Configurable Hardware: A New Paradigm for Computation [*]

J. P. Gray
European Silicon Structures Ltd.

T. A. Kean
University of Edinburgh

At present there are two main methods of implementing algorithms: interpretation of a data stream representing a program by an active processing unit (software) and interconnection of active logic elements (hardware). In one case the computation performed is dependent on data stored in memory and in the other on the interconnection between a set of physical devices (transistors). Both paradigms can be shown, given reasonable definitions, to be essentially equivalent in terms of the functions they can compute (see, for example, [Savage76]). In this paper we will make the case for a third paradigm: Configurable Hardware in which the interconnection between active logic elements, and hence the function computed, is dependent on a control store.

1 Introduction

This paper describes a computing stucture that is both general purpose and application specific. It is termed configurable hardware. We seek to expose a novel model of computation by demonstrating implementations of non-trivial algorithms that exhibit significant cost/performance improvements over more traditional implementations. The paper also suggests that it may be time to reconsider a long dormant line of computer genealogy [Minnick67, Shoup70] originally developed by Vonn Neumann, among others. The proposed model of computation is supported by a novel chip level architecture [Kean88] that is based on regular arrays of very fine-grain processing elements. The overall architecture has its origins in a confluence of observations on progress in

[*]To Appear in Proc. Decennial Caltech Conference on VLSI

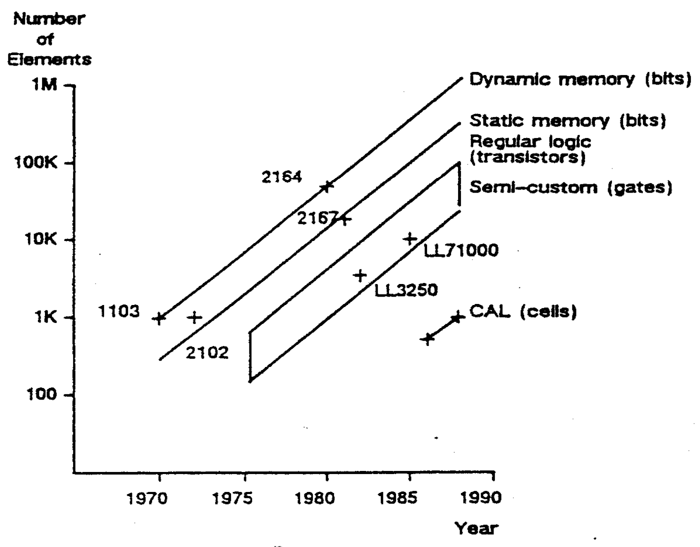

Figure 1: Technology Progress

the areas of semiconductor technology, figure 1, concurrent computing, figure 2, and algorithm implementation, figure 3.

In the first Caltech conference Moore [Moore79] restated his 'first law' and drew attention to the traditional bottlenecks of the semiconductor industry. We have updated his graph and added new data, figure 1, from the emergence of various ASIC families and markets over the recent past. What is interesting, apart from the inexorable increases in density with time, is the clear emergence of an enabling level of technology at about one thousand 'basic' units - bits, gates or cells. That is, it seems that whenever a particular technology is capable of delivering about a thousand units, an associated business moves into exponential growth. Today the so-called EPLD technologies are at this point, although only addressing a small applications area. However, if historical precedent is followed, we can expect an explosive growth in the density of such devices to the point at which configurable hardware becomes an implementation option for relatively large systems.

With respect to concurrent execution engines, figure 2 identifies current technology limits in chip and box level systems [Hillis85], [Seitz85]. It is interesting that, if provision of an absolute maximum of computing elements is important, then there is obviously more to be gained with fine-grain computation where presently box level products do not match the potential level of concurrency of application specific chips.

There are many important algorithms, notably the cellular-automaton simulations becoming popular in the physics community [Wolfram86], which require such fine grained concurrency and it is conjectured that configurable hardware products will allow exploitation of appropriate chip level technology and redress the imbalance between hardware and software with fine-grained programmable computation.

