in symbolic form.
- ANNs satisfy only the weak criterium while our approach satisfies the strong criterium. **Our approach operates on higher and more natural symbolic representation levels.**

Logic rather than evolutionary methods for learning. II

- The built-in mathematical optimization techniques (such as graph coloring or satisfiability) support the Occam's Razor Principle.
- Solutions are provably good in the sense of Computational Learning Theory (COLT).

Importance of Functional Decomposition

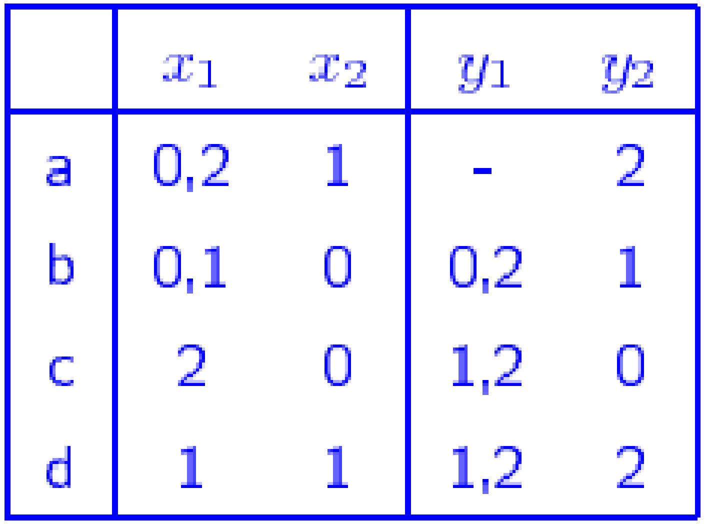
- Functional Decomposition is used in many applications: FPGA mapping, custom VLSI design, regular arrays, Machine Learning, Data Mining and Knowledge Discovery in Data Bases (KDD).
- Exact decomposition programs are slow.
- Approximate programs may give inferior quality solutions.
- How to create a decomposer that will be both effective and efficient ?.
- ANSWER: Software/Hardware Co-Design.

Learning in real time!

We do not like Genetic Algorithms. Any Discussions?

- In our experience, especially poor results on logic approaches are obtained using the genetic algorithms.
- The same was true based on literature.
- In our approach we want to make use of this accumulated human experience, rather than to "reinvent" algorithms using GA.

The Input Language to Represent the Learning Data



	x_1	x_2	y_1	y_2
a	0,2	1	-	2
b	0,1	0	0,2	1
c	2	0	1,2	0
d	1	1	1,2	2

- Table 1: Multi-Valued multi-output (combinational) relation in tabular form.

DATA MINING BY CONSTRUCTIVE INDUCTION MACHINES

- "Learning Hardware" approach involves creating a computational network based on feedback from the environment and realizing this network in an array of Field Programmable Gate Arrays (FPGAs).
- Feedback, is for instance by positive and negative examples from the trainer.
- Environment can be the trainer.
- Computational networks can be built based on incremental supervised learning (Neural Net training) or global construction (Decision Tree design).
- Here we advocate the approach to Learning Hardware based on Constructive Induction methods of Machine Learning (ML) using multi-valued functions.
- This is contrasted with the Evolvable Hardware (EHW) approach in which learning/evolution is based on the genetic algorithm only .

Project “Logic Machine”

Cube Calculus Machine

Decomposition Machine

Satisfiability-ESOP minimization Machine

Rough Set Machine

All these projects require logic design

Systolic or pipelined or cellular machines

FPGA realization (Xilinx, Altera,Cypress)

VHDL or high-level tools (Summit or Renoir)

Project 1: Universal Logic Machine

- ➔Combinational problems reduced to simple combinational problems such as graph coloring, set covering, binate covering, clique partitioning, satisfiability or multi-valued relation/function manipulation

- ➔ Cube Calculus Machine (CCM) operates on multiple-valued cubes (terms of MV literals).
- ➔First variant uses two FPGA 3090 chips and second the DEC-PERLE-1 board with 23 chips
- ➔General Special-Purpose computer for Cube Calculus

Universal Logic Machine

- ➔ Synthesis and Decision problems reduced to NP-hard combinational problems
- ➔ Combinational problems reduced to simple combinational problems such as graph coloring, set covering, binate covering, clique partitioning, satisfiability or multi-valued relation/function manipulation
- ➔ Cube Calculus Machine (CCM) operates on multiple-valued cubes (terms of MV literals).
- ➔ First variant uses two FPGA 3090 chips and second the DEC-PERLE-1 board with 23 chips
- ➔ General Special-Purpose computer for Cube Calculus

Universal Logic Machine

- ➔ Phase of learning (construction, synthesis)
- ➔ Phase of acting (function evaluation, state machine operation)
- ➔ You cannot redesign standard computer hardware when it cannot solve the problem correctly.
- ➔ The Learning Hardware redesigns itself using new learning examples given to it
- ➔ Michie makes distinction between black-box and knowledge-oriented learning systems
- ➔ Concepts of "weak" and "strong" criteria
- ➔ "The system satisfies a weak criterium if it uses data to generate an updated basis for improved performance on subsequent data" (Neural, Genetic)

Universal Logic Machine

- ➔ A strong criterium is satisfied if the system communicates in symbolic form concepts that it learned
  - ➔ Constructive Induction (Michalski), Rough Set Theory (Pawlak), Decision Trees (Quinlan), Decision Diagrams, Disjunctive Normal Forms.
  - ➔ Occam’s Razor Principle
  - ➔ Learning on symbolic level is the first main point of our approach, learning on the level of logic gates is the second
  - ➔ Our approach is based on decomposition of relations and functions and on synthesis of non-deterministic machines from declarative specifications
  - ➔ “Do-not-knows” become “don’t-cares” for logic synthesis

Universal Logic Machine

- ➔ The high quality of decompositional techniques in Machine Learning, Data Mining and Knowledge Discovery areas was demonstrated by several authors; Ross (Wright Labs),Bohanec, Bratko/Zupan, Perkowski/Grygiel,Perkowski/Luba/Sadowska, Jozwiak, Luba, Goldman, Axtel.
- ➔ Small learning errors. Natural problem representation
- ➔ We compared the same problems using several methods: decomposition, decision trees, neural nets, and genetic algorithms
- ➔ Decomposition is clearly the winner but it is slow because the NP-complete problem of graph creation and coloring is repeated very many times.

