AUTOMATIC SYSTOLIC ALGORITHM DESIGN II
(A Practical Approach)

G M MEGSON

Rapporteur: P Ezhilchelvan
D Comish

AUTOMATING SYSTOLIC ALGORITHM DESIGN II : (A Practical Approach)

G.M. Megson
Computing Laboratory
University of Newcastle-Upon-Tyne
Claremont Tower, Claremont Rd,
Newcastle-Upon-Tyne
NE1 7RU
U.K.

1.0 Introduction :

In the past systolic design has largely been conducted using the 'seat-of-the-pants' method. This method can be stated simply as follows :- you sit down with a pen and a piece of paper and sketch out some arrays and try various draft data flows until you eventually find a design that works or (which is more frequent) give up and decide that perhaps a systolic design is not the best approach. Part I of these tutorial presentations [1] has suggested a better design method which should be used to direct a designers effort more effectively. Unfortunately the underlying algorithms of the method involve substantial computational effort (e.g simplex method for the timing schedule, givens rotations for projections, and n dimensional convex hull for determining the directed half-spaces and extremal rays of the domain) and is not well suited to manual application. Indeed the task can be extremely tedious and error-prone even for a relatively straightforward design.

The theme in this paper is concerned with the development of environments for systolic algorithm design and the kind of computer aided design (CAD) tools which are required to support the design process. In particular we shall consider the DTAGS transformation kernal which forms part of a Systolic Algorithm Design Environment (SADE) being developed at Newcastle [2]. Before proceeding however, it is worthwhile devoting a few sentences to the general motivation for CAD tools. An observation by David Hilbert [3] used in the preface of a book on geometric modelling [4] seems quite appropriate here

> *On the one hand, the tendency toward abstraction seeks to crystallize the logical relations inherent in the maze of material that is being studied, and to correlate the material in a systematic and orderly manner. On the other hand, the tendency toward intuitive understanding fosters a more immediate grasp of objects one studies, a live rapport with them, so to speak, which stresses the concrete meaning of their relations.*

The goal of systolic synthesis is to crystallize and unify the different ad-hoc designs existing in the literature by a single mathematical framework. But at the same time one cannot deny the natural affinity that humankind seem to possess

this work is supported by SERC grant GR/F 80494

for a intuitive understanding of geometrical forms. Indeed my experiences with DTAGS and in the wider context of systolic design have afforded deep and often unexpected insights in to the nature of parallel algorithm design - this rapport is something that is to be valued greatly. Hilbert's remark is particularly pertinent when one considers the capabilities of modern day high performance workstations for performing interactive vizualization and manipulation of complex data models.

It is also important to be clear about what we mean by design. Design is simply the process of creating, describing, and then selecting a form which must fulfil some function. More precisely we are interested in engineering design - an interactive process defined by brief periods of insight and creativity followed by longer more tedious periods of refinement, selection, and modification until the product meets a set of constraints (or objective functions). In systolic design this means producing a systolic array for a particular algorithm and selecting the best array interms of number of cells, cell complexity, input-output connections, time complexity and so on. The final design may not be the one that satisfies theoretical models of optimality but rather one that interfaces readily with an existing system or product range. For the more commercially orientated reader the over-riding concern might be that the final product is the least expensive to manufacture, or is the most robust, or fault tolerant design available.

2.0 Environments and CAD Tools :

Systolic design is about *algorithmic engineering* [5]. The raw materials for the design process are sequential algorithms and the output is some description of a systolic array for performing that sequential computation. To get from the raw input to the final output requires the application of engineering design principles to transform the algorithm in a way that exposes the inherent parallelism and allows exploitation of current technology (such as VLSI techniques, programmable parallel architectures etc). The core (or kernal) of any design environment must be a set of routines that simplifies the computationally demanding parts of the design process. In a broader sense the environment must also provide basic infrastructure such as array generation, interactive features to allow manipulation of the design, and database facilities to build up a repository of common design objects. The key feature of the system is a single computational strategy which is independent of any special features inherent in the design objects. In the current context the control strategy is the synthesis procedure outlined in [6].

What kind of operations do users want in a systolic modelling system? Table I surveys a number of existing software tools that have made some attempt to provide CAD facilities, the list is not exhaustive and continues to grow. However it must be emphasized that many of the tools now appearing have features in common. For simplicity the middle column of the table identifies a few distinguishing features for each package. The right column lists a single feature that can be considered essential for a design environment.

NAME	CHARACTERISTICS	FEATURE
DIASTOL (Quinton et al 1986)	Data Dependence Mapping, Uniformization, Tupling, Parametrization	Linear Schedules, Optimization
ADVIS (Moldovan 1987)	Partitioning, Fixed Sized Arrays, Defined Connections	Restricted Topology
SDEF (cappello 1987)	Directed Acyclic Graphs, File Definition, Code generators	Code Generation
VACS (kung et al 1990)	Systematic searching of projection/timing functions	Constraint Matching
DECOMP (vehilies 1990)	Sequential programs, Internal cell organisation, Signal processing	Full Mapping
HIFI (De Lange 1991)	Graphics Interface, Tiling, VLSI layout	Graphics

Table I : Selection of CAD tools for systolic design

In a complete system the user requires facilities which allow for rapid modelling and prototyping of designs. The term modelling is used because although synthesis is used as a formal model of the design process more heuristic methods that rely on the designers intuition are required, this is especially true when the synthesis method breaks down (for example on non-constant data dependencies and unbounded domains). We are emphasizing engineering design so formulating constraints and systematic searches of the timing and projection functions are desirable. Unfortunately the search space can be quite large even for small designs so it is often necessary for the designer to prune the search manually - this is best achieved with the aid of some graphical display of the algorithm geometry. Ideally, a full environment should produce a full mapping, that is input as sequential code and output as a description of an array. If the array is programmable we will want to produce a parallel program suitable for an available (and restricted) topology. For a 'pure' systolic algorithm (represented at the bit level) we will want to produce a suitable circuit description such as VHDL which can form the input to a silicon compiler.

