Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 268402, 15 pages
doi:10.1155/2012/268402

Research Article

Design Space Exploration of Deeply Nested Loop 2D Filtering and 6 Level FSBM Algorithm Mapped onto Systolic Array

B. Bala Tripura Sundari

Department of ECE, Amrita Vishwa Vidyapeetham, Coimbatore 641 112, India

Correspondence should be addressed to B. Bala Tripura Sundari, balasrikanth2003@yahoo.com

Received 26 December 2011; Revised 9 April 2012; Accepted 23 April 2012

Academic Editor: Sungjoo Yoo

The high integration density in today's VLSI chips offers enormous computing power to be utilized by the design of parallel computing hardware. The implementation of computationally intensive algorithms represented by n-dimensional (n-D) nested loop algorithms, onto parallel array architecture is termed as mapping. The methodologies adopted for mapping these algorithms onto parallel hardware often use heuristic search that requires a lot of computational effort to obtain near optimal solutions. We propose a new mapping procedure wherein a lower dimensional subspace (of the n-D problem space) of inner loop is identified, in which lies the computational expression that generates the output or outputs of the n-D problem. The processing elements (PE array) are assigned to the identified sub-space and the reuse of the PE array is through the assignment of the PE array to the successive sub-spaces in consecutive clock cycles/periods (CPs) to complete the computational tasks of the n-D problem. The above is used to develop our proposed modified heuristic search to arrive at optimal design and the complexity comparisons are given. The MATLAB results of the new search and the design space trade-off analysis using the high-level synthesis tool are presented for two typical computationally intensive nested loop algorithms—the 6D FSBM and the 4D edge detection alternatively known as the 2D filtering algorithm.

1. Introduction

1.1. Prelude to the New Search Method. Today's reconfigurable SoCs feature processing elements (PEs) with significant amount of programmable logic fabric present on the same die. The management of complexity and tapping the full potential of these RSoC architectures present many challenges [1]. A large number of heuristic algorithms have been used in developing many novel scheduling and mapping algorithms [2–5]. However, these approaches face difficulties in dealing with large execution times.

n-dimensional (n-D) nested loop representations are used in the formulation of numerous computationally intensive multimedia computing/image processing and signal processing algorithms. Systolic array design style can effectively exploit parallelism inherent in the nested loop algorithm and, therefore, reduce processing time [2, 3]. Often heuristic procedures are used to search for the mapping transformations that are used to map the nested loop algorithms onto array architectures [4, 5]. Since the effort that goes into heuristic search is large and complex, the challenge lies in improving the process to reduce the computational effort in getting the mapping results.

Our main contribution in this paper is that we propose an augmented approach to the heuristic search. A new method of identifying the subspace to which the PE array is to be assigned is proposed based on the directional index of the computational expression that is explained in Section 2. The new vectors and terminologies used in the procedure are defined and elaborated in Section 2.

A modified heuristic search is implemented using the proposed procedure to determine the optimal solution to the n-D problem. The complexity analysis is performed by comparing the search space used in our method with the search space in [4]. The high-level synthesis tool GAUT is used to plot the design space trade-off curves to obtain the design space exploration curves.

The paper is organized as follows: in Section 3, mapping steps used in the heuristic method and our proposed modified search method are described. The 4D nested loop formulation of the 2D filtering problem is explained in Section 4. The methodology and the implementation of the above approach for the 2D filtering algorithm and the mapping results are presented in Section 4. The mapping process for 6D FSBM is elaborated in Section 5 followed by the results of the heuristic search for the reduced 4D FSBM and modified heuristic search for the same in Section 6. The design trade-off results using the high-level synthesis tool GAUT are presented in Section 6. Section 7 discusses the complexity considerations and comparisons. Section 8 gives the conclusion and future work.

2. Terminologies and Definitions

*2.1. Axis Vector **I**.* The multidimensional (n-D) problem is associated with an n-dimensional axis vector $\mathbf{I}$. Its components are $\{i_1, i_2, i_3, \ldots, i_n\}$, where the subscripts of the components belong to the integer set z. The components of the vector $\mathbf{I}$ represent the different axis directions of the n-dimensional vector $\mathbf{I}$. The letter $\mathbf{K}$ is used to represent a constant vector whose components are different constant numbers, $\mathbf{K} = \{k_1, k_2, k_3, k_4, \ldots, k_n\}$. Each k_z represents the upper limit of the corresponding vector component i_z of the vector $\mathbf{I}$. For example, axis component i_4 has a value varying from $i_4 = 1$ to $i_4 = k_4$.

