Image Pattern Recognition Using Configurable Logic Cell Arrays

J. Viitanen and T. Kean

ABSTRACT

A new approach to the solution of computation intensive problems using programmable logic cell array devices as dynamically run-time reconfigurable processors is introduced. The applicability of the approach for real problems is demonstrated in the case of two time-critical parts of image pattern recognition using the hierarchical chamfer matching method. The execution times of the programs performing certain computation intensive parts of the recognition process on a normal sequential computer and the new architecture are compared. Significant speedups are achieved over the sequential approach by exploiting optimisation and parallel processing at the gate level. The potential role of the approach in future computers is discussed, the application development process and the required development tools are described.

Keywords: *Image pattern recognition, VLSI design, computer architecture, template matching, parallel processing*

INTRODUCTION

The availability of logic cell array devices that can be programmed dynamically at run-time has made it possible to utilise a new approach for computation of data-intensive tasks. These devices are primarily intended for applications where traditional two-level Field Programmable Logic Array (FPLA) and the closely related PAL and PROM devices have been used extensively but are significantly more general in that arbitrary connections of gates and latches are supported allowing much more complex systems to be implemented. The run-time programmable/ reconfigurable logic cell array consists of a volatile block of static memory that defines the state of a switching matrix which connects the logic cell elements in the desired configuration, just like the fusible links configure the cells in an FPLA. The logic function of each cell is also defined by volatile memory. Hereafter, this kind of device is termed CLCA (Configurable Logic Cell Array) to emphasise the differences between it, the fuse-programmable and UV-erasable PAL devices and the fixed- logic gate array devices. In this paper we will consider two families of these devices: commercial Logic Cell Array (LCA) chips (Xilinx 1986) and a newer architecture called Configurable Array Logic (CAL) designed at the University of Edinburgh.

Even though the commercially available devices are not designed to be used in computer-type applications, their dynamic programmability makes it possible to load different 'programs' (the configuration data) into them for executing various types of computational tasks. Thus they can be used as coprocessors within a general purpose computer for speeding up specific tasks, where large amounts of data are processed using relatively simple algorithms. The Edinburgh system has been specifically designed with computational tasks in mind. The typical configuration of either CLCA for computational tasks is as a gate-level pipelined processor: but naturally any forms of parallelism which fit the device structure can be used. The examples that are presented in this paper deal with image pattern recognition using a template matching technique.

1. CHARACTERISTICS OF CONFIGURABLE LOGIC CELL ARRAYS

1.1. Background

The recently introduced CLCA is a descendant of the Field Programmable Logic Array (FPLA) devices that have been used for years for replacing various SSI or MSI logic chips on a typical digital circuit board. A traditional FPLA consists of logic cells that can be programmed to perform desired logic functions by opening the fusible links at proper positions inside the circuit. These devices are limited to implementing one or more two level AND/OR logic functions of input variables. Erasable Programmable Logic Devices (EPLD's) which can be reprogrammed after erasure by UV light were introduced later.

There are two main differences between the new CLCA devices and these architectures.

1. The programming is done using transistor switches controlled by static RAM cells. This has several important consequences:
   (i) The device is reprogrammable an arbitrary number of times.
   (ii) The store is volatile and must be restored every time power is applied, and
   (iii) The size of RAM cells limits the number of programmable connections in the device.

2. These architectures allow much more general interconnection structures than earlier devices. There are fundamental reasons which imply that higher generality arrays rather than simply larger two level arrays are necessary to implement large systems on a configurable architecture (Kean 1989).

1.2. The Commercial LCA Devices

The commercial LCA device is modelled after the gate-array architecture common in Application Specific Integrated Circuit (ASIC) designs. It consists of a relatively small number of complex logic cells separated by a wiring area. There are two families of these devices: the 2000 family contains 64 cells in an 8x8 grid and has been available for about two years and the 3000 family which supplies up to 320 cells and has just come into production. The 2000 family logic cells are capable of computing any function of 4 variables or any two functions of 3 variables and a complex flip-flop. The 3000 family cells can compute functions of 5 variables or two functions of 4 and have two flip-flops: the inter-cell routing structure is also slightly more complex. Note that the amount of control store required to implement any function of n variables increases as 2^n (since there are 2^{2^n} possible functions) so with constant processing technology one could have around twice as many 4 variable cells as 5 variable cells. Cell delays are of the order of 10ns. These devices are primarily intended as stand alone chips to replace random logic within target systems. Complex programmable I/O blocks are provided which are very useful in EPLD applications, but mean that regular arbitrary sized multi-chip arrays cannot be built since the interconnection structure is broken up at chip boundaries.

1.3. The Structure of the University of Edinburgh CAL.

The Configurable Array Logic (CAL) devices developed in the University of Edinburgh are radically different from the LCA's. The basic architecture is a cellular array with only nearest neighbour connections: thus longer connections must run through intermediate cells. This makes the switching structures on long interconnections more apparent but does not necessarily imply that there will be more intervening switches than in an LCA type architecture with separate routing facilities. Each cell within the array can realise two-variable logic functions or simple latches as well as providing routing support for pass through connections.

The architecture of the CAL device is motivated by three main design goals: simplicity, regularity and efficiency. We will consider the implications of each of these in turn.

- Simplicity. Architectural simplicity means that users and, more importantly, Design Automation tools have a clean model of the structure. Since CAL is intended to implement large systems most resource allocation decisions will be made by computer programs. This makes the provision of large irregular function units much less attractive, such units can often be used effectively in hand-crafted designs but cannot be used as effectively by design automation software. The situation is similar to the CISC/RISC debate in Von-Neumann computers: compiler writers prefer a small well defined set of fast instructions. This goal also applies to the routing system where only a single kind of routing resource is provided. This allows channel-routing algorithms instead of the much slower and less efficient maze routing algorithms to be used.

- Regularity. It is important that a system to support high generality designs provides a completely symmetrical routing structure. This allows sub-designs from a library to be rotated and reflected to meet global floorplanning goals.