Figure 1 gives an global view of a prototype Systolic Algorithm Design Environment (SADE) which incorporates the ideas outlined above. In addition it adds a few novel features which are particularly useful for supporting on-going

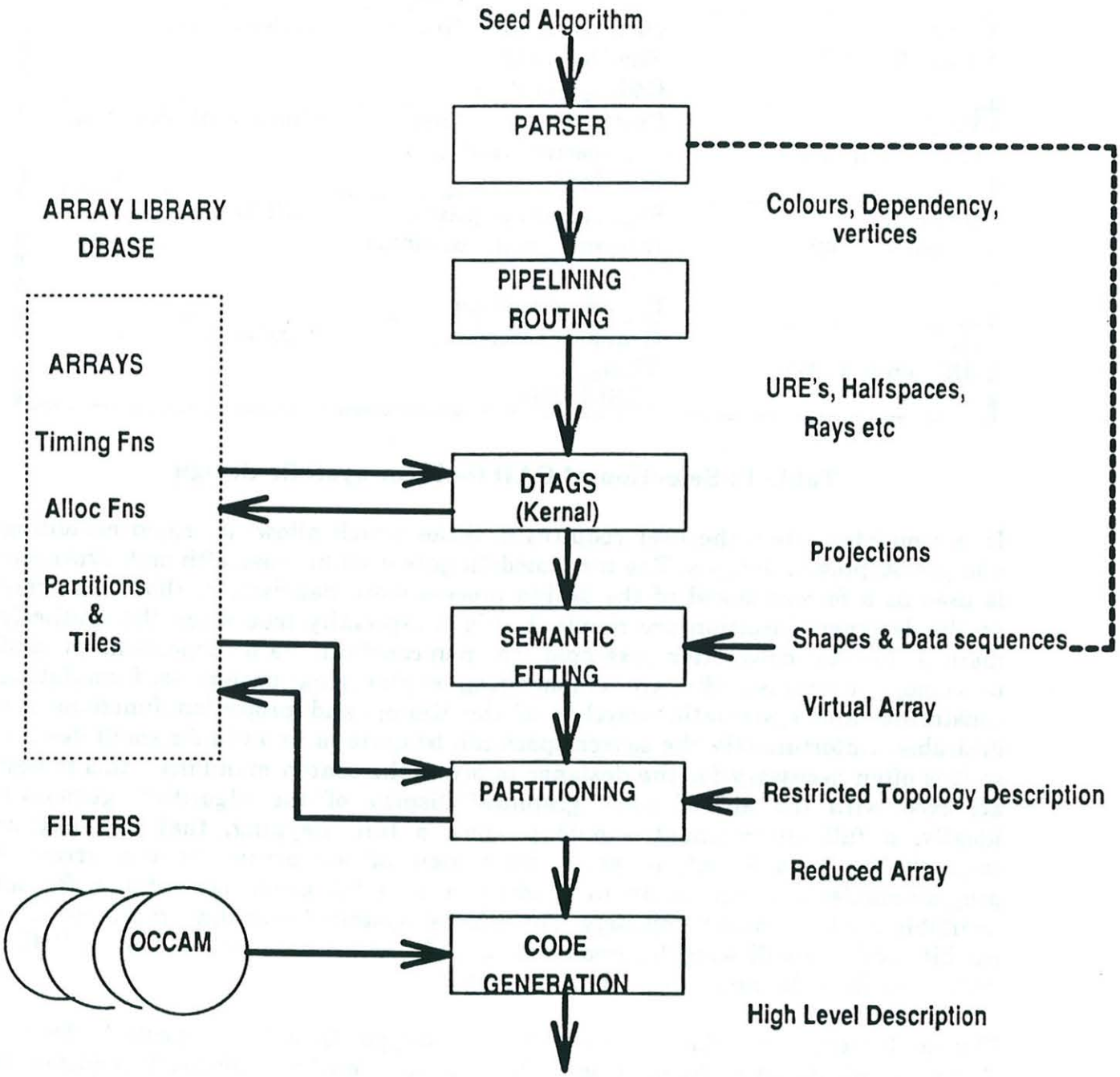


Figure 1 : A Systolic Algorithm Design Environment

research activities. The core of the system is the Dependency Timing Allocation Graphical Snapshots (DTAGS) kernal which performs transformations derived from synthesis or specified manually by the designer. DTAGS provides all the data structures necessary for manipulating URE's, timing, allocation, and data dependency structures (a more detailed description is given in section 3). Input can be made directly to DTAGS but it is more useful to have extra facilities which simplify the construction of a canonical form. For example a cut-down pascal parser is used to derive data dependency and domain vertex information from simple for-loop programs. Data movement is captured by the assignment of 'colours' to program variables, similarly operations at the dependence graph nodes are given a 'shape'. Non-linear dependencies can also be detected at this stage and the system is fully indexed. Given the dependencies and vertices the system can be re-written by applying pipelining and routing to yield a system of uniform recurrence equations. A large choice of pipelining vectors exist and so the designer can take the default assignment produced by the package or select their own (which are checked for consistency). The router also derives the half spaces for the domain by computing the n-dimensional convex hull of the vertex points. At this stage it is possible to generate sub-domains for each 'shape' in the design. The rays associated with the domain faces can also be easily evaluated.

