# GENERALIZED METHODOLOGY FOR ARRAY PROCESSOR DESIGN OF REAL-TIME SYSTEMS

*F. Moelaert El-Hadidy and O.E. Herrmann*

University of Twente, Laboratory for Network Theory
P.O. Box 217, 7500 AE Enschede, The Netherlands
Tel: +31-53-892822, Fax: +31-53-340045
E-mail: ferial@nt.el.utwente.nl

## ABSTRACT

*Many techniques and design tools have been developed for mapping algorithms to array processors. Linear mapping is usually used for regular algorithms. Large and complex problems are not regular by nature and regularization may cause a computational overhead which prevents the ability to meet real-time deadlines. In this paper, a systematic design methodology for mapping partially-regular as well as regular Dependence Graphs is presented. In this approach the set of all optimal solutions is generated under the given constraints. Due to nature of the problem and the tight timing constraints of real-time systems the set of alternative solutions is limited. An image processing example is discussed.*

## 1. INTRODUCTION

Array processors are well suited to efficiently implement a major class of signal processing algorithms due to their parallelism and regular data flow [KUN88]. A widely used approach for mapping algorithms to array processors is the Dependence Graph (*DG*) methodology. In this methodology, first an algorithm is developed in Single Assignment Code (*SAC*) where each variable is only allowed to have a single value. Then the algorithm is represented in a graphical form by a *DG* [KUN88]. The nodes of the *DG* are then mapped into an array processor. In literature, several techniques and software packages have been reported for the automation of the mapping (see e.g. [QUI84], [MOL87], [RAO88], [ANN88], and [JAY91a]). Except for [JAY91a], only mapping of regular *DG*'s has been fully automated. The *DG*'s for large and complex problems are not regular in general and are very difficult to make regular by adding dummy operations.

In an earlier paper [MOE92], based on work done in [JAY91b], we presented an Integer Linear Programming (*ILP*) formulation for mapping (semi-)regular *DG*'s to array processors. In this paper, we use a branch-and-bound technique to obtain the set of optimal solutions for scheduling and projection. Our aim is to design a powerful methodology for systematically mapping $M$-dimensional *DG*'s directly into $K$-dimensional array processors for real-time systems, where $M > K$. This method offers a flexible platform to investigate the possible mapping alternatives under a given set of constraints. Emerging from the fact that real-time systems have tight timing requirements, we show that the set of alternative solutions is limited. Further, we show that under a certain set of constraints the solution set is independent of the problem size.

In Sections 3 and 4 the branch-and-bound approach for the scheduling and projection problem is presented. Further, complexity issues are discussed in Section 5 and mapping to fixed size arrays is discussed in Section 6.

## 2. HIERARCHICAL DG REPRESENTATION

The best way to manage the complexity of large systems is to adopt a hierarchically structured design. In literature, a hierarchical design environment has been treated in [KUN84], [ANN88], [THI88] and [JAY91a]. Here we adopt the hierarchical form of the *SAC* proposed in [JAY91a]. This form is referred to as *Structured Single Assignment Code* ($S^2AC$). The graphical representation of the $S^2AC$ description is called *Structured Dependence Graph* (*SDG*). The canonical forms of the $S^2AC$ and *SDG* are used for the construction of the *DG* with local-dependence edges in a minimum dimension Euclidean space to keep projection simple.

An index point in a *DG* can, in general, contain a set of variables whose computations are dependent on variables from neighboring as well as same index point (multi-variable *DG*'s). Single-variable *DG* nodes can be linearly scheduled which is not always the case for multi-variable nodes [RAO88]. Yet most systematic methodologies proposed in literature treat multi-variable nodes as single-variable nodes. Here the *SDG* is used to model the single-variable as well as multi-variable *DG*.

0-7803-2440-4/94/ $4.00 

## 3. ALTERNATIVE NODE SCHEDULES

For mapping regular iterative algorithms the systolic schedule is used, represented by the schedule vector $\vec{s}$. A systolic schedule implies that there is at least one delay on each edge of the resulting array processor. In semi-regular arrays however, the best schedule is not necessarily lying along a linear path. Therefore a more efficient approach has to be derived.