Similarly, the system has only one resource: the cell (instead of several resources e.g. logic cells, wiring channels and long lines). In a system with many resources a design can fail in several different ways (not enough logic blocks, not enough wiring tracks etc.) whereas in a system with a single resource it is only a question of how many cells are required. This property simplifies the design of Design Automation tools. Regularity is extended to arrays containing many CAL chips: the architecture makes the individual chip boundaries transparent at the cellular level. This allows single subunits to be split over multiple chips and greatly simplifies the placement problem. This transparency is possible because routing delays in cellular structures are significantly higher than those in silicon designs (because of intervening multiplexers). Input/Output pads have also been getting significantly faster and delays are smaller than normal in array structures where pads drive only a single input on an adjacent chip. Thus I/O pads are not necessarily a bottleneck in cellular designs (incurring a delay equivalent to around 3 routing multiplexers for a chip boundary crossing) and, with carefully designed circuitry, can even be shared between several signals at an array edge without excessive performance penalties.

- Efficiency. Efficiency in the use of silicon area is central to the CAL design. It falls out naturally from the previous two considerations. Since we have a large array of very simple cells it is possible to put a large amount of effort into finding an efficient layout for the repeating unit (c.f. commercial RAM designs).

The array structure of the CAL device is apparent from the example design in figure 5. Apart from the nearest neighbour signals there are three global signals routed to all cells in the array. Two are inputs (G1 and G2) which can be used as a two-phase non-overlapping clock in user designs; the third (FTEST) is a test signal which can monitor the function unit output of any cell in the array. This simplifies access to internal signals for debugging purposes. Each cell can implement any of the 16 functions of two Boolean variables or one of 4 types of D latch (D,Clk),(D',Clk),(D,Clk'),(D',Clk'). The routing structure within each cell is shown in figure 1: it approximates a full crossbar switch with unnecessary connections (e.g. South In to South Out) removed. The cell is controlled by 20 bits of RAM. Use of a multiplexer based routing structure makes very efficient use of the control store. The present prototype device contains a 16x16 array of cells in a core symbol of just under 4817x4596um in 2um CMOS. Much larger arrays are possible: a 32x32 array in 1.5um tecnology is currently under development (Algotronix 1988) and a 64x64 array would be possible with the advanced processing technology used in commercial SRAM's. The architecture of the CAL chip is described in detail by Kean (1989).

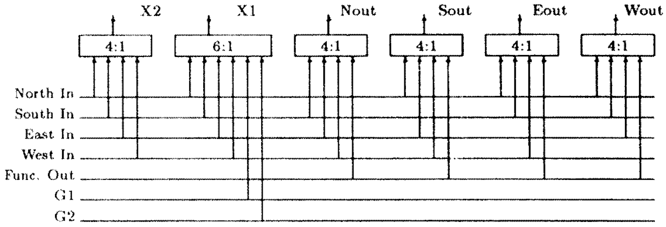


Fig. 1: The CAL cell routing.

2. THE APPLICABILITY OF CLCA's TO IMAGE PROCESSING

Image processing and image pattern recognition are fields where reconfigurable special processors could be useful. Early image processors like the Cytocomputer (Sternberg 1985) were fast pipelined processors performing a limited set of operations using a fixed neighborhood size. Typical functions were performed over a period of one complete image frame (several milliseconds); during this time the function of the neighborhood logic remained fixed. Many SIMD-processors have similar characteristics: performing fixed operations over many data items is very fast but changing programs is fairly slow (because code must be propagated to each processor). In systems without a dedicated broadcast channel for code transfer programs commonly remain fixed over the time period of one image frame or window (although the programs stored in each PE may be quite long). Examples of such SIMD-processors are CLIP (Fountain 1983) and the NCR GAPP (Davis 1984). Some more general purpose processors, like the Connection Machine (Hillis 1985), have similar features. All these computers typically are coprocessors controlled by a general purpose computer which takes care of program and data traffic.

Image processing is an application area where a fast processor with a small 'program store' and slow 'instruction fetch' could be successful because algorithms often have loops with high repetition counts and relatively simple operations inside the loop body. Other potential target areas include statistical simulation, and traditional vector processor applications such as ray tracing, and finite element methods. In this paper we shall consider one possible system architecture using CLCA's consisting of a master CPU with a connected slave CLCA (or several of them). The master processor controls the configuration of the CLCA, sends data to it, and reads back the results. This architecture provides an attractive development environment but the need to channel data going to and from the CLCA array through the microprocessor can be a bottleneck in many applications. Architectures with high-bandwidth connections and fast memories are necessary to take advantage of the raw power of large CLCA arrays in highly parallel applications such as cellular automata algorithms (Kean 1989).

The principal difference between the CLCA and previous cellular array computers is its gate-level programming. This may appear to be a limitation in that each 'processor' can only execute very simple operations: however the reverse is true. Using behavioural compilation techniques developed for silicon compilers we can configure the array to implement almost any operation we desire (it is easy to show that CLCA arrays are general computing structures in the sense of Turing machines). We could even implement a complete Von Neumann processor on the CLCA if that kind of operation was required. Traditional cellular computers with their small function units are much more limited in this sense. The number of available cells in the CLCA and propagation delays on long wires within it provide a practical limit on the size of the functional units implemented.

There are several types of image processing operations that could be considered for implementation on a CLCA. Some obvious ones are preprocessing tasks, like image smoothing with simple masks, edge detection, median filtering, and thresholding. Other possibilities are histogram calculation, searching for maximum or minimum, and similar tasks. However, in the coprocessor architecture we are considering the speedup for these operations may be quite low compared to a program run on a high performance signal processor or a RISC (reduced instruction set computer). This is because these operations are so simple that data I/O overhead will be a major part of the execution time and the host CPU may be able to calculate the function itself almost as quickly as it can transfer the data to and from the coprocessor. Special processors are already available with real-time execution speeds for many of these basic image preprocessing functions.

Operations with intermediate complexity, where the I/O overhead percentage is lower, but large image data sets are processed are better suited to the external coprocessor architecture. Examples of such tasks in the field of image pattern recognition are pattern matching, distance calculation using different metrics, coordinate transforms and Hough transforms. This paper will describe the use of a CLCA in two important steps of pattern recognition, the distance image calculation and hierarchical matching. The original implementation was done using commercial LCA devices but we will also describe a CAL implementation of the distance transform unit to allow comparison. A system with a single LCA is assumed, with logic array reconfiguration at the beginning of each task. The execution speed of the new approach, based on simulations of the designed configurations of the LCA, is compared with a previously tested program on a standard sequential signal processor.