2.2. Data Representations. Considering the input data set to the algorithm, the input data is represented using letter $\mathbf{A}$ with subscript z. The input data set consists of collection of data $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ where k is some constant integer number. Each of this type of data is associated with the axis vector $\mathbf{I}$. For example, for $\mathbf{A}_1$, we call it as $\mathbf{A}_1(\mathbf{I})$. Now every such data is associated with a particular axis component i_z in $\mathbf{I}$. i_z is the axis vector along which the data $\mathbf{A}_1(\mathbf{I})$ is read into the n-D multidimensional algorithm using a set of ports. The input data is represented as $\mathbf{A}_1(i_1, I_2, i_3, \ldots . i_n)$. This means that input data $\mathbf{A}_1(\mathbf{I})$ is fed along i_2 axis. The corresponding word size is k_2, and the port size required to feed this data is k_2. The input data is reused either within the same computation or in different computations within iteration (depending on the application considered). If the reuse is within the same clock cycle/clock period CP, it is made possible by propagating the data (with zero delay) termed as data broadcast. The reuse direction of each data is represented by the directional vector termed as the "dependence vector"—D_v. D_v is determined as follows: as shown in Listing 1, the data $\mathbf{A}_1$ on the LHS is assigned from the data $\mathbf{A}_1(i_1, I_2, i_3 - 1, \ldots, i_n)$ on the RHS in equation (1a) in Listing 1. This means that it is broadcast within the same iteration in the i_3 direction and fed along the i_2 axis using k_2 ports (Figure 1).

The output data is represented as $\mathbf{C}(i_1, i_2, i_3, \ldots, I_n - 1)$ which means that the data is output along i_n axis and propagated along i_n direction. When we consider the output data, the word "propagation" is replaced by the term "update direction." The vector associated with update direction is termed as the Computational Trail Vector (CTV). The updation of CTV may be with delay or without delay as demanded by the application.

The vector representing the update direction in this example is given as

$$\text{CTV} = [0, 0, \ldots, 1]. \tag{1}$$

The form of representation of the n-D algorithm in Listing 1 wherein the broadcast direction and computations are shown with the complete detail is termed as the uniform recurrence relation or the URE form of the n-D nested loop algorithm. In the expression (2) for CTV in Listing 1, the computational output data C is represented as $\mathrm{C}[\vec{\mathbf{I}}]$ (arrow line on top of the symbol) which indicates that it is associated with an update direction. The corresponding vector $\vec{\mathbf{d}}$ in the RHS of (2) represents the CTV defined in (1).

The functions f_1 and f_2 in (3) in Listing 1 are simple commutative operators which are executed independent of any other output component computations of C. These are assumed to be operators with no precedence constraint. f_2 especially is an operator that has no precedence constraint. It needs not wait for any past computations. It can proceed independently provided as much parallel hardware is available. There is only one output computation expression in Listing 1. Listing 1 is said to have a single CTV with no precedence constraint.

2.3. n-D Nested Loop Problem. A general n-D nested loop algorithm is illustrated in Listing 1. i_1, i_2, and $i_3, \ldots, i_n$ are the loop indices. Together they form the n-D (iteration) index space. Representation of the n-D loop computations as a dependence graph (DG) leads to each point in the index space corresponding to a single node in the DG. Theoretically each node can be assigned a processing element (PE). The n-D iteration space is constructed as follows.

2.3.1. An n-D Iteration Space Computation in Terms of (n−1)-D Subspace. First an $(n-1)$-D dependence graph (DG) as in Figure 1 with an $(n-1)$ multidimensional indexed positions given by

$$\left[\ \vec{\mathbf{I}}_{n-1}^{i_n}, 1\right] = \{i_1, i_2, i_3, \ldots, i_{n-1}, 1\} \tag{2}$$

is constructed showing the data input directions and data broadcast directions. Here we show one of the data input directions and data broadcast directions for the sake of illustration. The data specifications or the dependence relations within each cell in the iteration space show the different data broadcast directions as shown in Figure 1.