Finally, figure 3 is a crude summary of the options facing an engineer with a specific applications problem. The purpose of the figure is to draw attention to some of the less obvious implementation options: in particular, the routes to configurable hardware designs. This paper shows 'existence proofs' for both routes identified in the figure. The performance multipliers, based on [Culloch88, Efland88, Viitanen88], only indicate approximate relativities. The figure also highlights a potential for a revival of interest in the general purpose cellular-automaton computers suggested by [VonNeumann66]: while the cellular architecture described here cannot directly offer all the capabilities he foresaw it can provide the basis of a powerful testbed for new automata designs. If his insights on cellular automata carry the same weight as his early work on computing, we may see the revival of a major branch of computer architecture hand in hand with a major standard parts industry based on configurable hardware.

In this paper we describe a general fine-grained architecture, Configurable Logic and a specific implementation, Configurable Array Logic (CAL) currently under evaluation. Using this implementation we show how two significant applications can be mapped onto the architecture and give performance predictions for the implementations.

2 Configurable Logic

The basic configurable logic structure is a rectangular array of identical cells with the same orientation and nearest neighbour connections, figure 4. Each cell has a simple function unit and a permutation network. Non-local connections must be routed through intermediate cells. A range of such architectures is possible according to the complexity of the function unit and routing network within each cell. The choice of this basic structure is dictated by a basic property of VLSI: planarity. In this paper we will consider only rectangular cells but other geometries are possible. Cellular structures have the important properties of having a single resource (cells) and infinite expandability by using multi-chip arrays.

The key design decision behind the CAL architecture was to min-

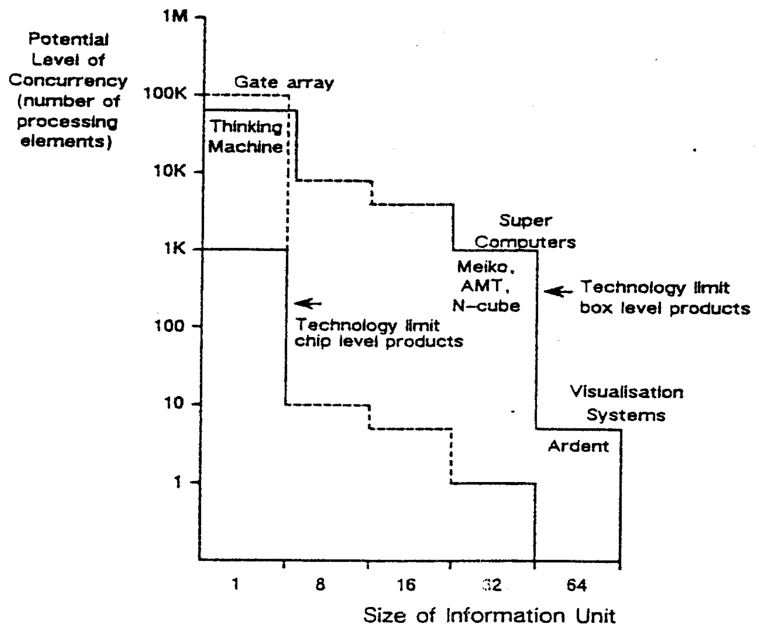

Figure 2: View of Concurrency

imise the complexity of the basic repeating cells. This produces a large array of small cells rather than the small array of large cells [Xilinx86]. Thus, building blocks for particular algorithms are formed from an aggregate of underlying cells rather than a subset of a larger unit. This reduces the variance on the efficiency of implementation of different designs.

3 The Configurable Array Logic Design

3.1 Routing Network Design

The routing network provides one bit wide input and output connections between each cell's North, South, East and West neighbours and its own function unit inputs and output. A fundamental problem with having only nearest neighbour connections is that propagation delay increases quickly for long wires. This problem is particularly severe in the case of clock wires and special measures have been taken to overcome it: two global signals are routed to all cells in the array without passing through any multiplexers. These signals are intended to be