We define a $DG$ as a directed graph $G = \{V, E\}$ where $V$ is the set of nodes and $E$ is the set of directed edges. The set of nodes $V = I \cup N \cup O$ contains input nodes $I$, output nodes $O$, and intermediate nodes $N$. The feasibility of a schedule is determined by the partial ordering and process assignment scheme. A node should have valid data on all its input edges before it can be scheduled. Given the earliest schedule of all nodes $n^j \in I$, the earliest schedule time of any node $n^j \in N \cup O$ can be found. The latest schedule time of nodes $n^j \in O$ is also known since the system must meet a set of deadlines which imposes that the output must be available before a specific time. The latest schedule of all nodes $n^j \in N \cup I$ can thus be calculated. Therefore, for each node $n^i$ an earliest schedule $s_e^i$ and latest schedule $s_l^i$ is given. If $s_e^i = s_l^i$ then node $n^i$ is called a critical node. A path containing only critical nodes is called a critical path. If $s_e^i > s_l^i$ for any $n^i \in V$ then no valid schedule can be found to meet the given deadline. For $s_e^i < s_l^i$ a range of schedules is available.

**Definition 3.1 Schedule range of a DG node** *For a dependence graph $G = \{V, E\}$ given the earliest schedule time $s_e^i$ of all input nodes $n^i \in I$, and the latest schedule time $s_l^i$ of all output nodes $n^i \in O$, the schedule range $r^i$ of node $n^i$ is $[s_e^i, s_l^i]$, where $s_e^i$ is the earliest schedule time of the node $n^i$ and $s_l^i$ is the latest schedule time of the node $n^i$.*

From the above we conclude that for a given $DG$, each node $n^i \in V$ can only be scheduled in the schedule range $r^i = [s_e^i, s_l^i]$, where $s_e^i, s_l^i \in Z^+$. There are a number of basic requirements needed for generating a suitable schedule for a $DG$. Based on the features of array processors we define the basic requirements for scheduling.

**Definition 3.2 Scheduling requirements**

1. *The schedule of a node n must be at least one unit delay higher than the highest schedule of all nodes having an edge to node n.*
2. *Input nodes $n^i \in I$ should not be scheduled earlier than the requirements defined by the system.*
3. *Output nodes $n^j \in O$ should not be scheduled later than the system defined deadline.*

The basic idea of the Alternative Node Schedules $ANS$ algorithm is to generate an enumeration tree of all possible schedules in the $DG$ given the schedule ranges $r^i$ such that the requirements in Definition 3.2 are satisfied. The $ANS$ algorithm generates a set of solutions $GS$. A solution $GS^i \in GS$ is a set of $\{< n^j, s^j >\}$ where $n^j$ is a node with schedule $s^j$. The $ANS$ algorithm in its general form generates a set of solutions which grows exponentially with the problem size. In practice the designer is only concerned with solutions that are optimal with respect to certain criteria. Note that introducing a wide schedule range gives a huge amount of solutions which may be redundant, time consuming, and may even be hard to generate for large $DG$'s. There exists a maximum range beyond which the set of solutions offers no more improvement. This range depends on the dimension of the array processor.

A semi-regular $DG$ contains a set of connected sub-$DG$'s. These sub-$DG$'s are regular. Keeping uniform delay distribution in the sub-$DG$'s simplifies the design. Let $R^i$ be the set of all edges along a linear path and $E^j$ be the set containing all edges on a selected number of linear paths belonging to several $R^i$ with parallel edges. We partition the set of edges in the $DG$ into a number of sets $E^i$, such that $\forall_i \forall_{j \neq i} E^i \cap E^j = \phi$ and $\cup_i E^i = E$. A set $\{< E^i, d^i >\}$ specifies for each $E^i$ a delay $d^i$ (i.e. all the edges in $E^i$ have the same delay $d^i$). Once the schedule time for an edge along a linear path in $E^i$ is chosen, the delay $d^i$ is fixed for all edges in $E^i$. This is done for all sets $E^i$.

Further, for mapping from a $M$-dimensional $DG$ to a $K$-dimensional array processor, any node can be connected to a maximum of $3^K - 1$ nodes scheduled in the same time slot. We now define the set of constraints for scheduling as follows.