The n-D iteration space is constructed by replicating the $(n-1)$-D iterationspace along the i_n direction. Each $(n-1)$-D subspace is termed as a *cell* (or iteration). An array of PE is assigned to this cell, and the computation of the cell is completed in 1 clock period (CP). In the next CP, the PE array is assigned to the next cell along the i_n direction. The direction of PE array assignment to consecutive subspaces is termed as the scheduling direction

Do i_1 = 1 to k_1;
Do i_2 = 1 to k_2;
Do i_3 = 1 to k_3;

$$A_1(i_1, I_2, i_3, ..., i_n) = A_1(i_1, I_2, i_3 - 1, ..., i_n);\text{//broadcast in } i_2 \text{ direction} \quad \textbf{(1a)}$$

$$A_2(i_1, i_2, I_3, ..., i_n) = A_2(i_1 - 1, i_2, I_3, ..., i_n);\text{//broadcast in } i_1 \text{ direction}$$

$$A_3(I_1, i_2, i_3, ..., i_n) = A_3(I_1, i_2, i_3 - 1, ..., i_n)\text{broadcast in } i_3 \text{ direction}$$

$$A_{n-1}(i_1, i_2, i_3, I_4, ..., i_n) = A_{n-1}(i_1, i_2, i_3 - 1, I_4, ..., i_n);$$

$$C[\vec{\mathbf{I}}] = f_2(C[\vec{\mathbf{I}} - \vec{\mathbf{d}}], f_1(A_1 A_2\ [\vec{\mathbf{I}}\ ])) \quad \textbf{(2)}$$

or

$$C[i_1, i_2, i_3, ..., i_n] = C[I_1, I_2, i_3, ..., i_n - 1] + A_1(i_1, I_2, i_3, ..., i_n) \times A_2(I_1, i_2, i_3, ..., i_n) \quad \textbf{(3)}$$

End Do i_n; ....;
End Do i_2;
End Do i_1;

Listing 1: n-D multidimensional algorithm in URE form.

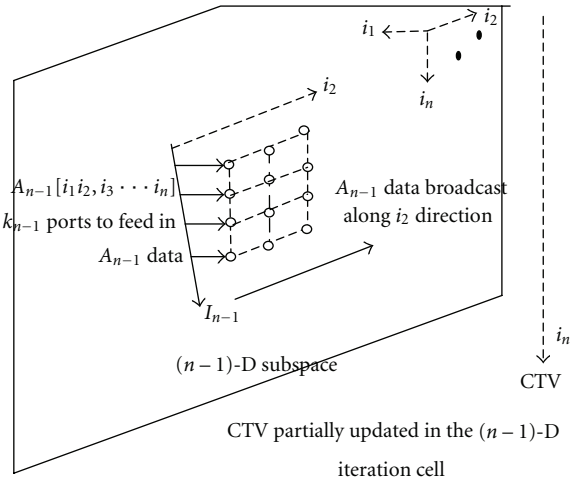


Figure 1: The input data set and computation in the first $(n-1)$-D subspace or cell represented as DG.

represented as the scheduling vector $\vec{sd}$. As per Listing 1, the CTV is also updated along the same i_n direction. The CTV is partially updated in CP1, and the updation continues as the scheduling advances along the i_n direction in every CP till the completion of computation in k_n CPs.

2.4. Mapping and Scheduling. Any node in the iteration space is $N[i_1, i_2, i_3, \ldots, 1]$ and is mapped onto the PE array assigned to the iteration subspace. This is termed as *mapping*. The time "t" at which the node $N[i_1, i_2, i_3, \ldots, 1]$ is mapped on the PE in the PE array is termed as *scheduling*. The mapping and scheduling are derived for each application in detail in the corresponding sections.

2.5. Computation of n-D Iteration Space Using an $(n-2)$-D Subspace. In an alternate generalization, we represent the n-D nested loop problem as identified to have an iterative $(n-2)$-D subspace as shown in Figure 2. An $(n-2)$-D dependence graph (DG) with an $(n-2)$-D multidimensional indexed positions is given by

$$\left[\vec{\mathbf{I}}_{n-1}^{\,i_n, i_{n-1}}, 1, 1\right] = \{i_1, i_2, i_3, \ldots, i_{n-2}, 1, 1\}. \quad (3)$$

The collection of indexed node positions in (3) is termed as the $(n-2)$-D subspace or hyperplane, which is represented showing the data input directions and data broadcast directions in Figure 2(a). The n-D iteration space computation is completed by replicating the $(n-2)$-D DG. We expand the iteration space along the i_{n-1} direction, followed by its expansion along i_n direction. Each $(n-2)$-D subspace is termed as a *cell* or iteration cell. An array of PE is assigned to this cell, and the computation of the cell is completed in 1 CP.