The output of DTAGS is a graphical description of the array together with a schedule of the shapes allocated to each cell. This description can be used as an intermediate form to generate code for different underlying machine structures. In a full environment a set of software filters (e.g Occam - transputers, LINDA - Encore Multimax, STRICT, VHDL - silicon compilers) are envisaged. The DTAGS output is problem size dependent consequently any code generated (unless it maps onto the available architecture naturally) will not produce efficient code. Consequently the intermediate form has to be manipulated to produce an alternative reduced array which maps directly onto a restricted topology. Finally, an array database is used to store lists of possible designs, timing and allocation functions, and partitioning rules. The user can build up a working set of designs and orientate the library towards their own particular application area and hence reduce design time.

3.0 DTAGS :

The main purpose of this paper is to illustrate how DTAGS can be used for design. Figure 2 shows a screen dump of the kernal system (at this stage not much effort has been devoted to the interface). The top window identifies the environment module and summarizes the state of the package - at the moment we have three designs in the system and are working on the third one. For this design there are four timing functions and we are using the first. Similarly there are three allocations available (at the moment) and we are using the second one. The last item indicates that three graphs have been generated (possibly one data dependency and two allocations) and we are using the third.

The second window displays the DTAGS options. The *design* option allows the

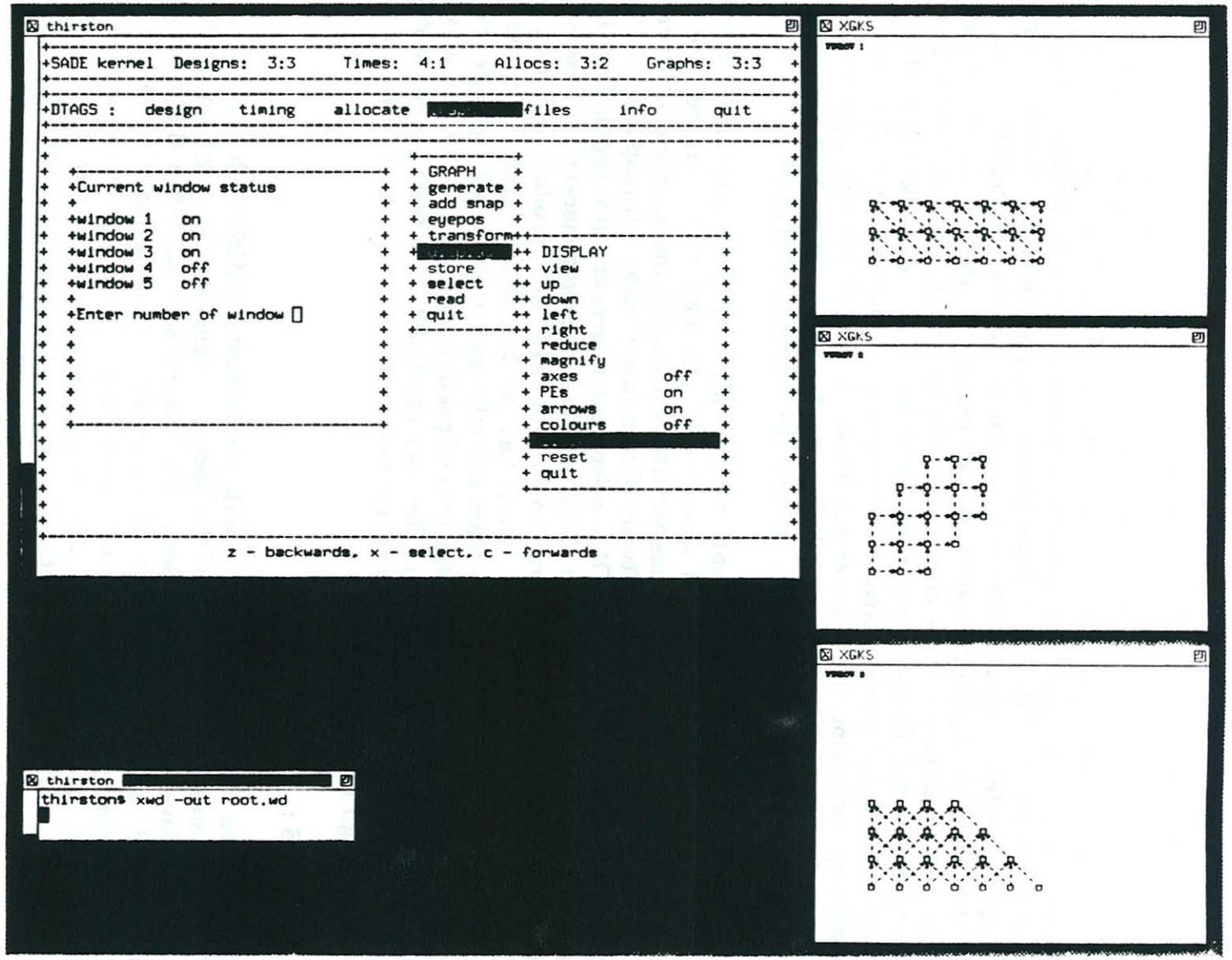


Figure 2 : DTAGS (Kernal)