PLAN OF EVOLVABLE AND LEARNING HARDWARE LECTURES

- **Our hardware** : the **DEC-PERLE-1** board.
  - Programming/designing environment for DEC-PERLE/XILINX.
  - Two different concepts of designing Learning Hardware using the DEC-PERLE-1 board.
- Compare **logic** versus ANN and GA approaches to learning.
- Introduce the concept of **Learning Hardware**
- Methods of **knowledge representation** in the **Universal Logic Machine (ULM):**
  - **variants of Cube Calculus**.
- A general-purpose computer with instructions specialized to operate on logic data: **Cube Calculus Machine**.
  - Variants of cube calculus - arithmetics for combinatorial problems
  - Our approach to Cube Calculus Machine
- A processor for only one application: **Curtis Decomposition Machine**.

We are here

CUBE CALCULUS and other representations

STANDARD BINARY CUBE CALCULUS

- Represents product terms as cubes where the state of each input variable is specified by a symbol:
  - positive (1),
  - negative (0),
  - non-existing (a don't care) (X),
  - or contradictory (epsilon).
- Each of these symbols is encoded in **positional notation** with two bits as follows: 1 = 01, 0 = 10, X = 11, epsilon = 00.
- Positional notation for cube 0X1 is 10-11-01.
- Each position represents a state of the variable by the presence of "one" in it: left bit - value 0, right bit - value 1.
- This encoding presents simple reduction to **set-theoretical** representations

STANDARD BINARY CUBE CALCULUS

- A cube can represent :
  - a product, a sum,
  - a set of symmetry coefficients of a symmetric function,
  - a spectrum of the function,
  - or another piece of data on which some symbol-manipulation (usually set-theoretical) operations are executed.
- Usually the cube corresponds to a product term of literals.
- For instance, assume the following order of binary variables: **age** , **sex** and **color_of_hair.** Assume also that the discretization of variable **age** is:age = 0 for person's **age < 18** and **age = 1** otherwise
- Men are encoded by value 0 of attribute **sex** and women by value 1.
- **color_of_hair** is 0 for black and 1 for blond.
- A blond woman of age 19 is denoted by 110 and a black-hair seven-years old person of unknown sex is described by cube 0X1.
- Cube XXX is the set of all possible people for the selected set of attribute variables and their discretized values.

STANDARD BINARY CUBE CALCULUS (3)

- **Two-dimensional representation** is just a set of cubes where the connecting operator is implicitly understood as:
  - OR for SOP;
  - EXOR for ESOP;
  - concatenation for a spectrum,
  - or other.
- For instance, assuming each cube corresponding to AND operator and the OR being the connecting operator;
  - the list {0X1,110} is the SOP which represents the above mentioned two people (or a set of all people with these properties).
- Multi-valued and integer data can be encoded with binary strings in this representation,
  - so that next all operations are executed in binary (we use this model in the decomposition machine)

STANDARD BINARY CUBE CALCULUS (4)

- For instance, if there were three age categories, **young, medium** and **old**, they can be encoded as values 0, 1 and 2 of the ternary variable *age*, respectively.
- Variable **age** could be next represented in hardware as pair of variables **age_1** and **age_2**, where

  0 = 00, 1 = 01, 2 = 10,

- thus encoding:

young = NOT{**age_1**} NOT{**age_2**},

medium = NOT{**age_1**} {**age_2**},

old = age_1 NOT{age_2}.

MULTI-VALUED CUBE CALCULUS (MVCC)

- **A superset of CC.**
- It represents product terms as cubes where each input variable can have a **subset** of a finite set of all possible values that this variable can take.
- Each element of the set is represented by a single bit, which makes this representation not efficient for large sets of values.
- In the above example we could have for instance a 5-valued variable **age** for five age categories, and a quaternary variable **color_of_hair**
- Each position of a variable corresponds to its possible value.

MULTI-VALUED CUBE CALCULUS (MVCC)

- For instance, 10000-10-0100 describes a 7-year old boy with black hair
- This is an example of a minterm cube, i.e. with single values in each variable.
- 01100-11-1100 describes group **G_1** of people, men and women, that are either in second or in third age category and have either blond or black hair.
- This is an example of a cube that is not a minterm.
- 100000-00-1000 describes a first-category-of-age person with blond hair who has some conflicting information in **sex** attribute, for instance a missing value (this is also how contradictions are signalized during cube calculus calculations).
- The hardware operations in MVCC are done directly on such MV variable cubes so that the separate encoding to binary variables is not necessary.

GENERALIZED MV CUBE CALCULUS

- A superset of MVCC.
  - Each **output variable** can be also a subset of values.
  - Such cubes can be directly used to represent MV relations, as in **Table 2**
- Its operations are more general than MVCC, because more interpretations can be given to cubes
- This calculus has more descriptive power, but the respective hardware processors are much **more complicated.**

SIMPLIFIED BINARY CUBE CALCULUS

- A subset of CC. It operates only on **minterms** .
- It has application in decomposition of functions. **Minterms** can be of different dimensions.
- The hardware is much simplified: operations are only set-theoretical.
- This is **the simpliest virtual machine** realized by us, so larger data can be processed by it because more of a machine can be fit to the **limited FPGA Array resources** of DEC-PERLE-1.