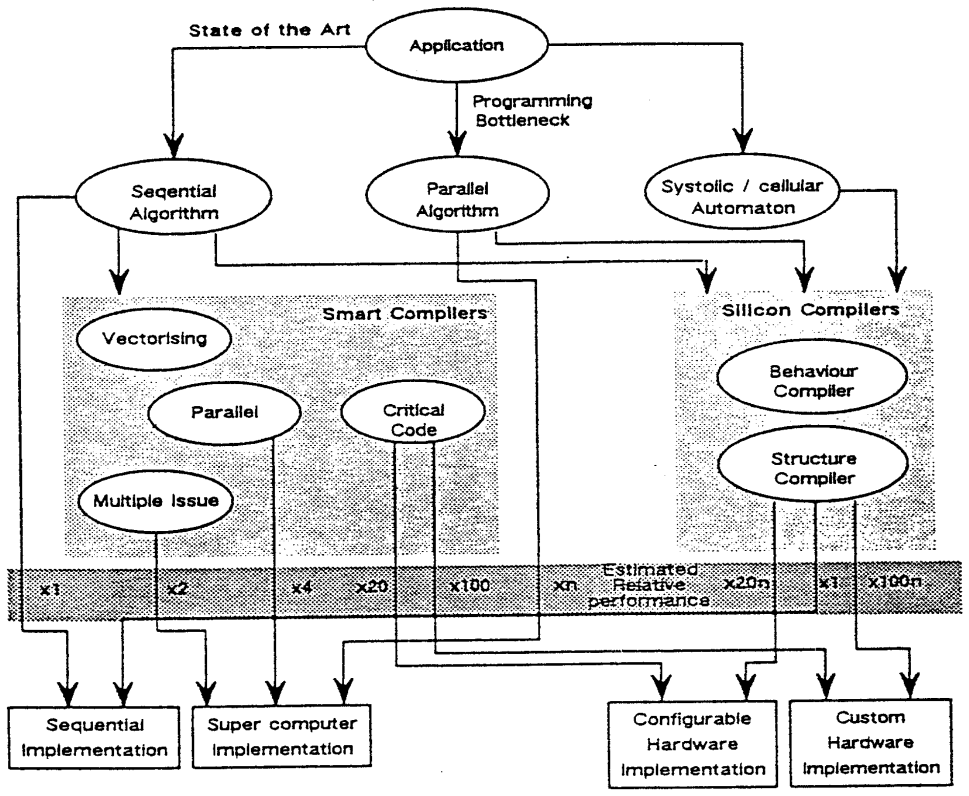

Figure 3: Implementation of Algorithms

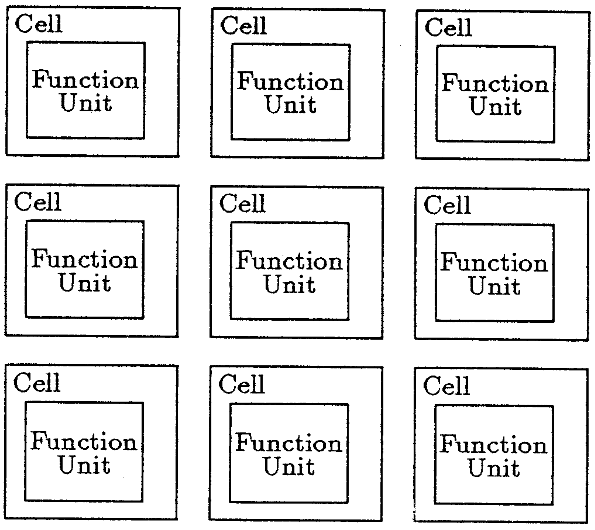

Figure 4: Basic Structure

used as clocks by user designs. This has the useful side effect of freeing a large number of connections for use by data. These global signals are often used to clock pipeline registers implemented using the cell's latch function within user designs. The routing structure is shown in figure 5. Note that because every output has its own multiplexer there is no possibility of one connection through the cell 'blocking' another as can happen in 'wiring-channel' like architectures.

An additional global output signal *FTEST* allows users of the system to monitor the output of any cell function block within the array for debugging purposes without the problem of routing it to the edge.

3.2 Function Unit

Given that we are using a two input function unit to minimise cell complexity we choose to minimise the number of function blocks required by making any 2 input 1 output combinational logic function implementable within one cell. The present design provides 4 kinds of D latch as well. The ability to obtain a latch with only one cell is important from the point of view of density.