creation of a design manually (by typing the dependencies, and half-spaces etc), or the reading of a file (possibly generated by the pre-processing elements of the environment). The *timing* option allows the reading of a library file containing default timing functions, the manual entry of a timing function, or construction of the timing function by solving a linear programming (LP) problem. The LP problem is generated and solved automatically (if no vertices or rays have been specified they are generated - assuming a hypercube domain, and rays that pass through the origin). The user can control the LP by selecting just the optimal or a list of feasible timing functions. Alternatively, the designer can structure the timing function by restricting the scope of the coefficients (to non-negative, zero, or free variables). The *allocate* option builds a list of possible allocation matrices. The designer has limited control and can specify an arbitrary direction, or choose from a default list of allocations, or ask for a direction orthogonal to the timing function. The *files* option gives a current list of the files being used to store the design, similarly the *info* option gives a brief summary of the current design (data dependencies, timing functions, etc).

In the Figure the option selected is *graph*, this section allows manipulation of data dependency and projected graphs and provides a wide range of options. The most important options are *generate* and *add snap*, the first produces a data dependency graph from the description, the second simulates the array operation. For three or less dimensions both options produce graphical output which can be recovered using the display option - for higher dimensions a file dump describing the array or it's operation is possible. The key insight into a complex design occurs when the dependency graph is manipulated in some way. The *transform* and *eyepos* menu options provide these facilities. In the former various geometric manipulations such as scale, rotate, shear etc can be applied either before (pre) or after (post) projection. One use of this feature is to orientate two separate domains so that they can be pipelined together. In the latter the designer can specify the position from which they want to view the graph. The resulting transformation is used as an allocation (and on request is automatically inserted into the allocation list). The remaining options allow storage and retrieval of lists describing designs and are available in all the other menu options.

Finally let us briefly consider the *display* section. As we would expect all the facilities for manipulating the graphs on screen are available. In addition options are also provided for clarifying complex designs. For example, the processing elements and arrows can be switched off (in practice they often make the displayed graph look overly complicated). Alternatively individual colours can be selected or masked so that the designer can concentrate on choosing projections to make the data flow of individual variables stationary or non-stationary. Different designs can be given a separate window for the display so that the user can switch easily from one design to another. For example in Figure 1 three windows are open. The first is a Data dependency graph for the convolution problem. The second is a projection of the banded matrix product problem (and so is a possible systolic array) and the third is a Data dependency graph for Aitkens method which we discuss shortly. A straightforward window dump can be used to produce good

quality output for inclusion in papers, reports, or class exercises. Notice that each of the windows is for a different algorithm, associated with each of these algorithms are corresponding lists of timing and allocation functions. We conclude that DTAGS is a flexible system for interactive algorithm design.

4.0 Aitkens Method (an example) :

One of the most frustrating features of current literature on systolic synthesis is the tendency to concentrate on only one or two simple examples (e.g matrix product, convolution, matrix triangularisation). Indeed these designs are so simple that algorithm designers can easily produce the arrays by ad-hoc methods. Indeed that is how the first arrays were developed before the emergence of basic synthesis techniques. Clearly the test of any *real* and useful design tool is its ability to cope with complex designs which are at the limits of an experienced designers ability - it is at this level that a tool provides real insight into algorithm design and becomes useful. To illustrate some of the capabilities of DTAGS we will derive some systolic algorithms for Aitken's method an algorithm for determining the roots of transcendental equations.

More formally we want to compute the roots of $K \geq 1$ polynomials of the form

$$f_k(z) = 1 + a_{1,k}z + a_{2,k}z^2 + \cdots \tag{4.1}$$

for $k = 1(1)K$. Aitken's method solves a single equation of (4.1) by computing a doubly infinite table of so-called Hankel determinants using the recurrence