A part of the output expression termed as the computational expression is assumed to be computed in the inner loop formed by the $(n-2)$-D iteration space as depicted in Figure 2(a). The directional index representing the propagation direction or the update direction of the computational expression is termed as the Computational Trail Vector (CTV). The CTV is partially updated in CP1, and the updation continues as the scheduling advances along the i_{n-1}, showing that in the next CP the PE array is assigned to the next iteration cell along the i_{n-1} direction (as shown in Figure 2(b)) to complete the first row of computation in k_{n-1} CPs. The sequence direction of subspace assigned to the PE array in consecutive CPs is termed as the scheduling direction represented by the scheduling vector $\vec{sd}_1$, which is along the i_{n-1} direction, and CTV is also updated along the same i_{n-1} direction.

Following this, the PE array assignment is done to next i_n giving the scheduling vector $\vec{sd}_2$ as i_n as in Figure 2(b). The total number of CPs used to complete the computation is $k_{n-1} \times k_n$.

2.6. n-D to $(n-x)$-D with CTV and Scheduling Directions. In the previous section, the $(n-1)$-D subspace is built using a sequence of $(n-2)$-D subspaces by scheduling along

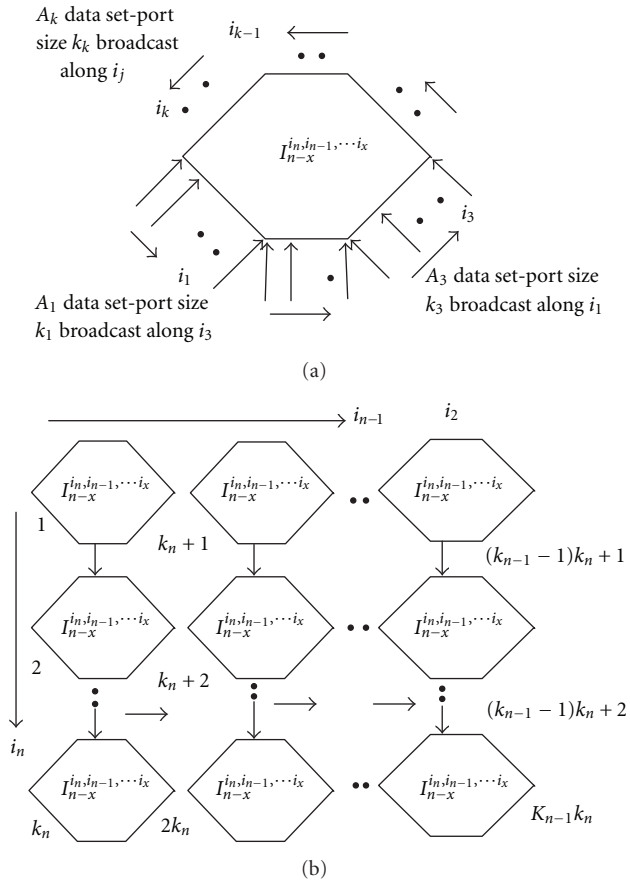


Figure 2: (a) The $(n-2)$-D iteration cell. (b) $(n-2)$-D iteration space with scheduling $\vec{sd}_1$ along i_{n-1} direction CTV is also updated along the same i_{n-1} direction, followed by $\vec{sd}_2$ along i_n, and CTV is also updated along the same i_n direction.

the appropriate $(n-1)$th dimension followed by scheduling along the appropriate nth dimension—say along i_n with an assumption that CTV has the same direction as the scheduling vector which may not be true always. There are two approaches to complete the n-D computation using the $(n-2)$-D subspaces. The PE array assignment to the $(n-2)$-D subspace is one order closer to the physical realization. For a practical implementation, this process has to be continued down to 2D level.

In general, the direction of updation of the computational expression is defined as a vector termed as the Computational Trail Expression (CTV) of the n-D problem. We identify the corresponding $(n - x)$-D computational hyperplane in which the CTV lies, forming an $(n - x)$-D subspace in the n-D space. The PE array is assigned to this plane. This is followed by the reuse of the $(n - x)$-D plane along the scheduling direction/s.