3. THE SAMPLE PROBLEM

Our example application, the Hierarchical Chamfer Matching Algorithm (HCMA), was first described by Borgefors (1988) and fast algorithms for its calculation were presented by Viitanen et al. (1987, 1988) for the three-parametric translation-rotation problem. The advantage of HCMA is its robustness but in previous implementations it had the disadvantage of relatively long execution times.

HCMA is a model based, template matching operation. It uses simple operations, like additions, searching for a minimum, and offset addressing into small tables. The main phases in the recognition are image capture, feature detection (typically simple edge detection and thresholding to a binary image), calculation of a distance image (an image where the pixel values are proportional to the distance, or approximated distance, of the pixel from the closest detected feature), hierarchical pyramid creation from the distance image (quadtree, octree, or even pentatree rules can be used), and finally, the actual matching. Coarse matching is done at the lowest resolution pyramid, trying every third or fourth translational position in both directions; rotation is estimated at the same time. Finer matching is done at higher resolution levels for a few best candidate positions.

The matching process involves transforming the model coordinates to different geometrically distorted positions with respect to the distance image. The practical case with three parameters, rotation and X-Y translation, is analysed in the references. Those values of the distance image that are addressed by the distorted model coordinates are picked up and accumulated. The accumulated sum is proportional to the 'average' distance of the model from the 'correct' position under the metrics used in the distance transform. This is a classical multidimensional optimisation problem where the distance image values form the cost function. The reduction in computational load over the traditional template matching metho (where the cross-correlation function is calculated between the model and the scene) is obvious since only additions a needed. Viitanen (1987, 1988) shows how the explicit polar coordinate transforms can be avoided in the geometr transforms. Furthermore, the local convergence properties of the distance image are good allowing several levels of t hierarchical representation of the image data to be used. This reduces the amount of data and the processing time consid ably.

Here we will improve the speed of the time consuming distance image calculation and matching parts of the algorith using CLCA acceleration. In (Viitanen 1988) the measured execution times on a sequential processor were 1.7 seconds f calculating the distance image, and 10 to 20 seconds/model for matching against a 256 by 256 image, with a maximum o

48 coordinate points in the model. For practical use in robotics, the processing times should be a few hundred milliseconds. We will show how to achieve this performance using CLCA's. The next section describes the calculations which must be performed in detail.

3.1. The Distance Transform Calculation

The 3-4 integer distance approximation is applied in the calculation of the distance transform (DT) used for creating the distance image. The approximation has small errors compared to the true Euclidean distance but is sufficiently accurate for practical use. The calculation of the distance transform involves two passes over the binary edge image, using the method of Borgefors (1986). The calculation in the first iteration is done as follows. Let F(x,y) be the two-dimensional discrete image array with row index x and column index y, where F(x,y) = 0 at valid feature points and maximum otherwise, then the corresponding distance image value for each array position in the first iteration is:

$$G(x,y) = \min\{\ b(x,y),\ G(x-1,y)+3,\ G(x,y-1)+3,\ G(x-1,y-1)+4,\ G(x+1,y-1)+4\ \} \quad (1)$$

where processing is done row-by-row, in increasing x and y values. The second iteration is similar, expect that now the input image F(x,y) is the result of the first iteration, decreasing index values are used, and the signs of the index offsets above are negated. Because of the addition, a new search for the minimum has to be done among all the five pixels every time the mask is moved in the image. This makes the distance transform calculation fairly slow on a sequential computer.

A block diagram of the hardware to compute the 3-4 DT is shown in figure 2. It contains five registers in two groups corresponding to the two mask rows. The outputs of the registers are fed to adders which add the correct offsets. The most important part of the circuit is the parallel comparitor section which selects the minimum of the five elements in one asynchronous, ripple-through process.

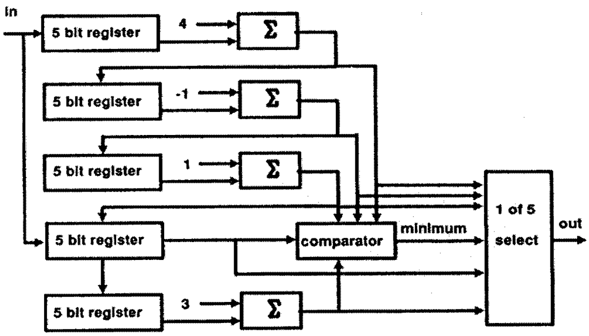


Fig. 2: The CLCA configuration for calculating the distance image.

3.1.1 The LCA Implementation.

The structure of the comparitor part of the circuit in figure 2 is shown in figure 3. It compares in parallel corresponding bits of the five pixels (marked by A ... E) with five bit accuracy. The result of the comparison ripples down from the MSB to the LSB, and the result is used to select the first pixel with the smallest value (several pixels may have this value). The circuit gives approximately constant time for the comparison from a certain bit significance level regardless of the difference. We could also have used faster lookahead to get a lower time for those comparisons where the minimum value could be determined at a high bit position but, since the host processor reads the results synchronously, the constant time approach was chosen. This unit consists of 84 gates, or the equivalent functional parts, in the LCA.

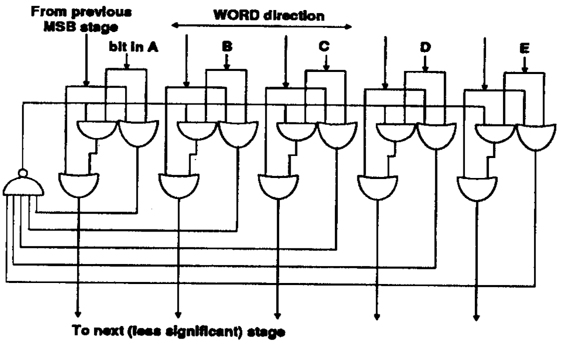


Fig. 3: One bit-slice of the parallel comparitor logic configuration.

Figure 4 shows the placement and the routing of the different functional parts on the LCA in this configuration. The smallest available LCA with 64 configurable logic cells was used: as we can see, only three cells were left unused and the routing capacity of the device was fully used at many positions. The smaller cells at the edges of the layout are programmable I/O blocks; a0...a4 and c0...c1 are the input terminals, and S0...SE4 output terminals, giving the selected minimum pixel value. Each logic cell has the inputs at the left, upper, and lower side, and two outputs at the right side. Clock is marked by 'ck'. Switching units are located in between the logic cells.