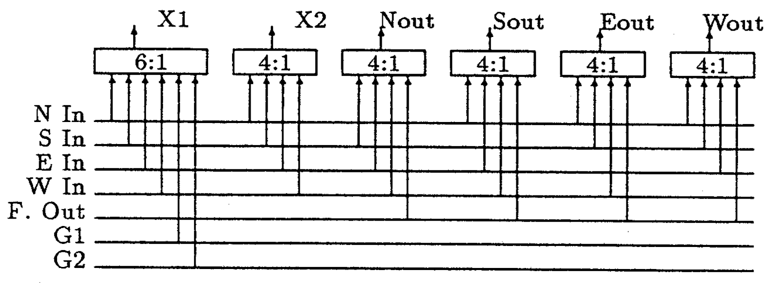

Figure 5: Cell Routing

A block diagram of the function unit within each cell is given as Figure 6. It relies on the fact that all functions of two variables, X1 and X2, can be computed by a 2:1 multiplexer (marked F in the diagram) which selects Y2 if Y1=1 and Y3 if Y1=0 [Chen82]. Note that the entire cell function is implemented by multiplexers and memory cells: both of these structures elegantly exploit the imperatives of the MOS medium.

4 Case Studies

4.1 Data Encryption Standard (DES) Example

DES [NBS77] is a substitution cipher on 64 bit binary vectors based on a 56 bit key. In 1980 the standard was updated with several new modes of operation intended to make key search more difficult [NBS80]: the unit described here implements the original Electronic Code Book (ECB) mode. The key inner-loop code where acceleration is critical is common to both standards.

The detailed design of the DES encryptor is presented in [Kean88]: here we will be content with presenting the cellular design for the f-box unit which performs the inner loop computation. The floorplan is given in figure 7 and the cellular layout in figure 8. In this diagram the grey boxes are individual cells, the black boxes within the cells represent 'active' function units. At lower scales the function being computed by each cell and port information would also be printed. Note that the pipeline registers mentioned below are not shown in this layout, pipelining is extremely easy to provide and consists simply of

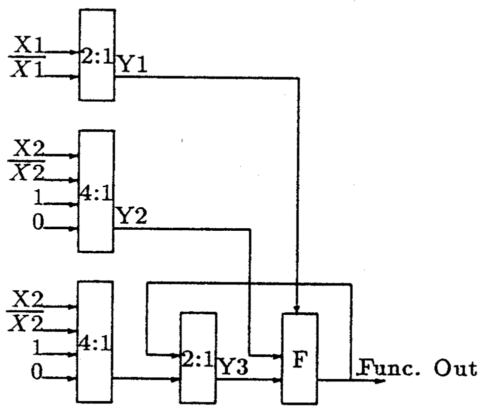

Figure 6: CAL Function Block Design

extra columns of cells each of which implements a D latch, the registers are omitted to allow the layout to be presented on a single page at a reasonable scale.

The performance of DES implementations can be measured in terms of the number of encryptions per second with the same key and different data. It was decided to aim for a throughput of 500,000 encryptions per second with a constant key. To achieve this one 64 bit ciphertext word must be calculated every $2\mu s$, since the f-box unit is used 16 times for each ciphering operation partially ciphered data must leave the f-box every 125ns. This is achieved using an 10 stage pipeline within the f-box. Table 1 shows figures for some DES implementations: the figures for workstations are using the *crypt* function in the UNIX C library (this implementation is not particularly efficient). The time for a single encryption is given as well as the number of encryptions per second to show the effect of pipelining. The time for the custom chip is taken from [Hoornaert88], this claims to be the fastest DES chip available with a throughput of 32M bits/sec (or 500,000 64 bit words/sec). Note that this chip has extra control logic to handle the more complex modes of DES in [NBS80] and also provides several registers for secure storage of keys on the chip. The CAL design makes use of several techniques developed by the group which designed this chip. Although the CAL appears to be as fast it should be pointed out that $3\mu m$ processing technology was used in the custom chip which would be expected to be slower than the $2\mu m$ technology used for the CAL. The CAL figures are based on worst case circuit simulations of cell computation