3. Methodology of Mapping

The mapping methodology used in the heuristic search of the mapping transformation matrix M is explained hereafter. In general, the mapping matrix M is constituted of the timing vector or hyperplane S and the space matrix or vector also called the space hyperplane P [6, 7]. Any node in the iteration space $N[i_1, i_2, i_3, \ldots, i_n]$ is mapped onto a PE in the PE array using the P matrix at a time "t" determined by the S vector of [4]

$$M = \begin{bmatrix} S \\ P \end{bmatrix}. \tag{4}$$

Table 1: Dependence vectors for 2D filtering.

Variable	LHS assignment	RHS assignment	Dependence vector
Image data	$I\ (i, j, k, l)$	$I\ (i+1, j, k-1, l)$	[1 0 1 0]
Image data	$I\ (i, j, k, l)$	$I\ (i, j+1, k, l-1)$	[0 1 0 1]
Window coefficient	$W\ (i, j, k, l)$	$W\ (i-1, j, k, l)$	[1 0 0 0]
Window coefficient	$W\ (i, j, k, l)$	$w\ (i, j-1, k, l)$	[0 1 0 0]
Output	$O\ (i, j, k, l)$	$O\ (i, j, k, l-1)$	[0 0 0 1]
Output	$O\ (i, j, k, l)$	$O\ (i, j, k-1, l)$	[0 0 1 0]

3.1. Heuristic Method [4]

Step 1. Generate the iteration space for the n-D nested loop application under consideration.

Step 2. Find the data dependencies in the algorithm and formulate the dependence vector D_v.

Step 3. The causality constraint is checked for using (5), that is, whether the condition

$$S * D_v > 0 \tag{5}$$

for all dependencies is satisfied, where D_v is dependence vector for each data variable (Table 1). Choose those s elements of S which satisfy the condition.

Step 4. Generate or modify the search space for the M matrix (M_{set}) to satisfy the rank condition [4].

Step 5. Chose a candidate M matrix from the above set.

Step 6. Save the candidate M matrix in M_{result}.

3.2. The Proposed Modified Heuristic Method. The following are the steps in our approach for modification of the heuristic search based on the optimal allocation method evolved in Section 2.

Identify the scheduling direction. Once a layer of PEs is assigned to the $(n - x)$-D subspace, the same array of PEs is to be used in the next computation. This reuse direction is known as the scheduling direction in Section 2. All these conditions are used in the modified heuristic search procedure in the following steps.

Table 2: Delay-edge determination—Step 11 in Section 4.5.

Case (i) window size = $[w_1\ w_2]_1$ = [3 3]	Case (ii) $[w_1\ w_2]_2$ = [4 3]
Image size = [R C] = [1 1]	[R C] = [1 1]
Image size = one window size	
D_v—dependence matrix	D_v dependence matrix
1 0 1 0 0 0	1 0 1 0 0 0
0 1 0 1 0 0	0 1 0 1 0 0
1 0 0 0 0 1	1 0 0 0 0 1
0 1 0 0 1 0	0 1 0 0 1 0
To determine delays use	Delays
Sdd vector = [1 0; 0 1; 1 0; 0 1; 0 0; 0 0]	Sdd vector = [1 0; 0 1; 1 0; 0 1; 0 0; 0 0]
sdd $*$ $[w_1\ w_2]_1$ = sdd $*$ [3 1]	sdd $*$ $[w_1\ w_2]$ = sdd $*$ [4 1]
Delays = [3 1 3 1 0 0]	sdd $*$ [4 1] = [4 1 4 1 0 0]
To determine edge_connectivity use	sde = [1 0; 0 1; 0 0; 0 0; 0 1; 1 0]′
Sde vector = [1 0; 0 1; 0 0; 0 0; 0 1; 1 0]′	
sde $*$ $[w_1\ w_2]_1$	= sde $*$ [4 1]
ans = 3 1 3 1 0 0	ans = 4 1 0 0 1 4

Step 7. The scheduling vector representing the scheduling direction represented by the $\vec{sd}_1$ vector is used to prune down the valid M matrices.

Step 8. Prune down the valid M matrices by choosing the $(n - x)$-D subspace to which the PE plane is discussed. This is done by identifying the iterative subspace. To summarise, the selected M_{mat} is obtained by pruning down the M_{result} using the CTV and PE plane assignment done as discussed in Section 2.

Step 9. Evaluate the cost function as given in (10) in Section 5.2. If $\text{Cost}_{\text{actual}} < \text{Cost}_{\text{required}}$, proceed to Step 6 else to Step 3.