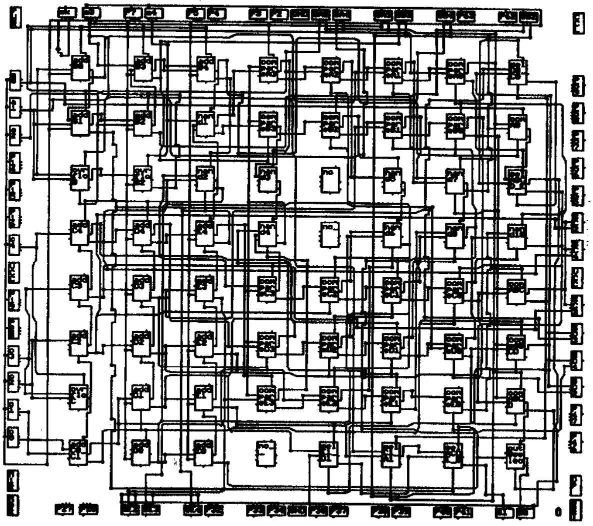


Fig. 4: The routing and placement of the LCA's logic cells of the configuration in Fig. 2.

The total delay through the comparitor is not a linear function of the equivalent gate delays along the signal path, because the LCA evaluates all Boolean functions up to four variables at the same speed; the performance of the circuit was estimated using the simulator supplied by the manufacturer. It gave a maximum value of 250 ns for the selection of the minimum from the worst case MSB transition. The maximum delay for one configurable logic block was 10 ns for the model used.

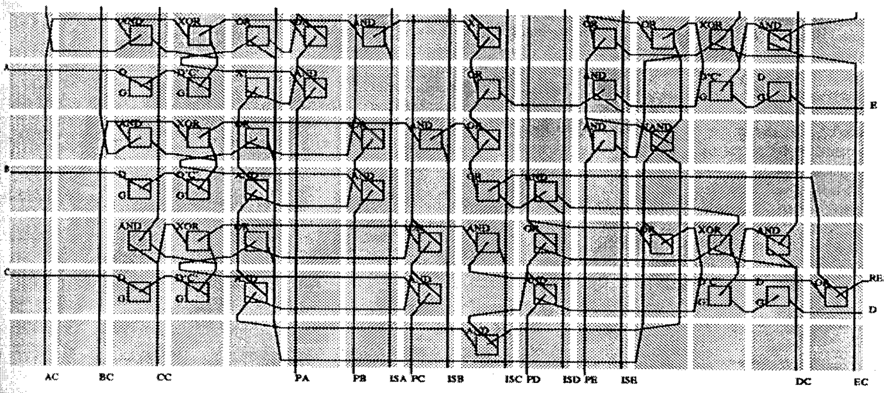


Fig. 5: The CAL design for the Distance Transform Unit.

3.1.2. The CAL Implementation.

Figure 5 shows a 'typical' bit slice through the CAL implementation of the comparitor. This design is different from the LCA one in that all the major units are integrated into a single regular structure. The adders are at the far left and right of the central logic. Half adders are used since we only need to add constants - when we want to add 1 at the current bit position we would use XNOR and OR instead of XOR and AND to generate SUM and CARRY to the next stage. Each adder looks like a 2x2 square with a master/slave register on the bottom and the SUM (XOR) and CARRY (AND) gates on the top. The registers are clocked by one of the global signals (G1). Carry routing goes from bottom to top on the left and right.

The selectors and mimimum detectors are implemented in the central area, 5 signals pA...pE go down the unit, and another 5 isA...isE up, pA...pE correspond to 'possibly' A through E and 'isA..isE' to definitely A through E (after all bit positions have been compared). The 'iSA...isE' signals are AND'ed with A...E and OR'ed (in the centre and far right of the design) to form a 5:1 selector and produce the result at this bit position (marked (RES)). The right hand side of the central area is less regular since it takes advantage of extra vertical space arising from only having two adders to save an extra column of cells.

The design is 14x7=98 cells, a complete comparitor would require 5 of these units. At the base (LSB) of the unit an extra row of cells is used to invert the 'pA...pE' signals and connect them directly to the 'isA...isE' signals. Potentially more than one 'is' signal can be high - this does not matter because the numbers corresponding to the high 'is' signals are equal so the selector will still give the correct result. At the top of the unit 3 rows of cells are used to connect the overflow signals from the adders to the corresponding 'p' signals in the MSB: this prevents pixels where the addition has overflowed from being selected as the minimum. The case in which all the additions overflow is also detected and the 'ovf' signal set which forces all result outputs to 1 (corresponding to the maximum legal value). The extra routing area could be avoided if special layouts were done for the MSB and LSB slices but this method produces a more regular design. The whole unit requires 546 cells.

The best way to compare this figure with the LCA is in terms of control store required since this will cancel out differences in processing technology and die size. The LCA array has 12038 bits of configuration RAM. At 20 bits of RAM per CAL cell this represents 601 CAL cells. This shows that the implementations are almost equivalent in terms of 'area' efficiency despite the large numbers of relatively high fan-in gates in the design. It must be noted that the LCA design has been fitted into a single chip where the CAL design is an arbitrary shaped rectangle of cells. This is representative of the different ways the two systems are intended to be used. CAL designs will normally be done in a large array built up from several chips and the comparitor would represent only a small part of the total system whereas LCA arrays are intended for relatively small designs which fit on a single chip. At present there are no facilities for accurately simulating the delays in a CAL design, however, rough calculations based on circuit simulations and measurements on the prototype chip suggest that the speed in this example would be of the same order.

The CAL design is much more regular than the LCA design and was produced faster (about 2 days versus 1 week) despite the fact that it was done as a hand layout whereas the LCA design used automatic routing tools and a good graphical editor. Most of the time in the LCA design was spent in manual editing of the routing since the autorouter quickly exhausts the routing resources in a design of this complexity leaving signals unrouted. Overall, the speed of design and its regularity coupled with the slight area advantage vindicate the 'keep-it-simple' philosophy of the CAL system.

3.1.3. Comparison with Conventional Processor

The corresponding operations on a TMS 32010 signal processor take from 29 to 37 instruction cycles of 200 ns, so the speedup factor is from 23 to 30, compared to the LCA implementation. The I/O overhead has to be added to both cases, depending on the actual implementation, so the total time for building the complete 256 by 256 distance image can be estimated to be about 64 milliseconds using a fast host CPU with a 120 ns I/O cycle time. The TMS 32010 sequential implementation took more than a second.