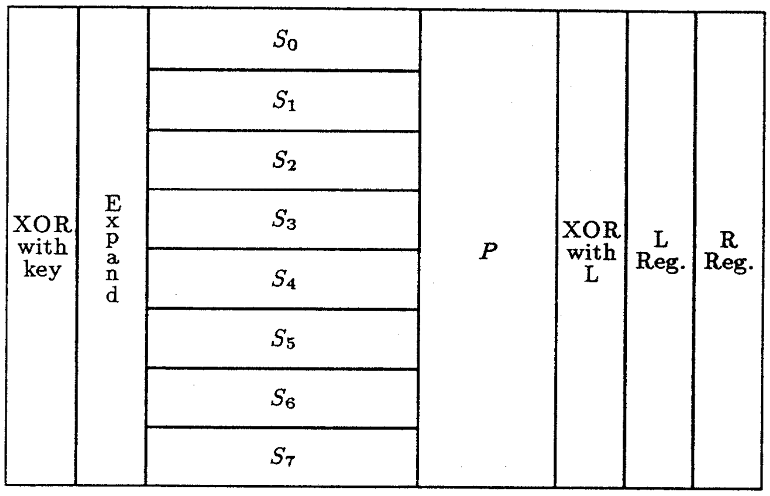

Figure 7: f-box Floorplan

Implementation	*Encryption Time*	*Encryptions/Second*
CAL	$60\mu s$	500,000
Custom Chip	$2\mu s$	500,000
Workstation (SUN 3/50)	0.5s	2
Workstation (SUN 3/260)	0.22s	5

Table 1: DES Encryptor Comparison

and routing delays rather than measurements on fabricated chips (now under test).

The CAL implementation of DES (including pipeline registers) requires an array of cells 88 cells high by 90 cells wide (27 for the key controller, 60 for the f-box and 13 for *IP*). This could be provided using a 6 by 6 array of 16x16 (prototype) CAL chips or a 3 by 3 array of 32 × 32 (volume part) CAL chips [Algotronix88].

4.2 Fluid Flow Simulation

Of particular interest are the cellular automata models for fluid flow simulation. The goal of these models is to set up a 'universe' based

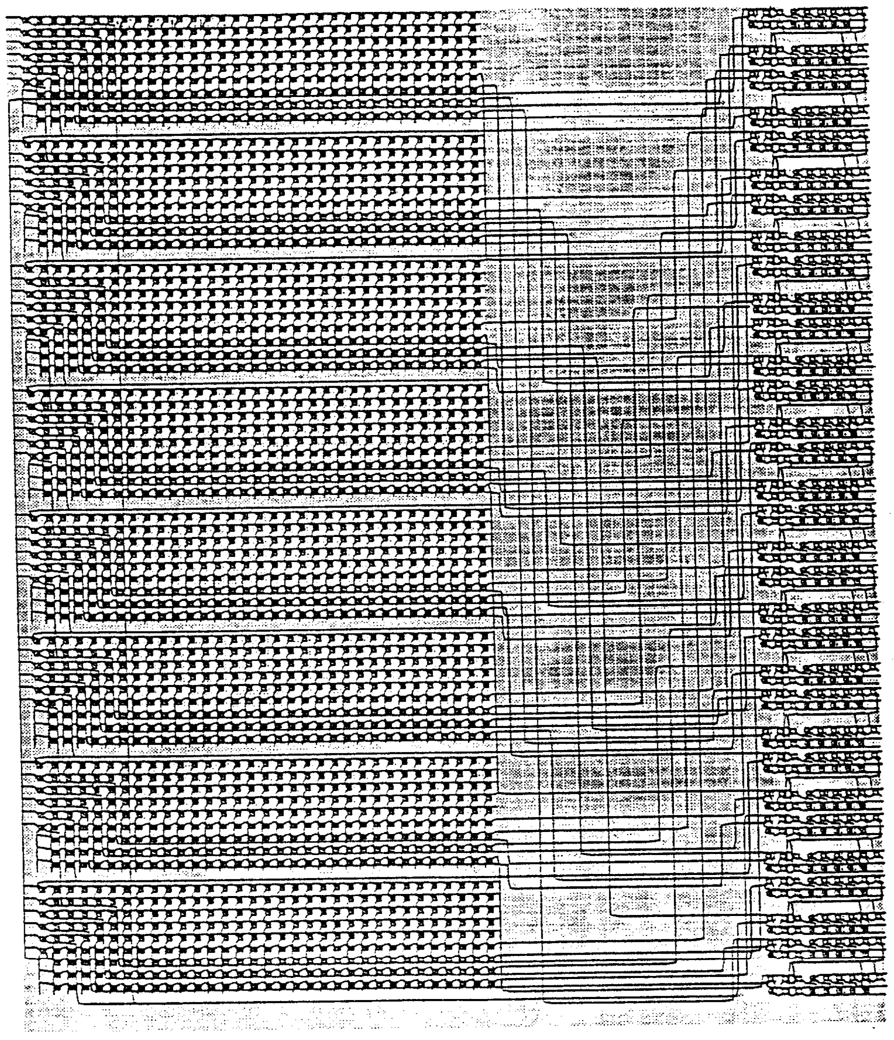

Figure 8: f-box layout

on a particular cellular automaton rule which mimics physical reality in some way and observe its evolution. Although there was initially a degree of scepticism, recent work on validating the cellular models has gained them wide acceptance [Frisch86],[Wolfram86],[Frisch87],[Fu87]. In this section we outline the design of a configurable logic based machine for solving such problems.