SIMPLIFIED MV CUBE CALCULUS

- **Cubes where for every input variable either**
  - **only a single value of its possible values is selected (which is denoted by a binary code (such as a byte) of a symbol corresponding to this value),**
  - **the variable is missing (which is denoted by a selected symbol, X),**
  - **or the variable is contradictory (another symbol, emptyset).**
- **Used for Rough Sets (Pawlak) and variable-valued logic (Michalski).**
- **For instance, assuming 10 age categories,**
  - **0 = 0 - 10 years,**
  - **1 = 10 - 19 years,**
  - **2 = 20 - 29 years, etc,**
  - **and 3 hair categories: 0 = blond, 1 = black, 2 = red,**
  - **the 7-year old boy with black hair is described as 0-0-1,**
  - **the 18-year old girl with black hair is described as 1-1-1,**
  - **the 28-year old woman with red hair is described as 2-1-2, and**
  - **a set of all people with red hair is X-X-2**

SIMPLIFIED MV CUBE CALCULUS

- There is no way now to describe in one cube people below 19 with red or black hair, which was possible in MVCC or GMVCC.
- This simplification of the language brings however big speedup of algorithms and storage reduction when applied for data with many values of attributes.
- The control of algorithms becomes more complicated, while the data path is simplified.

SPECTRAL REPRESENTATIONS

- Examples: Reed-Muller FPRM and GRM spectra, Walsh spectrum, various orthogonal spectra.
- These representations represent function as a sequence of spectral coefficients or selected coefficient values with their numbers.
- Some spectral representations are useful to represent data for **genetic algorithms**: the sequence of spectral coefficients is a chromosome.
- For instance, in the Fixed-Polarity Reed-Muller (FPRM) canonical AND/EXOR forms for n variables, every variable can have two polarities, 0 and 1.
- Thus there are $\mathbf{2^n}$ different polarities for a function and the GA algorithm has to search for the polarity that has the minimum number of ones in the chromosome.

SPECTRAL REPRESENTATIONS

- This way, every solution is correct, and the fitness function is used only to evaluate the cost of the design (100% correctness of the circuit is in general very difficult to achieve in GA.
- Therefore our approaches to logic synthesis based on GA are to **have a representation that provides you with 100% correctness** and have the GA search only for net minimization.
- This approach involves however a more difficult fitness function to be calculated in hardware than the pure GA or Genetic Programming approaches.
- Similarly, the other AND/EXOR canonical form called the Generalized Reed-Muller form (GRM) has **n** $\mathbf{2^{\{n-1\}}}$ binary coefficients, so there are $\mathbf{2^{\{n\ 2\wedge\{n-1\}}}}$ various GRM forms.

SPECTRAL REPRESENTATIONS

- Because there are more GRM forms, it is more probable to find a shorter form among them than among the FPRM forms.
- But the chromosomes are much longer and the evaluation is more difficult.
- This kind of trade-offs is quite common in spectral representations.
- Spectral methods allow for high degree of parallelism.

ROUGH PARTITIONS AND LABELED ROUGH PARTITIONS

- Rough Partitions (RP) represented as Bit Sets (Luba).
- This representation stores the two-dimensional table column-wise, and not row-wise as MVCC does.
- In r-partition every variable (a column of a table) induces a partition of the set of rows (cubes) to blocks, one block for each value the variable can take (there are two blocks for a binary variable, and **k** blocks for a **k**-valued variable).
- Rough Partitions are a good idea but they don't really form a representation of a function.
- Since the values of a variable are not stored together with partition blocks, the essential information on the function is lost and the original data can not be recovered from it.
- This is kind of an abstraction of a function, useful for instance in various decomposition algorithms.

LABELED ROUGH PARTITIONS

- A generalization of RS which has very interesting properties and allows to find different kind of patterns in data.
- It is useful for decomposition of MV relations and it preserves all information about the relation or function.
  - It can be also made canonical, when created for special cubes.
- Most of its operations are reduced to set-theoretical operations, so hardware realization is relatively easy.
- Relations happen in tables created from real data-base and features from images,for instance, MV relations are benchmarks **hayes**, **flare1**, flare2 from Irvine

LABELED ROUGH PARTITIONS (2)

- An example of application of relation in logic synthesis area is a modulo-3 counter (a non-deterministic state machine is a special case of multiple-valued, multi-output, relation) that counts in sequence **s0 -> s1 -> s2 -> s0** and if the state **s3** happens to be the initial state of the counter, counter should transit to any of the states **s0,s1,s2**, but not to the state **s3** itself.
- Generalized values for input variables are already known from cube calculus but **generalized values for output variables** are a new concept which allows for representation and manipulation of relations in LRP.

Cube Calculus Machines

CUBE CALCULUS MACHINE

•In our design, the Cube Calculus Machine is a coprocessor to the host computer and is realized as a virtual processor in DEC-PERLE-1.

the CCM communicates with the host computer through the input and the output FIFO.

The Iterative Logic Unit (ILU) is realized using a one-dimensional iterative network of combinational modules and cellular automata.

ILU is composed from ITs, each of them processes a single binary variable or two values of a multi-valued variable.

Any even number of variables can be processed, and only size of the board as well as bus limitations are the limits (it is the total of 32 values now, which is at most 16 binary variables, 8 quaternary variables, or 4 8-valued variables, or any mixture of even-valued variables).

The ILU can take the input from register file and memory, and can write output to the register file, the memory, and the output FIFO.

The ILU executes the cube operation under the control of Operation Control Unit **(OCU).**

The Global Control Unit **(GCU) controls all parts of the CCM and let them work together.**

•The machine realizes the set of operations from Table 3.

• The Table shows also their programming information. Each row of Table describes one cube operation.

• Each operation is specified in terms of:
  - **rel** - the elementary relation type between input values,
  - and/or - the global relation type, and the internal state of the elementary cellular automaton - before,active and after .

• The operation name, notation, the output value of rel (partial relation) function in every IT, and_or (relation type), the output values of before , active and after functions are listed from left to right.

•Partial relation rel is an elementary relation on elementary piece of data (pair of bits).

•These set theoretical relations such as inclusion, equality, etc.

•The value of and_or equals to 1 means that the relation type is of AND type; otherwise, the relation type is of OR type.