There are about 239 gate equivalents in the CLCA implementation, this number of additional gates would be needed in the ALU of a general processor to implement the same 'instruction'. No general purpose processor would be given this amount of additional gates for such a special purpose - even without the trend towards RISC computers! With 'reusable' reconfigurable arrays the special circuitry becomes feasible.

3.2. The Matching Process

The matching process utilises knowledge from both the model and the distance image. A window of the size of the model (or a part of it) is processed each time at every candidate position x,y of the distance image. In (Viitanen 1988) an algorithm is described that gives an estimate for the best matching rotational angle, as well as a measure of the fidelity of the match at that translational position using the best matching angle.

The matching is based on the following formulation: we denote by $F(\alpha,r,x,y)$ the distance image, and by $G(\beta,h)$ the model to be matched, both in polar coordinates. α and β are the angles with respect to the x axis, and r and h are the distances from the origin of the window where the calculation is carried out; x and y are the rectangular coordinates of the origin of the window on the scene. Both are discrete two- dimensional arrays. G( ) is a binary image with ONE's at the valid model feature points and zeros otherwise.

The estimation involves calculation of the following summation:

$$A(\alpha,x,y)=\sum_{n=1}^{N}\sum_{m=1}^{2}F(\alpha-\beta(n,m),h(n),x,y)G(\beta(n,m),h(n)) \qquad (2)$$

where N steps are taken over the valid (discrete) radial distance, and 2 model points are used in the calculation at distance n. The angle α that corresponds to the minimum value of A is taken as the best orientation of the model at position (x,y) and the minimum value itself is proportional to the average distance from the estimated correct position.

Figure 6 shows the block diagram of the CLCA in the configuration for calculating (2). Besides the logic cell array, two memories are needed. The first is used for storing the angle difference values of the steps along the 3-4 equal distance circle for every distance value of the polar coordinate representation. The second memory is used for storing the accumulated distance image values A( ) with the angle value as an address. The 'angle offset' register holds the model angle value at that distance. The minimum of A( ) and the corresponding angle are continuously updated and output as results of the calculation. As we can see from this example a computer design using CLCA's would benefit from some dual-port memory addressable by both the CLCA and the host for fast exchange of scalars, vectors, and tables.

A carefully optimised assembly language loop for this calculation took 30 cycles of 200ns on a TMS32010 signal processor giving a total time of 6000 ns. The critical part of the CLCA implementation which determines the speed of the matching is the loop that is formed by the 8 bit address counter, the RAM, the 5 bit adder, and the 5 bit latch. The 70 MHz version of the 64 cell LCA was used. The adder speed suffered from the slow carry lookahead possible on the LCA, so the worst case propagation delay from LSB in to sum out was 41 ns. We must also include the RAM update time of about 25 ns maximum from an address change using a fast device, the I/O latch and direct input delays of 20 ns maximum, and the address counter output delay at 10 ns. Adding a small safety margin, the total update time would be 100 ns. This gives a total speedup factor for this example of about 60.

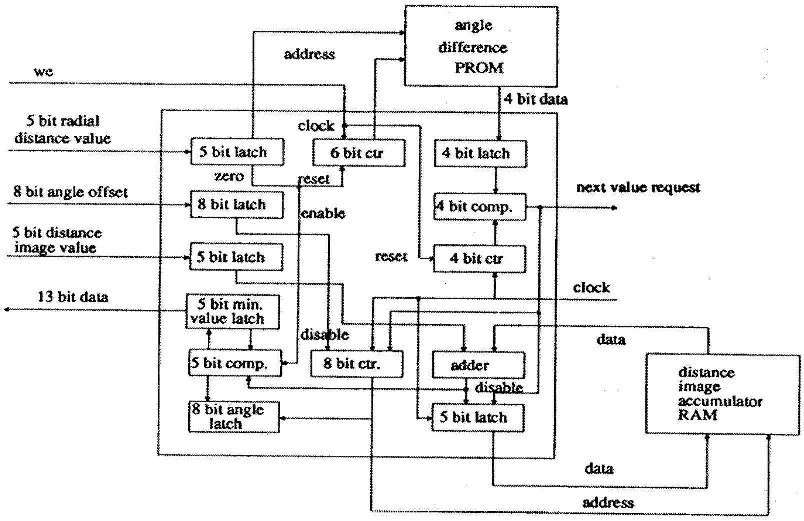


Fig. 6: The CLCA Configuration for the Matching Operation.

3.3. The Complete HCMA Calculation

We will now consider the total time for the whole operation rather than just the inner loop computation. In a typical case we might have the following time for performing a complete matching operation on one model, eg. for finding the best fitting location and the angle of rotation for the model in the image. We would use a three level pyramid, with dimensions 256 by 256, 128 by 128, and 64 by 64 at each level. The model has a size 63 by 63, with 40 valid model points, maximum of two points at each radial distance, and 24 radial steps at the lowest level, 12 at the next, and 6 at the highest pyramid level. At the highest pyramid level we would examine every third translational location, for a total trial count of 256. At the next lower levels we examine a maximum of 16 best candidate positions, and their 8 closest neighbors, and at the lowest level 4 candidates and the neighbors. So the total time at the highest level would be 100*256*2*6*256 ns = 78.6 ms. At the next level the corresponding figures are: 100*256*2*12*16*9 ns = 88.5 ms, and at the lowest level: 100*256*2*20*4*9 ns = 36.9 ms. The total time would thus be 204 milliseconds for one model. These times for the HCMA calculation are quite sufficient for use in robotics.

4. THE CLCA APPLICATION DEVELOPMENT PROCESS

In this section we will discuss the feasibility of implementing 'active' compilers for conventional high level languages which generate configuration data for CLCA's rather than normal machine instructions. These compilers are termed active because the computation is done by active computing elements (logic gates) rather than by a separate unit interpreting a passive byte stream.

Historically, many research projects have been carried out into behavioural compilation for catalogue part and silicon designs, notably the work at Carnegie Mellon University on the CMU-DA system and related projects (Thomas 1983). Recently, a highly developed system has been produced at IBM Yorktown Heights (Brayton 1986). This system features novel multi-level logic synthesis techniques and is probably the most fully engineered behavioural system available. We will model our discussion after the IBM compiler and another interesting system reported by Peng (1987). This uses Pascal as the behavioural description language and an asynchronous control strategy. The integrated development system is called CAMAD. CAMAD includes a data flow analyser that extracts the parts of the program executable in parallel.