Hexagonal Lattice Gas Model The most commonly used cellular automaton rule for fluid flow simulation is called 'hexagonal lattice gas' and simply specifies what happens when molecules collide on a hexagonal grid. All molecules are assumed to have the same mass and velocity: the density of the grid (i.e. the proportion of lattice sites which are populated at the start of the simulation) is proportional to the fluid's Reynold's number. An example rule is shown in figure 9. In the 'before' step all the arrows face towards the current site and in the 'after' step all the arrows face away: thus the situations can be coded up as six bit numbers representing the presence or absence of a particle travelling in a particular direction. Note that to model real fluids properly the rules must obey certain physical properties (e.g. conservation of momentum) and are symmetrical with respect to rotation and reflection.

To obtain interesting results we usually wish to place some object within the fluid. At those sites on the edge of the object a different set of rules will be used - e.g. particles reflect back with angle of incidence equal to angle of reflection. It is inefficient to reconfigure the update unit when these sites are being computed so the normal technique is to add an extra bit plane which contains ones along the outline of the obstacles. We then use an automaton rule with 7 inputs and 6 outputs (since the obstacles do not move) specified as the normal 6 bit rule when the 7th plane is 0 and the reflection rule when it is 1.

Although the basic computation is very simple, to get good results it must be repeated on a large number of grid points (2048x2048 is not uncommon) and perhaps 10,000 iterations must be run between samples. One such cycle would involve about 42×10^9 node updates. After sufficient number of cycles have been run the model is divided up into larger sectors with about 64 sites per sector, the average direction of the molecules within each sector is converted into an arrow whose length represents the number of particles heading in the most common direction and a diagram (see, for example, [Wolfram86]) is produced clearly showing the structure of the flow.

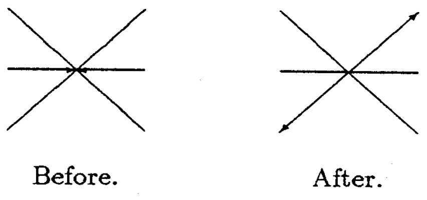

Figure 9: Example Lattice Gas Collision Rule

Architecture The system described here is modelled after RAP1 [Cloqueur87]. The memory subsystem is treated as several bit planes, figure 10, the values in the planes at a particular address correspond to the 'before' data for the corresponding site. The computation unit performs the update and writes back the new 'after' value. After all sites have been updated the memory subsystem manipulates bitplane address offsets to perform a separate 'movement' phase in which the bit planes are 'realigned' so that in the next computation sweep cells again have 'before' values. For example, to move a bit plane 'up' one would add an offset of the number of pixels in a row. This assumes that memory addresses 'wrap around' at the edge of the bit plane. Wrap-around is actually very desirable in a cellular automaton system since otherwise the fact that edge sites have no neighbours in one direction can distort the results.

Update Unit Implementation The first thing worth noting is that the case where the 'object' bit plane is 1 is very simple, reflection simply involves a swapping operation on inputs. The update unit is composed of an automatically generated logic block (termed CL-ROM) for the general case plus some wiring area to deal with the simpler reflection case.

There are 8 pipeline stages in the CL-ROM plus one for the extra multiplexing. Use of dynamic memory suggests a pipeline tick of 120ns to allow a memory access to occur and data be routed to the CAL in a single cycle. We will assume that 32x32 cell chips are being used. With 128 computation units (using 104 CAL chips) we can do 1.07×10^9 updates per second.