• This relation is created by composing elementary relations from ITs and variables.

horizontal data-path microprogramming

The Output Values of Bitwise Functions Used in Cube Operations

Operation	Notation	Relation		Output Function		
		rel	*and/or*	*before*	*active*	*after*
crosslink	$A \uplus B$	1110	1	0011	0111	0101
sharp	$A \#_{basic} B$	0010	0	0011	0010	0011
disjoint sharp	$A \#d_{basic} B$	0010	0	0011	0010	0001
consensus	$A *_{basic} B$	1111	1	0001	0111	0001
intersection	$A \cap B$	–	–	0001	–	–
super cube	$A \cup B$	–	–	0111	–	–
prime	$A\,'\,B$	0001	0	0011	0111	–
cofactor	$A \|_{basic} B$	1011	1	0001	1111	–

horizontal data-path microprogramming

•The machine is microprogrammable both in its OCU control unit part (by use of CCM Assembly Language) and in Data Path, as achieved by ILU operations programmability.

For instance, each operation is described by the binary pattern corresponding to it in the respective row of Table 3.

By creating other binary patterns in the fields of Table 3, new operations can be programmed to be executed by ILU.

As the reader can appreciate, there are very many such combinations, and thus CCM micro-operations.

horizontal data-path microprogramming

- **We call this** horizontal data-path microprogramming **.**

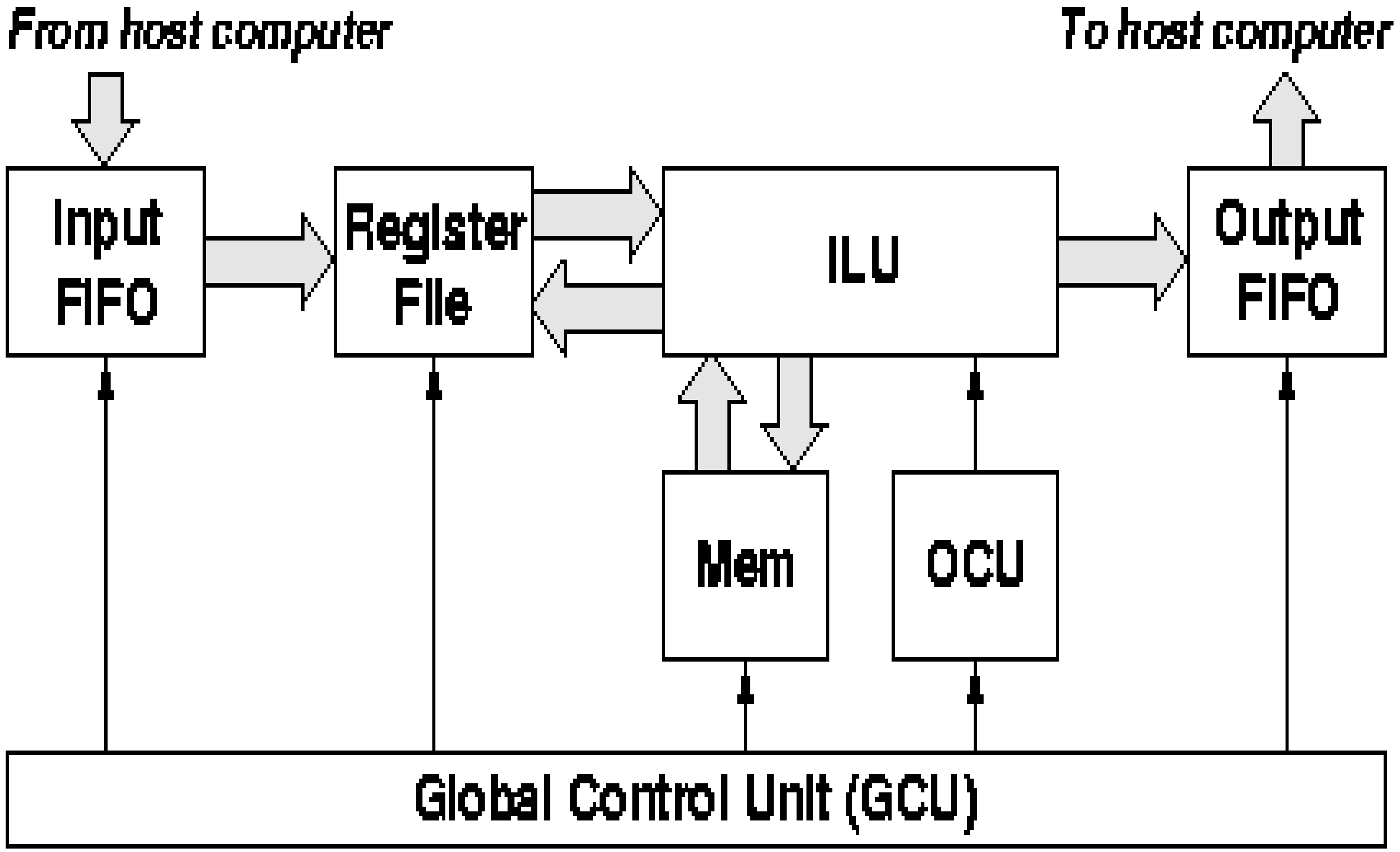
Higher order CCM operations are created by sequencing low-level operations.

This is called vertical control microprogramming **and is executed by OCU (within ILU) and GCU (for operations with memories and I/O).**