The asynchronous control is realised as an ETPN (Extended Timed Petri Net) structure and the implementation merges the data manipulation and the control structure in the final realisation on silicon. CAMAD synthesises the control structure as centralised microprogram storage that monitors the guards of the operations and has control over the functional units (although Peng does not explicitly limit the model to consist of only one microcode engine and refers to other methods of control realisation). This kind of centralised microprogram control is not the most efficient way to realise the pipeline parallelism that is present in many image processing applications. A more distributed control strategy is dictated by the simple structure of the present CLCA models: they have logic cells that are easily usable for distributed control, but a centralised microprogram store would spend too many of the available routing facilities going to and from the operational units. Propagation delay on long control lines would also be a significant problem.

Figure 7 illustrates the proposed CLCA program development process from high level language source to configuration information. This diagram is typical of most silicon behavioural compilation systems and is modelled after Peng (1987). The process is normally split into two parts: in the first part the behavioural representation is converted into a structural one (e.g. a hierarchical netlist of gates) and in the second (sometimes termed silicon assembly) the structural representation is converted into physical layout. Tools for the second phase are fairly well understood so we will concentrate on the behavioural compilation step and take each component of the diagram in turn.

The use of a standard HLL as source is important in the hardware configuration we have in mind of a fast general purpose CPU (e.g. RISC or bit slice processor) and the CLCA as a coprocessor which obtains its configuration data via the host CPU. The 'active' compiler would then decide which parts of the program to run on the CLCA and which to run on the host by considering the execution times of both processors (taking into account CLCA configuration loading time). With present technology, only loops or parts of loops that have high enough repetition counts (say over a few hundred) would be considered for running on the CLCA coprocessor. There are many tradeoffs to be made in this part of the process: while fully automatic compilation from a language like C or PASCAL is desirable from the point of view of ease of use it can never take advantage of the full power of the CLCA. This is because the specification of the computation to be performed using a language like C is overly constrained. An obvious example of this is that a problem which only requires 5 bit precision may be coded to work on 32 bit integers in C: the 'active' compiler has no way of telling from the source code that the extra 27 bits are redundant. This problem occurs in more subtle forms as well: for example, a C programmer might specify a quicksort algorithm where a hardware implementation would be faster using a distributed bubblesort. It is sometimes impossible for the programmer to write efficient code without knowing which processor it will run on. For these reasons some degree of manual control over what parts of the program get implemented on the CLCA and more hardware oriented specification languages may be desirable.

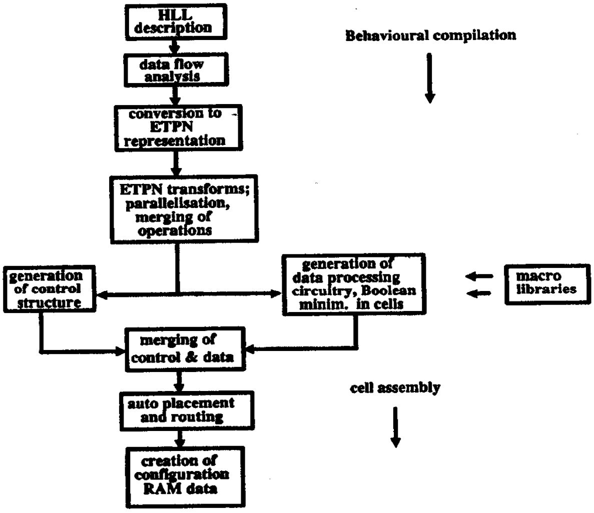

HLL description
data flow analysis
conversion to ETPN representation
ETPN transforms; parallelisation, merging of operations
generation of control structure
generation of data processing circuitry, Boolean minim. in cells
macro libraries
merging of control & data
auto placement and routing
creation of configuration RAM data
Behavioural compilation
cell assembly

Fig. 7: The proposed CLCA development process

In order to obtain reasonably efficient hardware realisations it is often necessary that many operations in the data-flow graph be performed by a single physical unit. There are two steps in figure 7, where this merging of operation can be done. The first comes, when the sequential program is transformed to the ETPN description. This utilises the control structure of the Petri net and compresses the straight line sections of the program with no external data dependencies to single operational units. The second merging step comes with the Boolean minimisation. Here, additions by a constant and similar ALU-only operations can be merged with the following operation. Programmable logic circuitry handles several variables at a time providing another useful speedup over traditional processors. A third merging phase may also be desirable where multiple units with data-dependancies are merged. Naturally, this reduces the amount of parallelism present and potentially reduces speed but it can also drastically cut down the amount of hardware in the implementation and the more compact unit could well be faster because of reduced routing delay. This merging phase can be done by manual intervention (perhaps using graphical tools to manipulate the ETPN)) or automatically using 'expert' systems or other heuristic techniques.

Our example of searching for the minimum of five words after adding a constant to each one is a good illustration of these optimisations. Several comparison operations can be formed into a single Boolean expression over all the five input words. The expression can be optimised at compilation time and the resulting logic function can be assigned to CLCA cells. The efficient automatic realisation of such Boolean expressions has only recently become feasible with advances in multi-level logic synthesis techniques (Brayton 1986). A sequential program for this operation would take at most two words at a time for processing, and consume several cycles for each suboperation. A parallel processor implementation would distribute partial comparisons to different processors and thus introduce a major communication overhead.

After the optimisation of the Boolean expression, we have a complete structural description of the design in terms of a net-list of logical units capable of being implemented by the primitive cells in the target array. The next steps in the process could be termed cell-assembly and consist of floorplanning, global routing and local placement and routing of functional cells. Floorplanning and global routing are high level processes applied to large hierachical structures (e.g. our five way comparitor) within the structural description. Given this high level plan detailed placement and routing within the large sub-structures and channel routing to connect them up into the final design is also required. Usually, heavy computation is needed in automatic placement and routing - techniques such as simulated annealing are often used in the floorplanning step to ensure good results. Good placement of the computational units is very important since excessive delays will result from long wires. Minimising the computation involved is important in a system like ours where frequent recompilations will occur as the program is developed. It is at this point that the advantages of the CAL architecture become apparent:

1. The architecture scales transparently over chip boundaries. Realistic size systems will never fit on a single programmable chip given the overhead of the configuration memory thus it is essential that multi-chip systems be supported. Architectures which use 'special-purpose' input-output blocks are unsuitable for large systems since single units (for example large logic blocks) in user designs will be hard to split over multiple chips.