4.3 Comparison with Previous Systems

The configurable logic architecture differs from previous special purpose cellular automaton machines, e.g. CAM-6 [Toffoli87] and RAP1

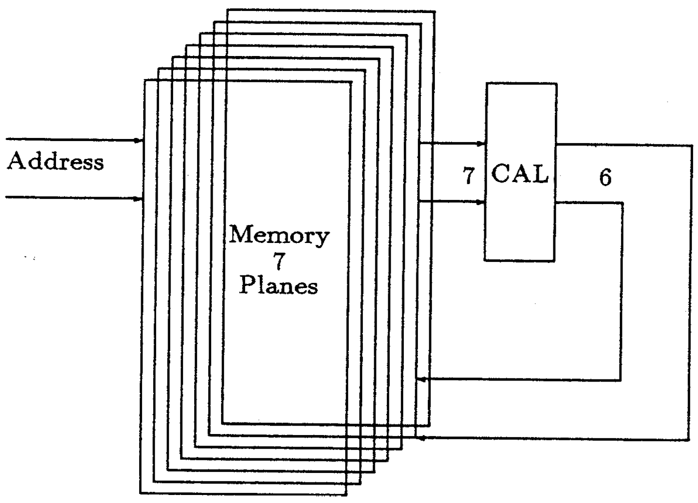

Figure 10: Basic Lattice Gas Simulator Architecture

System	*Site Updates/Second* $\times 10^6$
CRAY X/MP	500
Connection Machine	1000
Single Princeton Chip	20
Princeton System	1000
Single 64x64 CAL	40
CAL System	1070

Table 2: Performance on Fluid Flow Simulations

[Cloqueur87], by having many processors and using configurable ROM's rather than lookup tables for calculations. The main disadvantage of the lookup technique that the size of the RAM required grows as 2^n with the number of inputs n. RAM size and number of processors must also be fixed in advance - you do not get more processors when the rule is simple. Cellular automaton models with as many as 24 inputs have been suggested to increase the accuracy of the simulations. Naturally, if the rules were completely random functions of n variables configurable logic could do no better than lookup tables: however cellular automata rules for lattice gas equations are very regular (this follows from basic properties of the physical model such as conservation of momentum and symmetry).

We can see the advantages of the configurable logic approach from the example above: if a RAM lookup table had been used then the size of the RAM would be doubled by the seventh input bit, however with the CL implementation only a very small additional piece of logic is required. It is likely, therefore, that despite the number of inputs in the newer rules an implementation using relatively few logic gates will be possible. The structure of this implementation will vary from rule to rule. In the lattice gas model of this example rotation and reflection symmetry could be used to reduce the number of rules from 64 to 14 [Wayner88].

With a modest number of chips the Configurable Logic system can easily outperform any conventional computer on this problem: table 2 compares the expected performance of the system with supercomputer implementations and custom chips (the CAM-6 and RAP1 cellular automaton machines have much lower performance because of their single update unit). The first Princeton figure is a maximum per chip. There are 1500 chips in the present configuration, however I/O considerations limit performance to 1000 million sites/second. The Connection Machine and Princeton Chip figures come from [Wayner88], the figures for the CRAY are from [Fu87]. This is a very gross comparison because the cell update computation is organised in different ways in each system and there are vast differences in implementation technology, however, they do represent the state of the art.

5 Prototype CAL Chip

The prototype chip contains a 16x16 array of cells and has a core (i.e. before pads) symbol size of 4817 by 4596μm in 2μm double metal CMOS. It is a cut down version of a 32 $\times$ 32 cell design for safer prototyping. All peripheral signals from the array were brought out to pads and, after taking into account power, programming and the global G1,G2 and FTEST signals the chip fits exactly into a 144 pin Pin Grid Array (PGA) package.

The control store on the prototype chip is set up to look like a long shift register and the chip is programmed by shifting in a serial data stream. This data is generated automatically from an 'assembly-language' textual format where every cell function and multiplexer source and sink is declared explicitly. A suite of CAD programs which produce this textual format is described in [Kean88].

At time of writing, the prototype chip has been tested to the point of demonstrating full core functionality as defined in [Kean88], and a counter implemented using the cells has operated succesfully at 50MHz, which was the limit of the available test equipment.