The plots of Figure 5 show the comparison of heuristic method of Section 3.1 with the modified heuristic search method described in Section 3.2.

3.3. Direct Method

Step 10. The delay edge is calculated by the direct method as explained in Section 4.5. The results are presented in Table 2.

Step 11. The delay edge matrix in Table 2 is determined using the expression D_v defined in Tables 1 and 3 for 2D filtering algorithm.

3.4. Mapping Process. The main objective is to find the M matrix which consists of the processor allocation vector (P^t) and the scheduling vector (S^t).

Table 3: Dependence vectors for each variable for 2D filtering.

2D filtering	dv1	dv2	dv3	dv4	dv5	dv6	***
Index variables	I_1	I_2	w_1	w_2	O_1	O_2	
i	1	0	1	0	0	0	Next row window
j	0	1	0	1	0	0	Next window-column wise
k	1	0	0	0	0	1	PE array along k and l
l	0	1	0	0	1	0	

* p-direction—2D array represented as a 1D array.
** index variables.

First we take the boundaries of the search space between which the P^t and S^t are to be searched. The selection of search space is an important factor, because there is an exponential growth in both area and time complexity of the mapping methodology. Consider that $U_i, U_j, U_k, \ldots, U_n$ are the upper bounds of an n-dimensional nested loop algorithm. The heuristic followed in this work is to generate the search space that can be obtained by the following element set $\{0, 1, U_i, U_j, U_k, \prod(U_i U_j)\}$.

3.5. Methods and Resources Used in Obtaining the Mapping Methodology. As a whole, the implementation of the mapping methodology consists of two parts. The first is the heuristic search for the mapping. The heuristic search allows us to obtain the near optimal solutions and then pick up the feasible architecture by pruning the solutions based on Steps 4–9 as described in Section 3.3. The new mapping methodology is explained with respect to the 2D filtering algorithm in Section 4. The modified heuristic method based on the new method followed after implementing the steps in Section 3.1 are implemented using MATLAB to obtain the results of the search procedure of Sections 3.1 and 3.2. Also the comparative results between the heuristic and the modified heuristic method for the 6D full search block motion estimation (FSBM) algorithm are given. The second part is the design space exploration of resultant architecture. It is obtained as explained in the next section.

3.6. High-Level Synthesis (HLS). The input to high-level synthesis system is the problem represented in behavioural description in a high-level language. The optimization in a high-level synthesis is done at a level higher than the boolean optimization done by the RTL synthesis tools. This is suitable for hardware optimization of DSP and image processing algorithms [8]. This is followed by scheduling and allocation [9]. The GAUT [10] tool used incorporates all the above features and allows the design space exploration.

The algorithm is described in a high-level description in C, and this is used as the input design specification to the high-level synthesis tool. The high-level synthesis tool is used to obtain the Control Data Flow Graph (CDFG). The CDFG allows the designer to verify the design required at a later stage. It allows the tracing of data values as live variables in

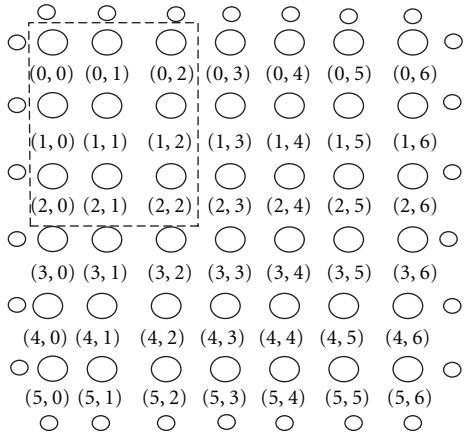


Figure 3: Window for the 2D filtering algorithm.

registers associated with the PE hardware. Also the high-level synthesis tool is used to obtain the design space exploration results which give the area Versus latency tradeoff.

4. Mapping of 2D Filtering Algorithm

4.1. 2D Filtering for Image Processing: A 4D Problem. The problem formulation of Section 2 and the methodology in Section 3 are applied to the 2D filtering problem. 2D filtering or convolution is one of the essential operations in digital image processing required for image enhancements. The grey levels are usually represented with a byte or 8-bit unsigned binary number, ranging from 0 to 255 in decimal. Equation (6) shows the two dimensional discrete convolution algorithm, where $I[x, y]$ represents the input pixel data image, W is the window coefficient, and O is the output image. The movement of the mask window function to calculate the window function value for the whole image region is shown in Figure 3:

$$\begin{aligned} O[x, y] &= W[x, y] \otimes I[x, y] \\ &= \sum_{i=0}^{4} \sum_{j=0}^{3} I[i + x, j + y] \otimes W[i, j]. \end{aligned} \tag{6}$$

Digital convolution can be thought of as a moving window of operations, where the window that is, mask, is moved from left to right and from top to bottom.