2. The architecture is completely symmetrical: this is important when floorplanning large systems since it allows large subunits to be rotated and reflected to obtain a dense packing. Algorithms for floorplanning silicon designs take advantage of this flexibility.

3. There is a single resource in the system. Large units are built up by composing small resources rather than breaking up large ones. One area where this is particularly important is channel routing. In a large design channels with as many as 20 tracks are likely to occur: in the CAL architecture there is potentially no limit to the number of tracks in a channel, although each additional track may require an additional line of cells (often two tracks can be fitted in a single line of cells). In an architecture such as the LCA with special fixed width wiring channel resources problems occur when that width is exhausted; possibly resulting in routing failure or grossly inefficient use of resources.

4. The routing model is simple and safe. Routing in a CAL design is easy, although potentially quite slow: there is only one class of routing resource so there is no question about which is the most suitable for a given signal. All paths are fully buffered so there is no need to worry about logic levels. These factors are important because they allow the use of standard 'channel routing' algorithms which can produce high quality routing relatively quickly.

More complex architectures can still be routed automatically using 'maze' routers but the results are likely to be worse and computation time significantly longer.

At its present stage of development the CAD tools for the CAL system are fairly primitive because of the limited manpower available for their development: channel routing, logic synthesis and tools to support manual design have been written. It is intended that support for CAL should eventually be integrated into an existing silicon compiler since many existing programs for functions like floorplanning, global routing and structural language input could be used unchanged. Thus designs could proceed from a single source format into either a silicon or a configurable logic implementation. This capability is important since one of the target application areas for CAL's is in Application Specific IC prototyping (Kean 1989).

The tools that are available for application development with the commercial LCA families consist of a logic cell editor, a macro library, a simulator, an autorouter and an autoplacement utility. At present design using these tools normally requires manual intervention to solve routing problems so they are not really suitable for use as a 'cell assembler' back end to an active compiler.

5. DISCUSSION: FUTURE GATE-LEVEL DYNAMICALLY RECONFIGURABLE PROCESSORS

The techniques discussed in the last section can provide significant speedups in many operations, however, to take advantage of them significant work will be required on both the active compiler system and the architecture of the CLCA's. New programming languages closer to current hardware description languages may also be required to make full use of the configurable structure. In the context of self-timed control structures use of an extended form of OCCAM with word length specification for operations is worth considering, especially if a transputer is used as the host computer.

The CLCA has four main speed advantages over normal processors:

1. There is no instruction fetch or instruction decode. In a CLCA the 'instruction' is the configuration information and it is fixed before processing starts. This provides a considerable speed advantage at the expense of requiring much more processing hardware - if two operations need to be performed then in effect we have two ALU's rather than time sharing the same one. Naturally, this poses a severe limitation on the complexity of the code which can be implemented on CLCA's. Small sequences of 'inner-loop' instructions extracted from HLL programs or simple calculations performed in systolic algorithms are attractive for CLCA implementation.

2. The hardware performs exactly the calculation required. In most processors computation cycles are synchronised to a clock set for a 'worst-case' instruction (e.g. carry propagation in a 32 bit adder), in a CLCA simple computations take only a few gate delays. This flexibility extends to the width of the operation and the number of operands used. Often several 'ALU' operations can be merged into one complex Boolean expression and implemented directly in the CLCA. In a systolic algorithm implemented on a fixed size CLCA all the silicon can be used effectively - if the computation requires relatively few cells then more Processing Elements (PE's) can be implemented on the array. Compare this with a conventional parallel computer where the number of PE's is fixed in advance.

3. Pipelining. In many of the inner-loop and systolic computations performed by CLCA's pipelining can be implemented very naturally. This is especially effective on the CAL architecture where pipeline registers use only a single small cell and can often make use of function units in cells which would otherwise be used just for routing. Pipelining can often make up for the slow propagation delays along the programmable interconnect. Reconfigurable pipelines have difficulty in making efficient use of fixed operational units, because the different tasks implemented may require different resources along the pipeline. Solutions like sharing of operational units or adding non-compute delays have not been found to be effective (Hwang 1984). CLCA's can solve these problems by building the resources on demand.

4. Routing Flexibility. Many calculations involve manipulations which can be accomplished by wiring rather than active circuitry. Examples are swapping registers, extracting bit fields and manipulating masks. Most conventional processors perform these operations extremely inefficiently because of their long word lengths. Often a problem specific routing structure can eliminate a large amount of redundant computation.

5. The advantages of asynchronous processing have been pointed out in recent works on VLSI design, for instance by Peng (1987), emphasising that the problems in large synchronous VLSI designs with large clock skews that limit the maximum clock frequency can be solved using asynchronous structures. A research computer designed for asynchronous processing is described by Nowak (1987), showing significant speed increases with only a few processing units. Asynchronous, self timed structures could be a potential choice to the control structure of a CLCA computer.

The examples given in this paper give hints of the potential of CLCA's to speedup inner loop code. We could program sections of 29 to 37 instructions on the smallest 64-cell device with 5 bit data obtaining a speedup factor of 23 to 30 in one case and a speedup factor of 60 in the another over a reasonably powerful microprocessor.

For wider use, at least 16 bit processing, and loops with more operations will be needed. The largest available LCA has roughly ten times the capacity of the device used in this paper so the extension in the wordlength can be achieved although use of bit-serial arithmetic may be preferable to having wide arithmetic units because of routing delays in carry chains. CAL can take this even further by allowing a board full of configurable devices to be used allowing implementation of large systems. Slow programming will always be a problem although the bottleneck can be expected to move from the CLCA's themselves to the rate at which the host processor can supply the configuration data. CLCA acceleration will always be most suitable for programs which require large numbers of repetitive operations so that the configuration data can stay constant for long periods.