6 Conclusions

In this paper we have demonstrated that significant algorithms can be implemented in Configurable Hardware with potentially large improvements in performance and reduction in cost. However, underlying all of this is a very general paradigm for computation in which parts of algorithms are mapped into hardware by active compilers. The CAL array represents a first step along a path which blurs the distinction between hardware and software. In the past hardware has been regarded as being completely fixed, with configurable hardware it can be restructured for a given algorithm. We can expect the rate of configuration change to increase, as new programming learning curves are climbed, leading perhaps to the self-configuring, fault tolerant systems envisioned by [VonNeumann66].

7 Acknowledgements

The authors wish to thank the UK Science and Engineering Research Council for a postgraduate studentship and B. Pruniaux, ES2, for fabricating the prototypes.

References

[Algotronix88] Algotronix Ltd. *CAL1024 Preliminary Data Sheet.* Edinburgh, UK. 1988.

[Chen82] X. Chen and S.L. Hurst. *A comparison of universal-logic-module realizations and their application in the synthesis of combinatorial and sequential logic networks.* IEEE Transactions on Computers, 31(2):140–147, February 1982.

[Cloqueur87] A. Clouqueur and D d'Humieres. *RAP1, a Cellular Automata Machine for Fluid Dynamics.* Complex Systems 1, 1987 pp 584-596.

[Culloch88] A. D. Culloch. *Parallel Programming Toolkit for 3L-C, Fortran and Pascal.* Proceedings of the 8th Technical Meeting of the OCCAM User Group. IOS, Amsterdam 1988.

[Efland88] G. Efland. Private Communication.

[Frisch86] U. Frisch, B. Hasslacher, and Y. Pomeau. *Lattice Gas Automata for the Navier-Stokes Equation.* Physical Review Letters, Vol 56 no 14, April 1986 pp 1505–1508.

[Frisch87] Uriel Frisch, Dominique d'Humieres, Brosl Hasslacher, Pierre Lallemand, Yves Pomeau and Jean-Pierre Rivet. *Lattice Gas Hydrodynamics in Two and Three Dimensions.* Complex Systems 1, 1987 pp 646-703.

[Hillis85] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge Mass. 1985.

[Hoornaert88] Ingrid Verbauwhede, Frank Hoornaert and Joos Vandewalle *Security and Performance Optimisation of a new DES Data Encryption Chip.* IEEE Jornal of Solid State Circuits, Vol 23:647–656, June 1988.

[Kean88] Tom Kean. *Configurable Logic: A Dynamically Programmable Cellular Architecture and its VLSI Implementation.* PhD Thesis.

[Minnick67] R.C. Minnick. *A Survey of Microcellular Research.* J. ACM, 14(2):203–241, April 1967.

[Moore79] G. E. Moore. *Are We Really Ready for VLSI.* Proc. Caltech Conference on VLSI, January 1979.

[NBS77] National Bureau of Standards. *Data Encryption Standard.* Fed. Inf. Process. Stand. Publ. 46, January 1977.

[NBS80] National Bureau of Standards. *DES Modes of Operation.* Fed. Inf. Process. Stand. Publ. 81, December 1980.

[Savage76] John E. Savage. *The Complexity of Computing.* John Wiley and Sons, 1976.

[Seitz85] C. L. Seitz. *The Cosmic Cube.* Communications of the ACM, Vol. 28 No 22,1985.

[Fu87] Tsutomu Shimomura, Gary D. Doolen, Brosl Hasslacher and Castor Fu. *Calculations Using Lattice Gas Techniques.* Los Alamos Science, Special Issue 1987.

[Shoup70] Richard G. Shoup. *Programmable Cellular Logic Arrays.* PhD thesis, Computer Science Dept., Carnegie-Mellon University, March 1970.

[Toffoli87] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines.* MIT Press 1985.

[Viitanen88] Jouko Viitanen. Efficient Utilization of fine-grained parallelism with configurable cellular processor. Poster, 7th Kobe International Symposium on Electronics and Information Sciences, Kobe, Nov 21-22, 1988.

[VonNeumann66] John Von Neumann (completed by A.W. Burks) *Theory of Self-Reproducing Cellular Automata.* University of Illinois Press 1966.

[Wayner88] Peter Wayner. *Modelling Chaos.* Byte, May 1988, pp 253–258.

[Wolfram86] Stephen Wolfram. *Theory and Applications of Cellular Automata.* World Scientific Publishing Co, Singapore. 1986.

[Xilinx86] Xilinx Inc.. *The Programmable Gate Array Design Handbook.* San Jose, CA., 1986.