$$H_{i,j-1,k}H_{i,j+1,k} = H_{i+1,j,k}H_{i-1,j,k} - H_{i,j,k}^2 \tag{4.2}$$

it is known that $H_{i,0,k} = 1$ and $H_{i,1,k} = c_{v,k}$ where

$$h_k(z) = g_k(z)/f_k(z) = 1 + c_{1,k} + c_2z^2 + \cdots \tag{4.3}$$

$$g_k(z) = 1 + b_{1,k}z + b_{2,k}z^2 + \cdots$$

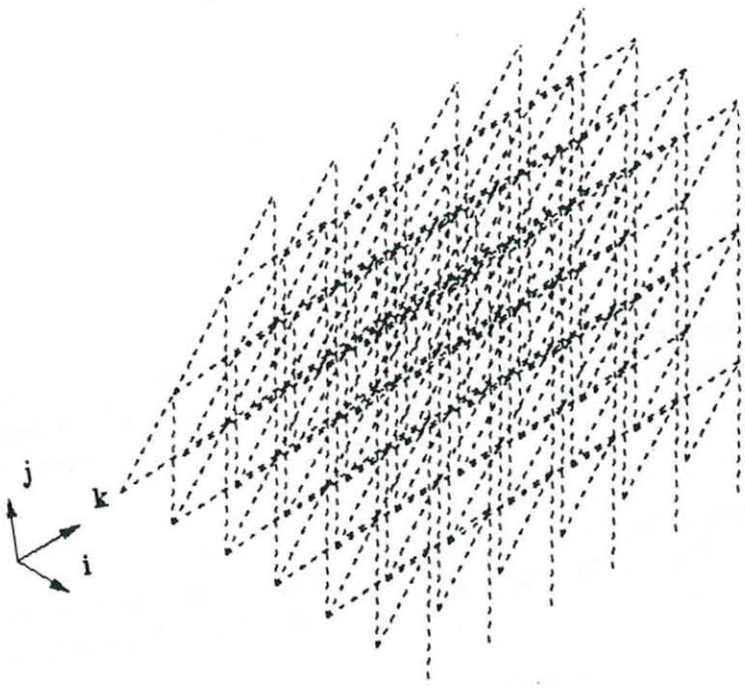
so that the first two columns of the table are given for each k. Similarly it can be shown that $H_{0,j,k}$ contains only one non-vanishing term. Infact for any column $j > 0$ the ratio $H_{i,j,k}/H_{i+1,j,k}$ has a limit that satisfies

$$\lim\ H_{i,j,k}/H_{i+1,j,k} \rightarrow r_{1,k}r_{2,k} \cdots r_{j,k} \tag{4.4}$$

where the $r_{j,k}$ are the zeroes of $f_k(z)$. Furthermore if $f_k(z)$ has any other zeroes they are strictly greater in modulus. Consequently, if we know two adjacent columns j and $j+1$ of table k we can determine $r_{j+1,k}$ using a simple ratio.

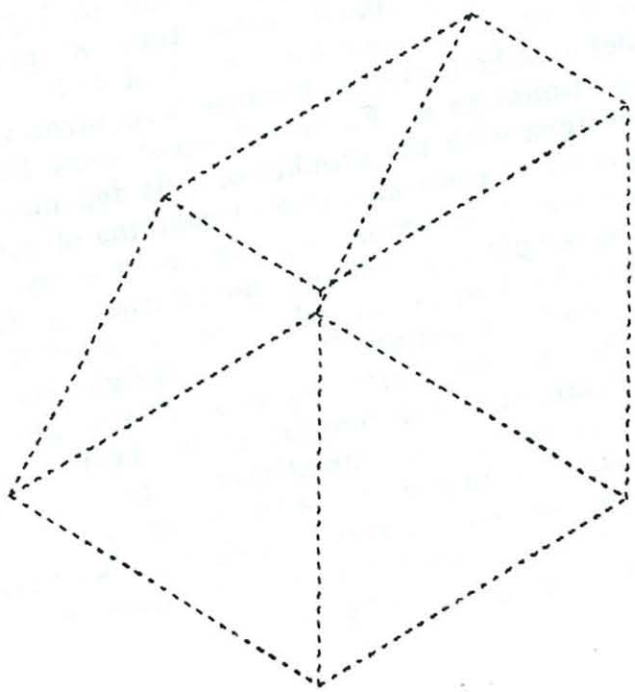
Now, infinite dimensions produce unbounded domains which compromise the design method by a) preventing the formulation and solution of the associated timing function LP and b) producing arrays with infinite dimensions. If just one dimension is finite then a feasible array can be produced by projecting in this direction. Fortunately we can bound the table by computing block wise (see [7]). For example suppose we define the block r,s of table k as $B_{r,s,k}$. Further suppose that the block is a trapezoid containing the elements

WINDOW 1



a) Dependency Graph

WINDOW 1



b) Domain .

Figure 3 : Dependency and Domains for Aitken's method

$$H_{m(r-1)+i,n(s-1)+j,k}$$

where $j=1(1)n$, $i=1(1)m+n-j$, and $k=1(1)K$, and $m,n>0$. The blocks can be evaluated in the order defined by the non-linear timing schedule

$$T(B_{r,s})=s+(r+s-2)(r+s-1)/2 \tag{4.5}$$

and the individual blocks can be treated as finite problems with three-dimensional domains (i.e. i, j, k) of bounded size.

Ideally, we would like the construction of the table associated with problem $k+1$ to be pipelined behind problem k as soon as possible (i.e we want to pipeline (4.2) in the k direction). This may be achieved by applying a shear to skew the domain in the k direction and to transform (4.2) into the form

$$H_{i,j+1,k}=(H_{i+1,j,k-1}H_{i-1,j,k-1}-H^2_{i,j,k-1})/H_{i,j-1,k-2} \tag{4.6}$$

The resulting data dependency graph is shown in Figure 3a and the convex-hull of the domain is given in Figure 3b. Solving the associated LP problem derives an optimal timing function as