Two other examples of the application of CAL to carefully chosen problems in (Kean 1989, Gray 1989) are also worth mentioning. In the first a DES encryptor is built from a large (about 8000 cells) CAL array, this computation is inherently bit level and very amenable to pipelining. Using a ten stage pipeline within the critical f-box section of the DES algorithm a performance of 500,000 encryptions per second can be obtained. This is as fast as the best DES custom chips (Verbauwhede 1988): optimised DES software running on conventional processors is about 1000 times slower (mainly because the bit scrambling done using wiring channels in the hardware versions is very hard to implement efficiently on a conventional processor). In the second example CAL is used to provide the computation in a cellular-automaton model for fluid flow simulation (Salem 1986). Using a system with 26 large 64x64 cell CAL arrays providing 128 pipelined update processors supported by a large dynamic RAM memory to keep the computation unit supplied with data performance of the same order as a Connection Machine multiprocessor, reported by Wayner (1988), is predicted. These figures give an indication of the raw power of the configurable logic approach in appropriate problem domains. The DES design required careful manual layout and the fluid-flow design requires a special purpose memory system to supply the data quickly enough so this kind of performance is not to be expected from the system described here.

6. CONCLUSIONS AND FUTURE WORK

The application areas of CLCA's are obviously those, where relatively simple operations are performed over large data sets. These kinds of task are found in image processing, computer graphics, statistical simulation and finite element methods. With appropriate compiler technology we would hope to obtain speed-up factors of roughly 20 times using fairly small CLCA arrays with normal procedural programming languages. With a large manual design effort on a particular problem much greater speedups are achievable.

More work is needed in future on the development of better tools for automatic configuration program generation and new and better structures for the configurable logic cell arrays. Better methods of expressing the newly available forms of parallelism are also needed in the source languages.

7. REFERENCES

Algotronix (1988) CAL1024 Preliminary Data Sheet, Edinburgh UK, 1988.

Borgefors G (1986) On Hierarchical Edge Matching in Digital Images Using Distance Transformations, dissertation TRITA-NA-8602, The Royal Institute of Technology, Stockholm, Sweden 1986. (To appear in IEEE PAMI)

Brayton RK, Camposano R, DeMicheli G, Otten RHJM, van Eijndhoven J (1986) The Yorktown Silicon Compiler System, Technical Report RC12500, IBM T.J. Watson Research Centre, Yorktown Heights.

Davis R, Thomas D (1984) Systolic array chip matches the pace of high-speed processing. Electronic Design, October 31, 1984, pp. 207 - 218.

Fountain TJ (1983) The development of the CLIP7 image processing system, Pattern Recognition Letters 1 pp. 331 - 339, North-Holland.

Gray JP, and Kean TA (1989), Configurable Hardware: A New Paradigm for Computation, to appear in Proc. Decennial Caltech Conference on VLSI, (1989).

Hillis WD (1985) The Connection Machine, MIT Press, Cambridge, Mass., U.S.A.

Hwang K, and Briggs FA (1984) Computer architecture and parallel processing, McGraw-Hill, New York, U.S.A. pp. 208...212.

Kean T (1989) Configurable Logic: A Dynamically Programmable Cellular Architecture and its VLSI Implementation, PhD Thesis, University of Edinburgh, Dept. of Computer Science.

Nowak L (1987) SAMP: A General Purpose Processor Based on a self-Timed VLIW Structure, Proc. 1987 Conference on Architectural Support for Programming Languages, pp. 32-39.

Peng Z (1987) A Formal Methodology for Automated Synthesis of VLSI Systems, Linkoping Studies in Science and Technology, Dissertation No 170, Dept. of Computer and Information Science, Linkoping University, S-58183 Linkoping, Sweden.

Salem JB, Wolfram S (1986) Thermodynamics and Hydrodynamics with Cellular Automata. Theory and Applications of Cellular Automata, Paper 3.10, pp 362--365. World Scientific Publishing Co., Singapore.

Sternberg SR (1985) Computer architectures specialised for mathematical morphology, in "Algorithmically Specialized Parallel Computers", ed. by L.Snyder et al. Academic Press, pp. 169 - 176.

Thomas D, Hitchcock III C, Kowalski T, Rajan J, Walker R (1983) Automatic Data Path Synthesis, IEEE Computer, December, pp. 59 - 79.

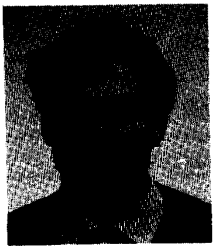
Verbauwhede I, Hoornaert F, Vandewalle J (1988) Security and Performance Optimisation of a new DES Data Encryption Chip. IEEE Journal of Solid State Circuits, Vol 23, pp 647--656, June 1988.

Viitanen J, Hanninen P, Saarela R, Saarinen J (1987) Hierarchical pattern matching with an efficient method for estimating rotations. Proc. of the IEEE Industrial Electronics Society conference IECON'87, Cambridge, Massachusetts, U.S.A. 3-6 November.

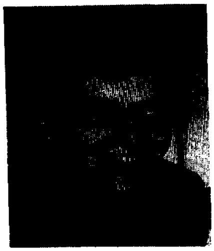
Viitanen J, Hanninen P, Saarela R, Saarinen J (1988) An Efficient Method for Image Pattern Matching, Proc. of The International Conference on Parallel Processing for Computer Vision And Display, Leeds, U.K. 12-15 January 1988.

Wayner P (1988) Modelling Chaos, Byte, May 1988, pp. 253 - 258.

Xilinx (1986) The Programmable Gate Array Design Handbook, Xilinx Inc. U.S.A. 1986.



Jouko O. Viitanen
Was born in Orivesi, Finland, on October 13, 1954. He received the M.S. degree in electronics in 1978, and the Lic.Tech. degree in computer science in 1984 from Tampere University of Technology. He is currently a Senior Researcher at the Research Institute for Information Technology, Tampere University of Technology. His research interests are image processing, image processor and computer architecture.



Tom Kean has recently completed his Doctoral thesis within the Computer Science Department of Edinburgh University.
His research interests are in dynamically programmable logic arrays, VLSI design and silicon compilation. He received the BSc degree in computer science with first class honours from Edinburgh University in 1985.

Present address:
SARI Project,
Department of Electrical Enginnering,
University of Edinburgh,
The King's Buildings,
Mayfield Road,
Edinburgh, UK.

R.A. Earnshaw, B. Wyvill (Eds.)

New Advances in Computer Graphics

Proceedings of CG International '89

With 375 Figures Including 126 in Colour

Springer-Verlag
Tokyo Berlin Heidelberg New York London Paris