**Definition 3.3 Schedule constraints**

1. *The maximum number of nodes $n^j$ having an edge to $n^i$ and the same schedule time must be less than $3^K - 1$.*
2. *For all nodes with only external input edges set $s_l^i = s_e^i$ and/or for all nodes with only external output edges set $s_e^i = s_l^i$*
3. *For each sub-DG, create a set $\{(E^i, d^i)\}$ for each edge direction $(3^K - 1)/2$ in the K-dimensional Euclidean space.*

We call the constraints in Definition 3.3 for generating the set of optimal solutions $GS_{opt}$, *tight* constraints. A solution may be either linear or non-linear. Under *tight* constraints, the set of solutions $GS_{opt}$ is constant for a specific algorithm with respect to problem size.

**Theorem 3.1 Bounding rule for the scheduling** *The size of the set $GS_{opt}$ is independent of the problem size if*

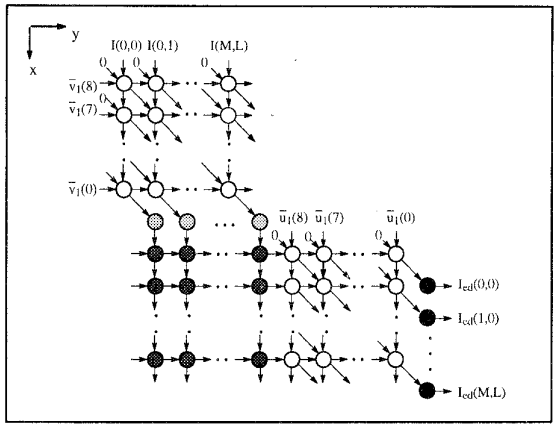


Figure 1: The DG of the Edge Detector for a mask width of 8 and image of $(M+1) \times (L+1)$ pixels.

*the bounding constraints are tight. Furthermore, the sets of solutions for different sizes are equivalent.*

The reader is referred to [MOE94] for the proof. In case of tight constraints the complexity of the *ANS* algorithm is $\mathcal{O}(UH)$, where $U$ is the number of sub-*DG*'s and $H$ is the number of edge directions. In fact, even if the constraints are not so tight, there still exists a set of solutions that is independent of the size of the DG. Since the result is invariant to the problem size, the set of optimal schedules can be generated from a *Reduced size DG* (*RDG*). The complexity of the algorithm for calculating the schedule time of all nodes in *DG* is $\mathcal{O}(N_r \times A_\alpha)$, where $N_r$ is the number of nodes in the *RDG*, $A_\alpha = \prod_{i=1}^{M} \alpha_i$ and $\alpha_i$ is the scaling factor for the $i^{th}$ dimension.

We choose the edge detection problem as an example. The *DG* of the Edge Detector is three dimensional. Figure 1 shows one part of algorithm. The other part is identical. Readers are referred to [JAY91a] for the derivation. Due to the huge processing power requirement parallel processing is needed. The black nodes on the far right add the result of both parts. White nodes are convolution functions and dark grey nodes are row to column translation functions. It is clear that the *DG* for this problem is inhomogeneous. We now apply the *ANS* algorithm for a mask width of 4 and an image width of 4, 8 and 16. The schedule range for all nodes is 2. A bounding constraint is added for each edge direction in all subgraphs. The behavior of the three different image sizes is compared in the plot of Figure 2. In all three cases, the final set of possible schedules contains five optimal solutions for all different *DG* sizes.

## 4. ALTERNATIVE NODE PROJECTION

We construct an algorithm to find all valid linear (non-

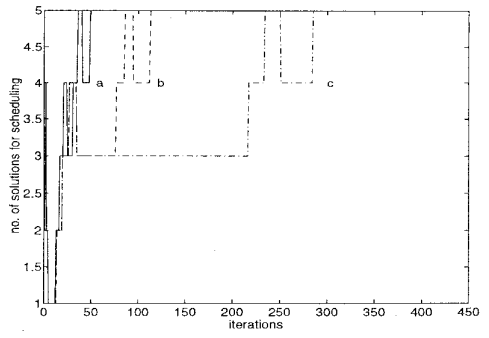


Figure 2: Image Detector with Mask Width 4 (a)Image Width 4 (b)Image Width 8 (c)Image Width 16.

linear) projections. Linear mapping involves projection along a straight line whereas nonlinear mapping means that multiple nodes not necessarily along a straight line map to the same *PE*. In certain circumstances, a nonlinear mapping may offer some unique flexibility and advantages such as fewer *PE*'s, faster pipelining, or higher utilization of the array. This may allow nonlinear scheduling. On the other hand it usually incurs the expense of somewhat sophisticated control. If an advantageous trade-off can be reached, a nonlinear schedule mapping may become preferred. We define the projection requirements based on the definition of array processors in [KUN88].