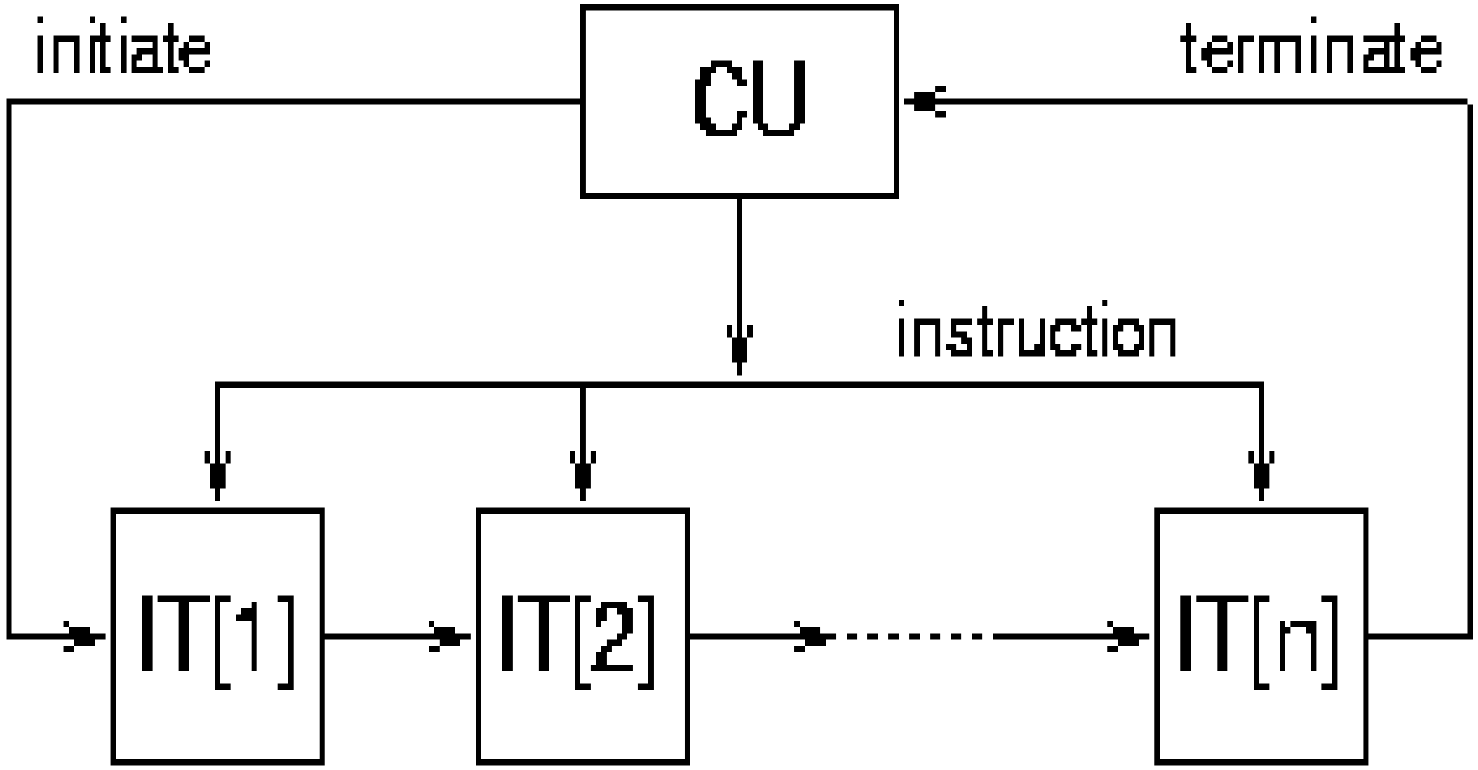
Thus, the user has many ways to (micro) program sequences of elementary instructions.

This is done in CCM Assembly language.

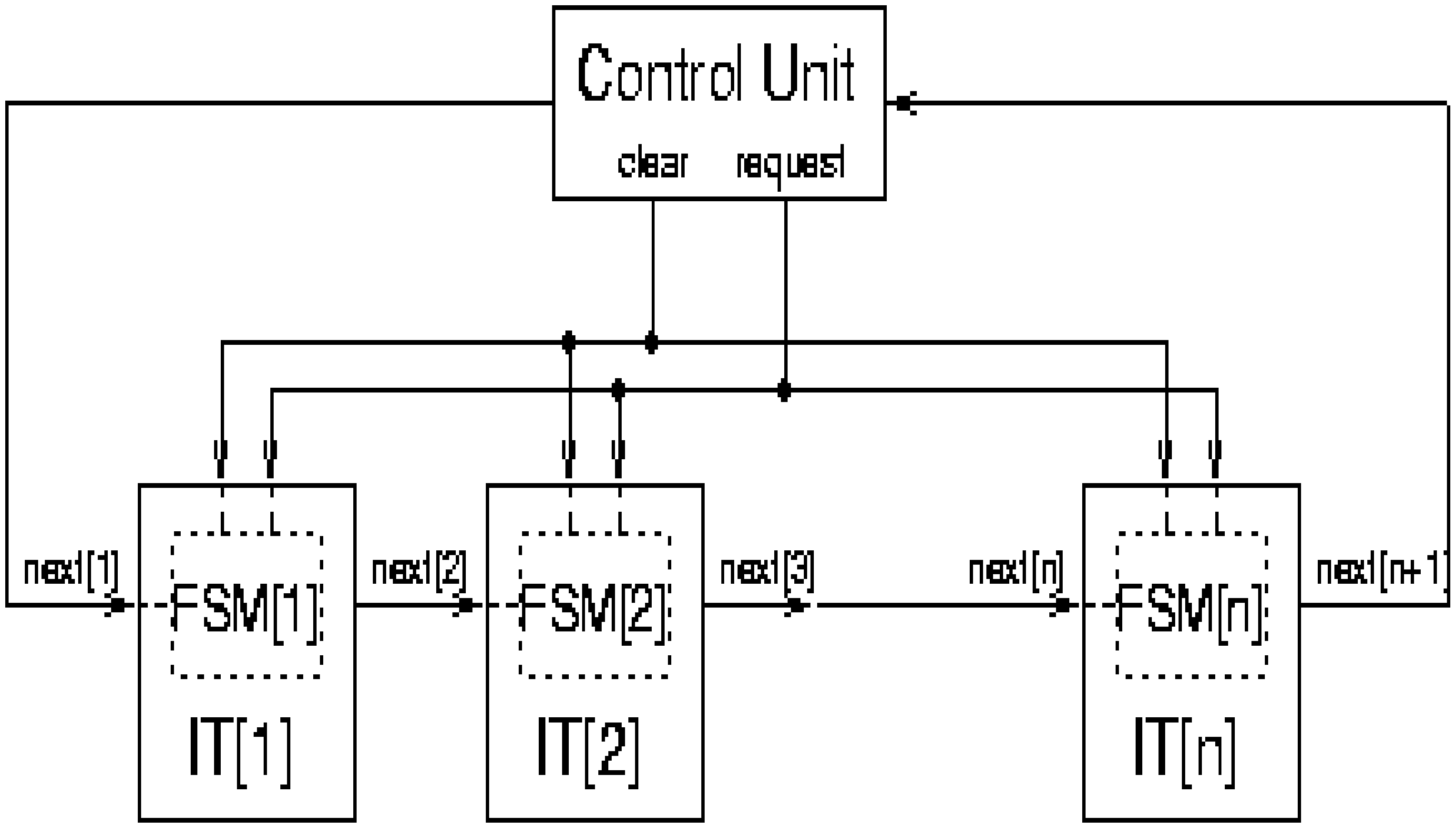
Simplified idea of the Cube Calculus Machine