The 2D image filtering problem is a representative example of a 4D nested loop involving 2D convolution, as in Listing 2 and Figure 3. The computation is highly redundant and requires high data reuse. This is considered here for systolic mapping. An image of size 0 to $+k_1$; 0 to $+k_2$ is considered convolved with a mask of size 0 to $+k_3$; 0 to $+k_4$. The mask coefficients are stored in memory. The significant features of the algorithm are listed in the following section.

4.2. Nested Loop Formulation. The nested loop formulation for the 2D filtering algorithm for image size $k_1 \times k_2$ and window function size $k_3 \times k_4$ is given in Listing 2—the same is represented in uniform recurrence equation form (URE) in Listing 3.

4.3. Single Assignment Statement Formulation or Uniform Recurrence Equation (URE) Form of 2D Filtering. The SAS of the 4D edge detection algorithm is in Listing 3, and the dependence vectors for the four level algorithms have 4 indices and the index space is generated by varying the four index values till the upper limit of each index as in Listing 3. The dependencies give the propagation direction of the input variables and update direction of the output data. In Listing 3, W_{new} represents the mask values in 2D filtering algorithm that are to be input at the fresh windowing and I_{new} to indicate the loading of pixel values for a new frame of image.

4.4. Dependence Vectors for 2D Filtering Algorithm. Listing 3 is well commented to bring out the formulation of the following dependence vectors in Table 1.

4.5. Delay-Edge Matrix-Direct Method of Determining Delay and Edge Connectivity. The delay edge mapping is obtained by the product of dependence matrix (D_v) and M matrix as shown in Table 2.

Step 11 in the mapping process uses the dependence matrix to compute the edges and delays as follows: D_v = [0 1 0 1; 1 0 1 0; 0 0 0 1; 0 0 1 0; 0 0 01; 0 0 1 0]′ (Table 1); the first half in each vector in D_v stands for the scheduling direction and the second half for the PE array directions. The first half (termed as sdd vector—$\vec{sdd}$) gives the delays associated with the corresponding edges given by the second half (sde vector = $\vec{sde}$):

$$\begin{aligned} \text{Delays} &= \left[\vec{sd}_2 \times w1\ 1\right] \times \vec{sdd}; \\ \text{Edges} &= \left[\vec{sd}_2 \times w1\ 1\right] \times \vec{sde}. \end{aligned} \tag{7}$$

This is computed and presented in Table 2.

Mapping results for 2D filtering are given in Table 4(a) for heuristic method, and Table 4(b) gives the modified heuristic method.

4.6. Space-Time Mapping Matrix (M) Illustration. The mapping was performed for 1D array. The generalized form of space time mapping matrix M is given here as shown in (3);

$$M = \begin{bmatrix} S^t \\ P^t \end{bmatrix}. \tag{8}$$

if $P^t = \begin{bmatrix} 0 & 0 & 3 & 1 \end{bmatrix}$; $S^t = \begin{bmatrix} 4 & 1 & 5 & 1 \end{bmatrix}$.

Table 4

(a) Heuristic search results for 2D filtering

NPE	Ncyc	M-matrix	Reg. cost
12	12	1 0 1 3	
0	0	1 1 1 2	10
12	14	1 0 1 3	
0	0	1 1 1 4	14
12	18	1 0 1 3	
0	0	1 1 1 3	12
12	16	1 0 1 3	
0	0	1 1 1 1	8
12	12	1 0 1 3	
0	0	1 1 2 1	10
12	15	1 0 1 3	
0	0	1 1 2 2	12
12	17	1 0 1 3	
0	0	1 1 2 0	8
12	13	1 0 1 3	
0	0	1 1 2 4	16
12	21	1 0 1 3	
0	0	1 1 2 3	14
12	19	1 0 1 3	
0	0	1 1 2 1	10
12	15	1 0 1 3	
0	0	1 1 0 4	12
12	15	1 0 1 3	
0	0	1 1 0 3	10
12	13	1 0 1 3	
0	0	1 1 4 1	14
12	21	1 0 1 3	
0	0	1 1 4 2	16
12	23	1 0 1 3	
0	0	1 1 4 0	12
12	19	1 0 1 3	
0	0	1 1 4 4	20
12	27	1 0 1 3	
0	0	1 1 4 3	18
12	25	1 0 1 3	
0	0	1 1 4 1	14
12	21	1 0 1 3	
0	0	1 1 3 1	12