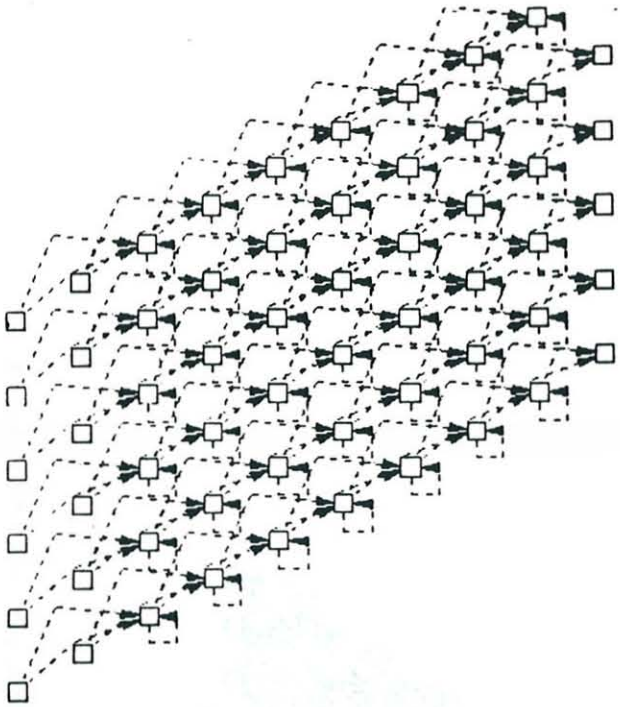
$$t(i,j,k)=j+k-1 \tag{4.7a}$$

which corresponds to evaluating a single column of table k at each time step, alternatively the following function

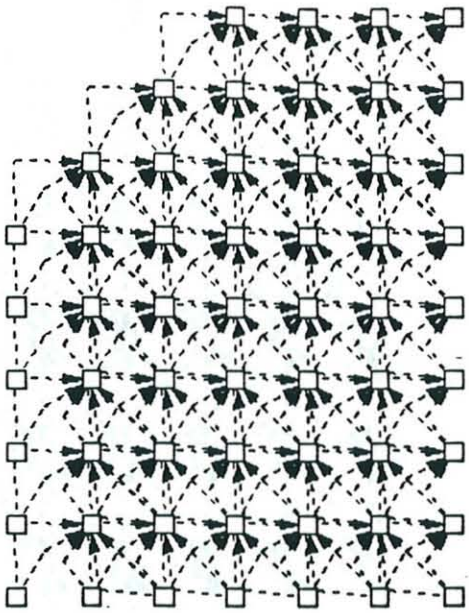
$$t(i,j,k)=i+2j+k-1 \tag{4.7b}$$

is also feasible and describes a wavefront method (it can be entered manually). In the former, any projections with a direction vector of the form $(\pm1,0,0)$ are perpendicular to the time axis and are not allowed. Similarly direction vectors of the form $(\pm1,-+2,\pm1)$ cannot be used for (4.7b).

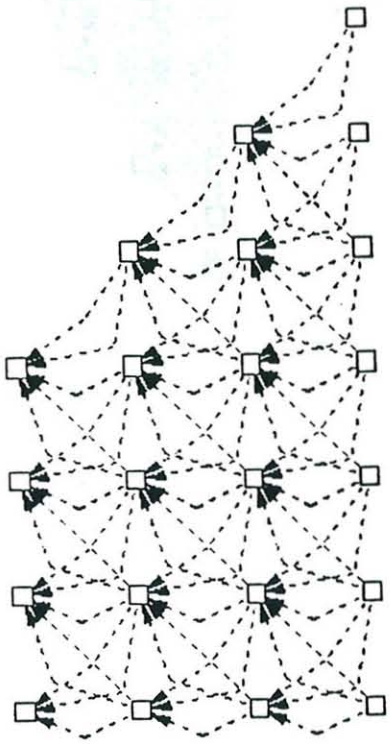
Figure 4 shows some projections of the domain in Figure 3 along with their direction vectors. Figures 8 (b,c,f) show that K problems can be solved simultaneously by using K independent copies of a 1-D array that evaluates a single table. Fig (8c) is to be preferred because it requires only uni-directional data flow and cells proportional to n. Figures 8 (d,e,f) show hybrid projections which produce pipelined designs with the number of cells dependent on the problem size. Figure 8d requires $m+n-1$ rows and $n+k-1$ columns of cells. Figure 8e requires $(m+2n-2)$ rows and $k+(n-1)$ columns, and Figure 8g n rows and $m+n+k-2$ cells. In contrast Figure 8a employs completely non-stationary data flow and requires $n(m+n-1)$ cells (i.e $O(n^2)$ when $m=n$), which is independent of K, and all input occurs at the boundary of the design. The other designs require some preloading and unloading of data at the start and end of computation which demands additional control mechanisms that complicate the basic cell implementation so Figure 8a is to be preferred as a pipelined architecture. We will not discuss the fine details of array operation here because the emphasis is on the use of DTAGS to explore systolic design options (the interested reader is referred to [7] where the designs are fully discussed, alternatively the more enthusiastic reader is encouraged to construct the projections by hand !!).