**Definition 4.1 Projection requirements:**

- *Preserve spatially local interconnection.*
- *The dimension of the array processor determines the maximum number of communication links allowed.*
- *Input/Output nodes should remain on the boundary.*

The set of permissible positions to place a node $n^i$ on a *PE* depends on the current position of all nodes that have an edge to node $n^i$. Each instance of the placement can be modeled by a rectangular volume which we call *polyrec. Polyrec is a polytope such that all the hyperplanes lying on its boundary are perpendicular to one of the axes and orthogonal to each other. Further a polyrec is fully represented by two extreme points lying on the boundary.*

Let $\vec{p}^j$ represent the location of a *PE* on the array processor such that $\vec{p}^j = [p_1^j, p_2^j, ..., p_K^j]^T$. Let $P^i$ be the set of all positions $\vec{p}^j$ of nodes $n^j$ having edges to node $n^i$. Given the set $P^i$, the set of all valid positions that node $n^i$ can occupy resides inside the polyrec $A^i$. The *Polyrec* $A^i$ can be fully represented by two extreme points [MOE94]. For a number of edges $t^i$ to a node $n^i$ and dimension $K$ the complexity of finding *polyrec* $A^i$ is $\mathcal{O}(t^i K)$. In general there are a maximum of $3^K$ positions a node can be projected into, where $K$ is the dimension of the array processor. We can now define *polyrec* $A^i$ for each node $n^i$ in

the $DG$ when the position of all the nodes having an edge to node $n^i$ is known. Based on this an algorithm is developed to generate all possible solutions. We start the Alternative Node Projection ($ANP$) algorithm by placing all nodes which have only external input edges in the $K$-dimensional array processor space. Due to data dependencies these combinations are limited. The rest of the nodes are then placed using *polyrec* $A^i$. We propose the following projection constraints:

**Definition 4.2 Projection Constraints:** *The enumeration tree for projection solutions must be pruned for each node $n^i$ if any of the following conditions hold:*

- *A node $n^i$ can only be projected onto a position $\vec{p}^j$ that lies within the polyrec $A^i$.*
- *Two nodes with the same schedule can not be projected to the same PE.*
- *The number of* PE*'s should not exceed some upper bound.*
- *Maximum number of communication links $3^M - 1$ must be preserved for each* PE.

Up to now, factors such as complexity of the resulting $PE$ and non-uniform distribution of I/O nodes on the boundary has not been taken into account. The ultimate performance goal of an array processor system is a computation rate that balances the available I/O bandwidth with the host. In order to achieve this we have to guarantee that the I/O nodes are uniformly distributed and match the interface to the outside world. An additional set of constraints are therefore needed.

**Definition 4.3 Additional projection constraints:**

- *Input/Output nodes should remain on the boundary.*
- *Prevent the mapping of nodes with different functionality onto the same* PE*(optional).*
- *Remove equivalent and similar solutions.*

The $ANP$ algorithm finds the set of all possible mappings $\{GP^i\}$ under the constraints in Definitions 4.2 and 4.3. It maps a $M$-dimensional $DG$ to a $K$-dimensional array processor and finds the set of all possible linear and non-linear projections. No solution is found if a node violates the set of constraints for all intermediate solutions.