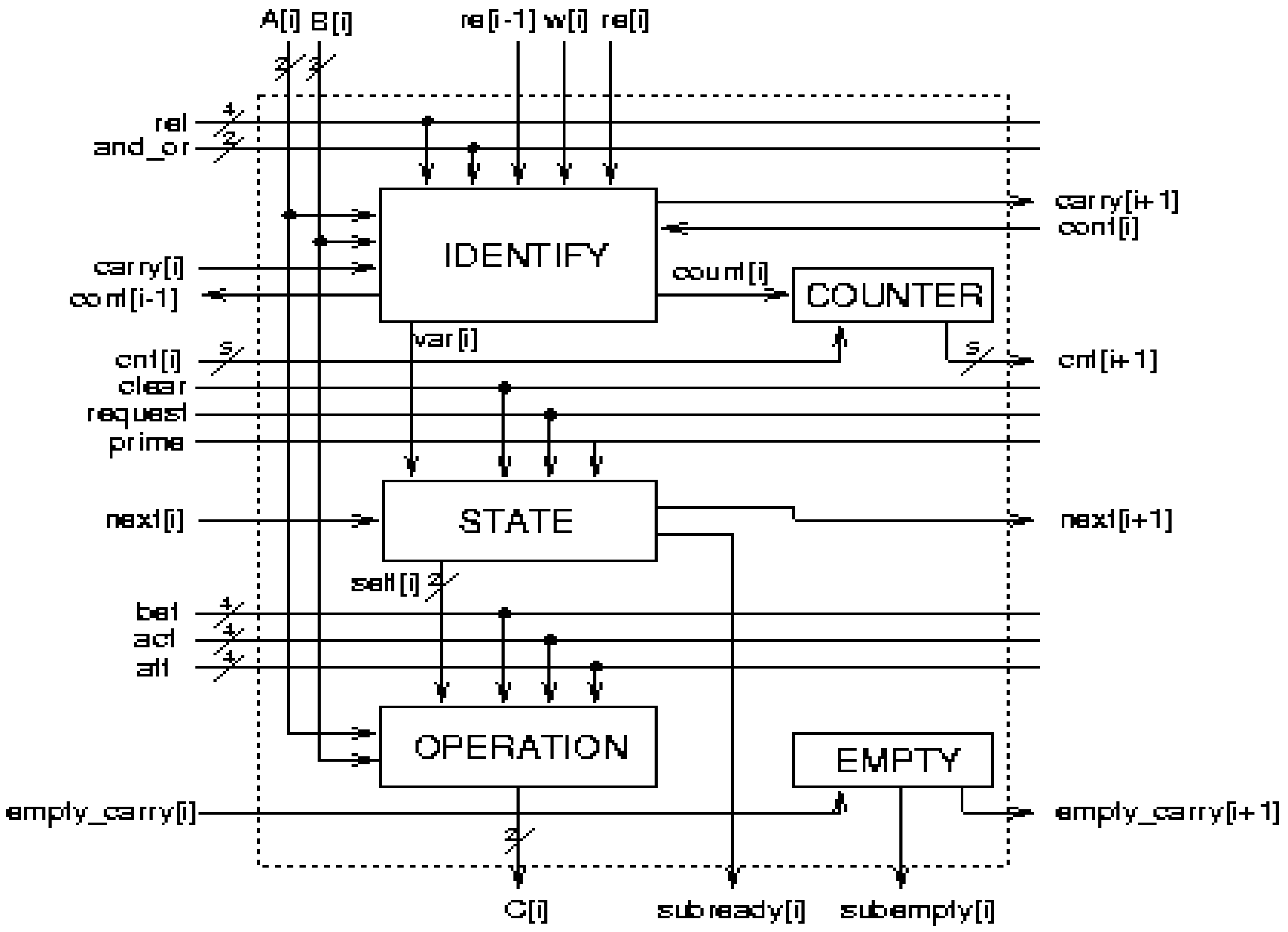
CCM as a SIMD machine



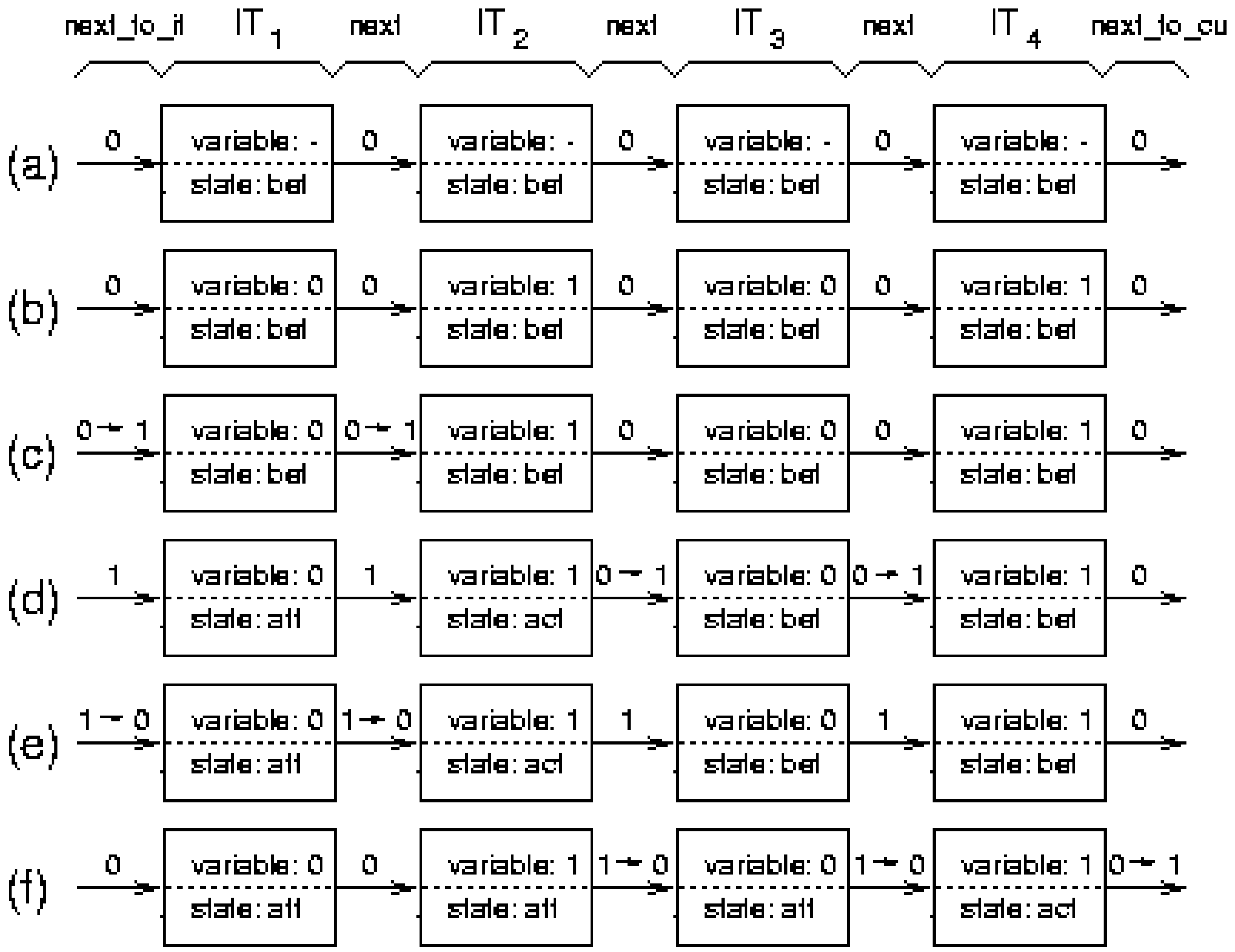
CCM as a Programmable Cellular Automaton



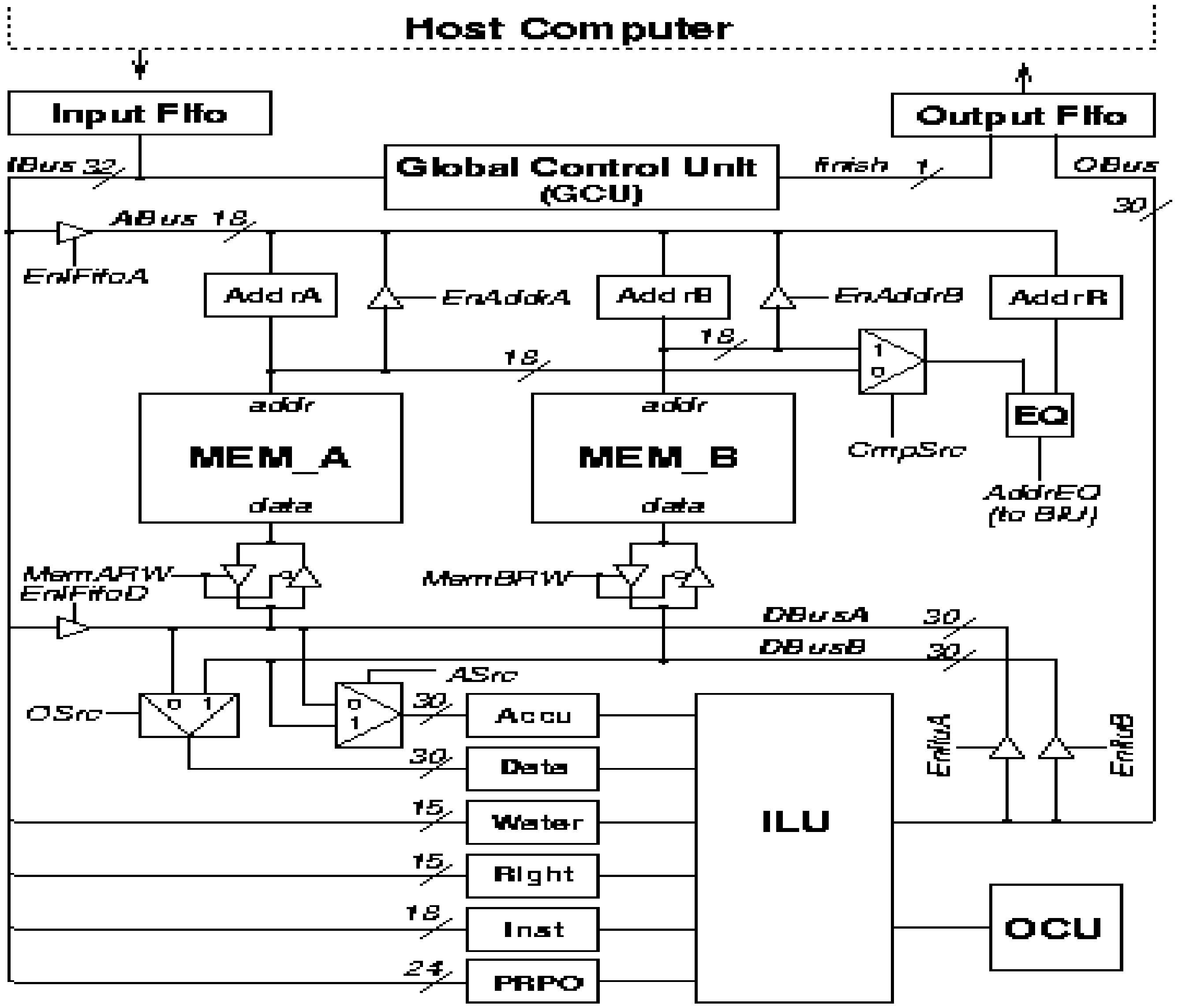
Details of the single cell of Iterative Logic Unit of CCM



Propagation of information in Cellular structure



General Architecture of CCM, new version



Evaluation of some previous Cube Calculus Machines

Evaluation.