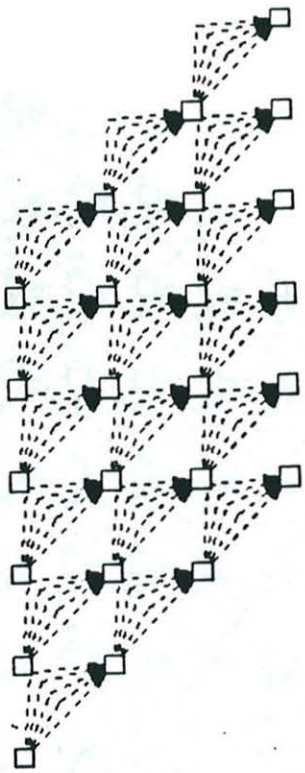
b) $[-1,-1,-1]^t$



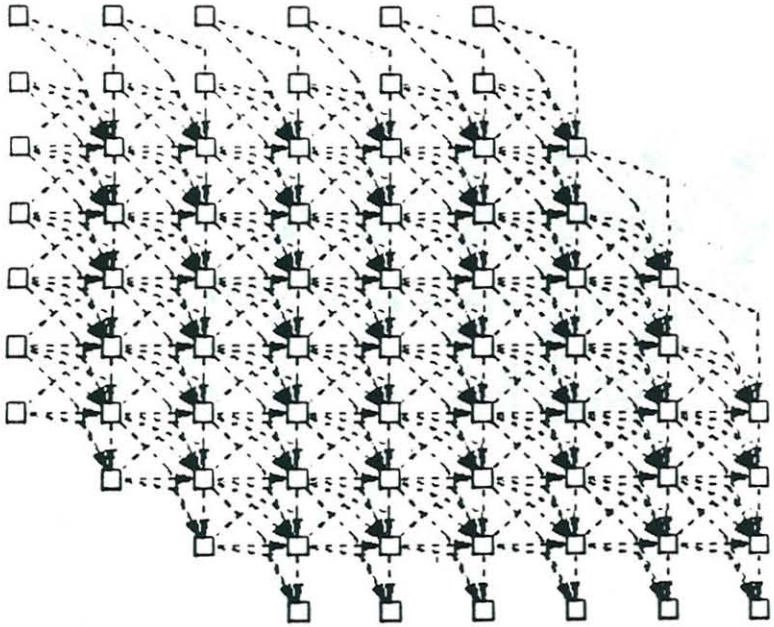
d) $[0,-1,0]^t$



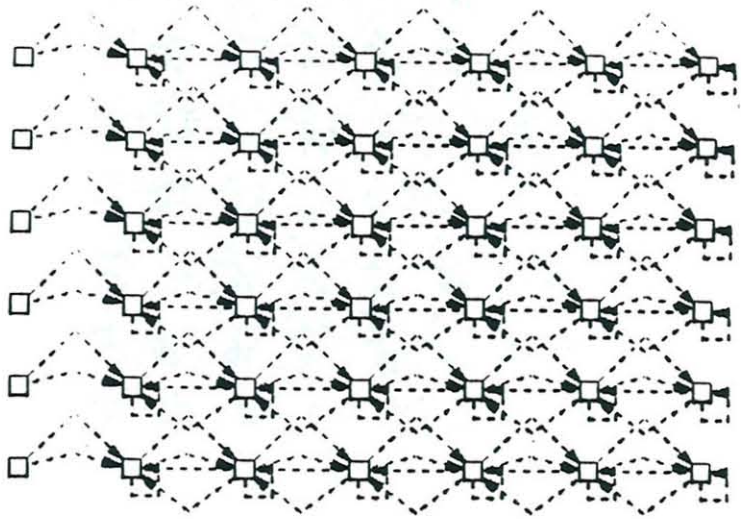
a) $[0,0,-1]^t$



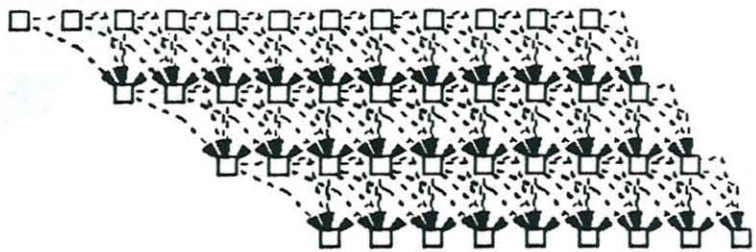
c) $[-1,0,0]^t$



e) $[-1,-1,0]^t$



f) $[0,-1,-1]^t$



g) $[-1,0,-1]^t$

FIGURE 4 : 2-D Arrays derived from projections of 3-D Dependencies $k=6, m=4, n=4$

Using DTAGS the designer needs to know very little about the design process. For direct input to the kernal a user has to be competent in basic synthesis techniques as described in part I of these tutorials so that dependencies and domain half-spaces can be produced. Alternatively this stage can be performed automatically using the parser and router tools. In the latter case all that is required is the formulation of a block computation using a three-loop sequential program. The domain and data dependency graph is produced automatically as can the optimal time function, and default allocations associated with the axes of the space. Thus the role of the designer is reduced to trying novel allocations and timing functions (which can be motivated by experiments using the default values or a more intuitive approach) and assessing the designs produced interms of the data flow and cell count. For example it is interesting to note that Figure 8c which is the most amenable to fault tolerant manufacture cannot use the optimal timing function. If the designer really wants to use this array the alternative timing function must be used. Another interesting fact is that the best pipelined design (Figure 8a) is infact the data dependency graph for the evaluation of a single block of the hankel table (i.e. K=1), so that 1-D arrays can be found by suitable projections.