Linear projection has been thoroughly studied in literature. Yet in certain circumstances a non-linear mapping may offer some unique flexibility and advantage. To extract the optimal linear and non-linear solutions $\{GP^i\}$ in terms of array processor characteristics and given constraints we need to define an extra set of constraints which we call bounding rules. Let us define a bounding rule for

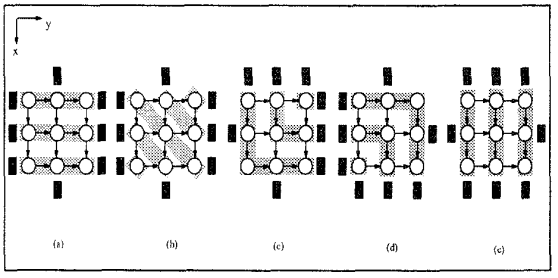


Figure 3: All possible linear and semi-linear projections (a) Horizontal (b) Diagonal (c) L-shape (d) inverted L-shape (e) Vertical.

a regular $M$-dimensional $DG$. The intersections of all hyperplanes lying on the boundary of the $DG$ form a polytope. We call this polytope a $DG$-polytope. Let $R^i$ be a set of all edges having the same direction along a linear path and $E^i$ be the set containing all edges on a selected number of paths.

**Definition 4.4 The bounding rule for a regular M-dimensional DG of size $(\alpha_1, \alpha_2, ..., \alpha_M) \in Z_M^+$:** *For set $R^i$ on the linear path joining two vertices of the DG-polytope and lying on the boundary, all edges in $R^i$ have to follow the same rule of projection i.e. edge directions after projection are identical to each other.*

Definition 4.4 guarantees that I/O nodes are mapped uniformly. This can be generalized to include the set $E^i$. The enhanced $ANP$ algorithm uses set $E^i$ to add bounding constraints. An example of the projection set $\{GP^i\}$ for a 3x3 matrix-vector multiplication using Definition 4.4 is given in Figure 3. This set contains linear as well as semi-linear mapping. Whether the matrix is a $3 \times 3$ or $n \times n$ $DG$, the set $\{GP^i\}$ contains 5 alternative solutions which are equivalent for all sizes of the $DG$. In case an $m \times n$ $DG$ where $m \neq n$, only solutions (a), (b) and (e) are possible. This means that given a regular array and using Definition 4.4, the set $\{GP^i\}$ is dependent on the topology of the $DG$ but independent of the size of the $DG$. This is a very interesting result.

The above discussion assumes that the $DG$ boundaries lie on hyperplanes orthogonal to each other. This is not always the case e.g sorting problem. We therefore define a general bounding constraint for regular $DG$'s.

**Definition 4.5 General bounding rule for a regular M-dimensional DG:** *For set $E^i$ of the linear path joining two vertices of the DG-polytope and lying on the boundary of the DG, all edges in $E^i$ have to follow the same projection rule i.e. edge directions after mapping are identical to each other. If the path consists of floating nodes, create a set $E^i$ of all edges, parallel to each other*

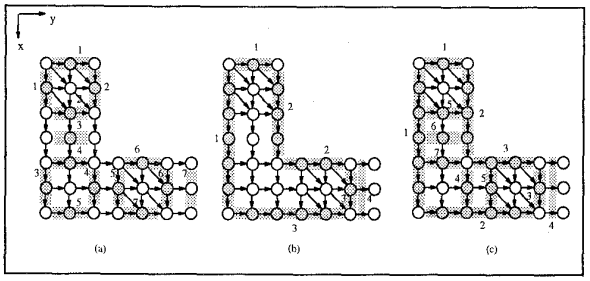

Figure 4: Different constraint techniques for the image detector (a) Local Boundary (b) Global Boundary (c) Global Boundary with orthogonal boundary constraints.

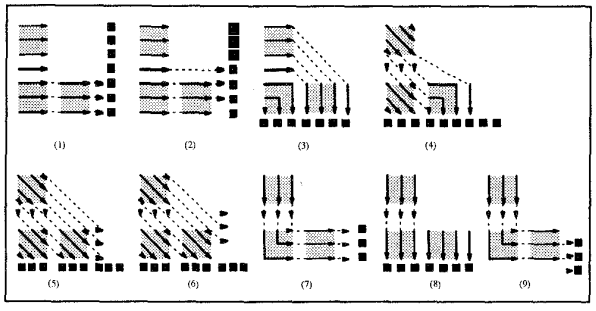

Figure 5: Solutions for an image width of N and a mask width of N using global bounding constraints.