- For comparing the performance of the CCM and that of the software approach, a program to execute the disjoint sharp operation on two arrays of cubes was created using C language.
- Then this program and the CCM are used to solve the following problems:
  - (1) Three variables problem: $1 \#$ (all minterm with 3 binary variables).
  - (2) Four variables problem: $1 \#$ (all minterm with 4 binary variables).
  - (3) Five variables problem: $1 \#$ (all minterm with 5 binary variables).
- The C program is compiled by GNU C compiler version 2.7.2, and is run on Sun Ultra5 workstation with 64MB real memory.

Evaluation.

- The CCM is simulated using QuickHDL software from Mentor Graphics.
- We simulated the VHDL model of CCM, got the number of clocks used to solve the problem, then calculated the time used by CCM using formula: clock *$ clock-period.
- A clock of 1.33 MHz (clock period: 750 ns) is used as the clock of the CCM.

Compare CCM (1.33 MHz) with software approach

Problem	3 variables	4 variables	5 variables
Ultra5	111 usec	268 usec	812 usec
CCM	546×0.75 $= 409$ usec	1285×0.75 $= 963.75$ usec	3405×0.75 $= 2553.75$ usec
speedup	0.27	0.28	0.32

Evaluation.

- It can be seen from Table that our CCM is about **4 times slower than the software approach.**
- But, the clock of the CPU of Sun Ultra5 workstation is 270 MHz, which is 206 times faster than the clock of the CCM.
- Therefore, we still can say that the design of the CCM is very efficient for cube calculus operations.

Evaluation.

- It also can be seen from Table   that the more variables the input cubes have, the more efficient the CCM is.
- This is due to the software approach need to iterate through one loop for each variable that is presented in the input cubes.

Evaluation.

- However, the clock period of 750ns is too slow.
- From the state diagram of the GCU, it can be found that the delays of **empty carry path** and **counter carry path** only occur in a few states.
- Thus, if we can just give more time to these states, then we can speedup the clock of the whole CCM.
- This is very easy to achieve: for example, the state P2 of GCU needs more time for the delay of counter carry path, so add two more states in series between states P2 and P3.

Evaluation.

- These two extra states do nothing but give the CCM two more clock periods to evaluate the signal **prel_res,** which means that the CCM has 3 clock periods to evaluate signal **prel_res** in state P2 after adding two more ``delay'' states.
- After making similar modifications to all these kind of states, the CCM can run against a clock of 4 Mhz (clock period of 250 ns).

COMPARE CCM (4MHZ) WITH SOFTWARE APPROACH

Problem	3 variables	4 variables	5 variables
Ultra5	111 usec	268 usec	812 usec
CCM	611×0.25 $= 152.75$ usec	1486×0.25 $= 371.5$ usec	4078×0.25 $= 1019.5$ usec
speedup	0.72	0.72	0.80

Evaluation.

- It is very hard to increase the clock frequency again with this mapping because some other paths like **memory path** have delays greater than 150 ns.

Experimental Results

- Speedup on 3 variables is 0.72, 4 variables - 0.72, 5 variables - 0.8
- Frequency of FPGA 3090 was 4MHz
- Frequency of Sun Ultra was 270MHz
- If we map the entire CCM into one chip delay would be reduced
- New chips are faster and denser.
- The delay of CLB of 3090 is 4.5 nS, the delay for CLB of 4085XL is 1.2 nS.
- 4085 has array 56 * 56 and 448 user I/O pins.

Experimental Results

- We can map entire CCM into one 4085
- Clock of 4085 is 20 MHz
- Clock of CCM is five times slower than Sun
- CCM will run 4 times faster than software approach

RESULTS OF COMPARISON

- A design like CCM with a complex control unit and complex data path is not good for the architecture of the DEC-PERLE-1 board.
- It can be seen from our CCM mapping that since a lot of signals must go through multiple FPGA chips, this leads to greater signal delays.
- For instance, if we can connect the memory banks and the registers directly, then the memory path has a delay of only 35 ns. But our current memory path has a delay of 160 ns.
- Another issue is that XC3090 FPGA is kind of ``old'' now (8 years old technology).
- The latest FPGAs from Xilinx or other vendors have more powerful CLBs and more routing resource, and they are made using deep sub-micron process technology.

POSSIBLE IMPROVEMENTS

- Mapping the entire CCM inside one FPGA chip would speedup the CCM:
- If we map entire CCM into one FPGA chip, the signals do not need to go through multiple chips again, which means the routing delay is reduced.
- Since the new FPGA chip has more powerful CLBs and routing resource, we can map the CCM denser. This also reduces the routing delays.
- Since new FPGA chips are made using deep sub-micron technology, the delay of CLB and routing wires are both reduced.
- For example, the delay of the CLB of XC3090A is 4.5 ns while the delay of CLB of XC4085XL (0.35 micron technology) is only 1.2 ns. This means that it is very easy to achieve 3 times faster mapping.

NEW FPGA CHIPS FOR NEW VERSION

- XC4085XL FPGA from Xilinx has a CLB matrix of **56 * 56** and up to 448 user I/O pins.
- The CCM should be able to map into one XC4085XL FPGA.
- It should not be difficult to run the CCM against a clock of 20 MHz (clock period: 50 ns).
- This means that our CCM will be about 4 times faster than the software approach while the system clock of the CCM is still 5 times slower than that of the workstation.

CONCLUSIONS

- Principles of Learning Hardware as a competing approach to Evolvable Hardware, and also as its generalization.
- Data Mining machines.
- Universal Logic Machine with several virtual processors.
- DEC-PERLE-1 is a good medium to prototype such machines, its XC3090A chip is now obsolete.
- This can be much improved by using XC4085XL FPGA and redesigning the board.
- Massively parallel architectures such as CBM based on Xilinx series 6000 chips will allow even higher speedups.

Thank you