5.0 Summary :

To conclude, systolic designs for a relatively complex problem were generated in a straightforward manner with the aid of DTAGS and the SADE environment. To use the system one needs only to know how to formulate a for-loop program for the problem and the have an intuitive understanding of the design process. In our example the 3-D data dependency graph and domain are non-trivial so that manual application of the design methodology would be extremely tedious. Indeed it is not at all obvious what types of arrays are possible. By a systematic search of projections we found a uni-directional design suited to fault-tolerant implementation, and a 2-D design for solving K problems on an architecture dependent only on the block size of the table partitioning (which is arbitrary). In addition we also determined a case when the optimal timing function could not be employed. All this work took approximately two hours. In comparison an ad-hoc (and practised) designer would require at least a week to enumerate all the possibilities by hand.

Although the DTAGS kernal is usable and fairly complete significant work still remains on the basic infra-structure of the the environment. The SADE we have proposed has been designed in a modular way so that extensions and upgrades to the synthesis techniques can be easily incorporated. For example, a systematic search of all possible timing and allocation pairs is perfectly feasible - it simply requires the traversal of lists built by DTAGS. Unfortunately, generating the number of possible pairs is a time-consuming combinatorial problem so at the moment the task is left to the user. Other metrics such as number of cells, number of inputs-outputs to cells, or generating arrays with stationary data flow for specific 'colours' are all practical and easy to implement. Perhaps more interesting and certainly more challenging is the automatic generation of code for

commercially available parallel machines. The current version of SADE is implemented in C with system dependent features such as the use of X windows and GKS graphics confined to well defined areas of the software. Development work so far has been carried out mainly on a SUN-4 workstation. Our intention is to at eventually port the software to IBM PC's.

Finally, a few remarks about the SADE philosophy. The SADE project places emphasis on *Systolic modelling*, that is, representing computations as parallel algorithms in the systolic paradigm. The back-bone of the design procedure is systolic synthesis but synthesis is not the whole design system. For example we can experiment freely with alternative designs where the current synthesis techniques break-down. The SADE is flexible enough to allow the easy incorporation of new synthesis techniques but also allows more heuristic and intuitive design approaches which enable the designer to venture beyond existing synthesis theory. Perhaps more significantly it is a software tool that is currently being used practically to develop new algorithms.

References :

[1] Megson G.M., 1991, Automating systolic algorithm design I : (basic synthesis techniques), this proceedings.
[2] Megson G.M., Comish D., 1991, "Systolic Algorithm Design Environments", 2nd International Seminar on the design and application of parallel digital processors, IEE Pub 334, pp100-104.
[3] Hilbert D., Cohn-Vossen S., 1952, "Geometry and the Imagination", Chelsea, New York.
[4] Mortenson M.E., 1985, "Geometric Modelling", John Wiley & Sons.
[5] McWhirter J.G., "Algorithmic Engineering in digital signal processing", 2nd International Seminar on the design and application of parallel digital processors, IEE Pub 334, pp11-18.
[6] Quinton P., Van Dongen V., 1989, "The mapping of Linear Recurrence Equations on regular Arrays", J. VLSI Signal Processing, 1, pp95-113.
[7] Megson G.M., Brudaru O., Comish D., 1991, "Systolic designs for Aitken's root finding method", to appear Parallel Computing.

DISCUSSION

Rapporteurs: Paul Ezhilchelvan and David Comish

Lecture One

Professor P A Lee started the discussion by asking whether the designs produced by the synthesis techniques could be mapped onto other types of architectures as well as directly onto silicon. The speaker replied that there was no reason why a systolic algorithm derived using the synthesis techniques could not be mapped onto different paralllel architectures, such as a multiprocessor. Indeed if the pipelining and routing stages of the techniques were omitted during the design process, producing a semi-systolic design, possibly a better design for the target architecture could be derived.

Next Professor B Randell inquired whether these techniques had been used to derive systolic designs to solve Digital Signal Processing problems. The speaker responded that there existed many systolic algorithms relating to Digital Signal Processing problems, such as the ones being used at RSRE (Royal Signal and Radar Establishment) to perform Kalman filtering.

Professor D Swierstra asked whether any system of recurrence equations could be mapped into a system of UREs (Uniform Recurrence Equations). The speaker replied that not all recurrence equations could yet be mapped into UREs and cited the Knapsack problem as an example of this.

Professor P A Lee concluded the discussion by inquiring whether any algorithms which have not fitted into the technology, have then led to the discovery of better algorithms after studying the derived systolic algorithms. The reply was to confirm that this has indeed happened and cited the example of the dynamic programming problem solved by P Quinton.

Lecture Two

Initially Professor J Gurd raised several issues. The first was whether the tool gave any insights into the design of an array which were not immediately apparent to the designer working by hand. The speaker responded that this was indeed the case and added that when using the tool to design a systolic array to realise Aitken's algorithm some of the resulting arrays were far from obvious.

Professor Gurd then added that presumably the design process should work even if the original problem domain was in four dimensions. Also that if the original problem was too complex for the design process to cope with, it could be broken down into component parts which could be solved separately then integrated after the design process. The speaker agreed with this but warned that this led to problems with adding control logic to the different cells in the systolic array.

Dr K Wright asked if an execution of the original algorithm involves an attempt to divide by zero, would the corresponding systolic algorithm still work. The reply was that if the original algorithm did not work then the derived systolic algorithm would not do either. Professor B Randell added that this could probably be

overcome by representing the exception as a Boolean Algebra expression and thus integrating it into the original algorithm.