*and entering the floating nodes. The source nodes of these parallel edges are treated as boundary nodes.*

A more general problem representation is given by semi-regular $DG$'s. This is evident in algorithms which consist of a set of interconnected recurrency equations. $DG$'s of such algorithms consist for example of a cascade of regular sub-$DG$'s. Treating the $DG$ as a whole will ease the data reformatting and increase the pipeline rate.

In the edge detector example there are 34 projections possible when boundary constraints of Definition 4.4 are placed on each sub-$DG$ as shown in Figure 4 (a). Notice that all the nodes in the last sub-$DG$ are floating (far right). In that case additional boundary constraints between sub-$DG$'s are needed. Placing boundary constraints on the $DG$ as a whole (Figure 4 (b)) results in 9 optimal solutions (Figure 5). **Definition 4.5 is generalized for semi-regular $DG$'s as follows:** *For all boundaries between sub-DG: if any of the boundaries have floating nodes add bounding constraints as above.* We now define an important theorem.

**Theorem 4.1 Tight Bounding rules for projection of a DG** *Given a DG there exists a bounding constraint which generates the set of all optimal solutions $GP_{opt} = \{GP^i\}$. The set $GP_{opt}$ is independent of the problem size if the bounding constraints are tight and the scheduling chosen applies a uniform distribution of delays.*

See proof in [MOE94]. This method can be applied on any form of mapping by changing the valid-position rules and the constraints rule. Even if the bounding constraints are not tight enough, then the reduced size DG will still generate a set of optimal solutions $S_r$. This is evident in Figure 4 (b) and (c). Figure 4 (b) has constraints on the complete $DG$ which are not so tight on the sub-$DG$'s. In this case, both bounding constraints give the same set of mappings seen in Figure 5. Yet the computation time of Figure 4 (b) is higher because the internal nodes have a higher degree of mobility. Further, the set of solutions in Figure 5 has linear as well as nonlinear projections. The non-linear solution (3) has a simpler $PE$ since the convolver and row to column translation nodes are mapped to different $PE$'s. Furthermore, the number of $PE$'s performing multiplication is fixed to 8 independent of the image size.

## 5. COMPLEXITY ISSUES AND SCALING

The time bound for both algorithms is limited by $V-1$ stages where $V$ is the set of nodes. The average computations per stage are proportional to the set of edges to a node. It is apparent that all computations along the hyperplane orthogonal to the flow of data have no mutual dependency. Therefore they can be executed simultaneously. In general there is always a certain degree of dependency which dictates the sequence of the computation. The choice of the order in which nodes are to be placed in each step has an influence on the computation time but has no effect on the end result. Take the image detector example with image length of 3 and mask width of 3 and find the set $\{GP^i\}$ for the constraints as given in Figure 4 (c). For different ordering we get a different distribution of the set of intermediate solutions. Three orders were simulated as shown in the plot of Figure 6.

The local maximum of the peaks increase as the computation gradually proceeds because within the search path, the internal nodes have a higher degree of freedom than boundary nodes. For a small size $DG$ this is not a problem but as the size of the $DG$ increases this grows exponentially. This may cause the algorithm to run out of memory before reaching a solution. There are two ways to solve this problem. One is to add additional internal constraints concentric to the boundary constraint. This will reduce the internal peaks and speed up the calculation yet guarantee that the set of optimal solutions $\{GP^i\}$ is the same. Since the result is invariant to the size according to Theorem 4.1, another way is to solve for small size arrays and then scale up the result to the required size. The complexity of the algorithm for scaling is $\mathcal{O}(\mathcal{W}+\mathcal{E})$, where

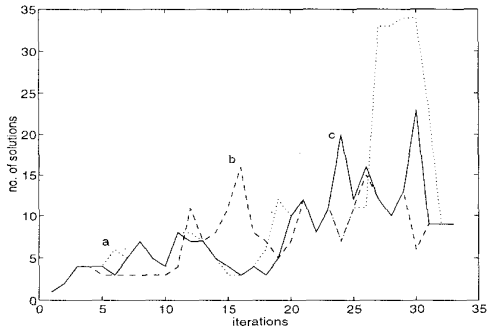


Figure 6: Variation of the number of solutions as a function of (a) Horizontal order (b) Vertical order (c) Breadth first preordered spanning tree.

$\mathcal{W}$ is the number of nodes in $DG$ and $\mathcal{E}$ is the number of edges in $RDG$.

## 6. MAPPING TO A FIXED SIZE ARRAY

A major area of research for systematic design methods is dedicated to the general problem of mapping classes of algorithms onto regular array processors with limited number of processing elements, communication link or memory size. Systematic design of processor arrays with a given dimension and given number of *PE*'s is called *partitioning*. Existing approaches to the partitioning problem, however do only partially treat the problems like mapping from a $M$ to $K$ dimensional space directly, where $M > K$. Another point is that the approaches are bound to special structures. A unified approach to the solution of the partitioning problem to realize all known partitioning schemes [TEI93] and to linear and nonlinear mapping is not available. The algorithm mentioned in this paper can be used to map arrays with limited resources. An upper limit on the number of *PE*'s can be used or a *boundary representation* (b-reps) is defined.

## 7. CONCLUSIONS

A systematic approach is presented for mapping algorithms into array processors. This approach uses the branch-and-bound technique to find the set of all optimal solutions. The power of this approach lies in the ability to generate the set of possible mapping alternatives using mixed linear and non-linear mapping. It has also been shown that the resulting set is limited and independent of the problem size. This is especially interesting for modeling large and complex problems. Further, mapping from $M$-dimensional space to $K$-dimensional space, where $M > K$, is done in one step.

For mapping to fixed size arrays, it has been shown that different partitioning techniques, can be modeled in the algorithms using *regularized Boolean set operations* for the design of 2 and 3-dimensional array processors.

## 8. REFERENCES

[ANN88] J. Annevelink, *A Design Method for Implementing Signal Processing Algorithms on VLSI Processor Arrays*, Ph.D. Thesis, University of Delft, The Netherlands, 1988.

[JAY91a] J.A.K.S. Jayasinghe, *An Array Processor Design Methodology for Hard Real-Time Systems*, Ph.D. thesis, University of Twente, The Netherlands, 1991.

[JAY91b] J.A.K.S. Jayasinghe, F. Moelaert El-Hadidy and O.E. Herrmann, *An Array Processor Design Methodology for Hard Real-Time Systems*, IEEE International Symposium on Circuits and Systems, Singapore, 11-14 June 1991.

[KUN84] S.Y. Kung, J. Annevelink and P.M. Dewilde, *Hierarchical Iterative Flow-Graph Integration for VLSI Array Processors*, VLSI Signal processing, IEEE Press, pp. 294-305, 1984.

[KUN88] S.Y. Kung *VLSI Array Processors*, Prentice Hall, 1988.

[MOE92] F. Moelaert El-Hadidy and O.E. Herrmann, *Integer Linear Programming Algorithms for Array Processor based Real-Time Systems*, University of Twente, Internal Report number 92N188, August 1992.

[MOE94] F. Moelaert El-Hadidy , *Generalized Methodologies for Array Processor Design of Real-Time systems*, University of Twente, PhD. Thesis, 1994.

[MOL87] Dan I. Moldovan *ADVIS: A Software Package for the Design of Systolic Arrays*, IEEE Transaction on Computer-Aided Design, vol. CAD-6, no. 1, pp. 33-39, January 1987.

[QUI84] P. Quinton, *Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations*, Proceedings of the 11th. Annual Symposium on Computer Architecture, pp. 208-214, July, 1984.

[RAO88] Sailesh K. Rao and Thomas Kailath, *Regular Iterative Algorithms and their Implementation on Processor Arrays*, Proceedings of the IEEE, vol. 76, no. 3, pp. 259-269, March 1988.

[ROY86] V.P. Roychowdhury and T. Kailath, *Regular Processor Arrays for Matrix Algorithms with Pivoting*, ISI preprint, Standford University, Standford, CA, 1989.

[TEI93] J. Teich and L. Thiele, *Partitioning of Processor Arrays: A Piecewise Regular Approach*, Integration, The VLSI Journal, vol. 14, no. 3, pp. 297-332, February 1993.

[THI88] L. Thiele, *On the Hierarchical Design of VLSI Processor Arrays*, IEEE Symposium on Circuits and Systems, Helsinki, pp. 2517-2520, 1988.