(b) Mapping results using the modified heuristic search results process 2D filtering

Window size = 3 × 3; 2D result arrived by using Step 11 [pe_arr, Ncyc_arr, M or Tmat]			Window size = 4 × 3 [pe_arr Ncyc_arr M or Tmat]		
NPE	Ncyc	M matrix = [P; S]	NPE	NCYC	M matrix = [P; S]
9	9	1 0 0 4; 1 1 2 1	12	12	1 0 1 4; 1 1 3 1
9	9	1 0 0 4; 1 3 0 4	12	12	1 0 1 4; 1 3 1 4
9	9	1 0 0 4; 1 3 2 1	12	12	1 0 1 4; 1 3 3 1
9	9	1 0 0 4; 1 2 0 4	12	12	1 0 1 4; 1 2 1 4
9	9	1 0 0 4; 1 2 2 1	12	12	1 0 1 4; 1 2 3 1
9	9	1 0 0 4; 1 4 0 4	12	12	1 0 1 4; 1 4 1 4

(b) Continued.

Window size = 3 × 3; 2D result arrived by using Step 11 [pe_arr, Ncyc_arr, M or Tmat]			Window size = 4 × 3 [pe_arr Ncyc_arr M or Tmat]		
NPE	Ncyc	M matrix = [P; S]	NPE	NCYC	M matrix = [P; S]
9	9	1 0 0 4; 1 4 2 1	12	12	1 0 1 4; 1 4 3 1
9	9	1 0 0 4; 1 1 0 4	12	12	1 0 1 4; 1 1 1 4
9	9	1 0 0 4; 1 1 2 1	12	12	1 0 1 4; 1 1 3 1
9	9	1 0 0 4; 0 1 0 4	12	12	1 0 1 4; 0 1 1 4
9	9	1 0 0 4; 0 1 2 1	12	12	1 0 1 4; 0 1 3 1
9	9	1 0 0 4; 0 3 0 4	12	12	1 0 1 4; 0 3 1 4
9	9	1 0 0 4; 0 3 2 1	12	12	1 0 1 4; −0 3 3 1
9	9	1 0 0 4; 0 2 0 4	12	12	1 0 1 4; 0 2 1 4
9	9	1 0 0 4; 0 2 2 1	12	12	1 0 1 4; 0 2 3 1
9	9	1 0 0 4; 0 4 0 4	12	12	1 0 1 4; 0 4 1 4
9	9	1 0 0 4; 0 4 2 1	12	12	1 0 1 4; 0 4 3 1
9	9	1 0 0 4; 0 1 0 4			
9	9	1 0 0 4; 0 1 2 1			
9	9	1 0 0 4; 2 1 0 4			
9	9	1 0 0 4; 2 1 2 1			

* Search space for M matrix without the use of the scheduling vector $\vec{sd}$; the execution time takes more execution time to obtain Table 4(a), than the search time which uses the $\vec{sd}$ as the projection direction for reassignment of PE plane used to obtain Table 4(b).

```
For(i1 = 0; i1 < k1; i1++)
For(i2 = 0; i2 < k2; i2++)
{
O[i1, i2] = 0;
        For (i3 = 0; i3 <= +k3; i3++)
        For (i4 = 0; i4 <= +k4; i4++)
        O[i1, i2] = O[i1, i2] + I[i1 + i3, i2 + i4] × w[i3, i4]        (8)
        //Output one window function evaluation
        Ends do i4; End do i3;}
    End i2;
End i1;
```

Listing 2: 2D filtering algorithm nested loop formulation.

4.6.1. Delay-Edge Matrix. The delay edge mapping is obtained by the product of dependence matrix (D) and M matrix:

$$\mathrm{DE}_{\mathrm{mat}} = M * \mathrm{DV}_{\mathrm{mat}},$$

$$\begin{bmatrix} 4 & 1 & 5 & 1 \\ 0 & 0 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 3 & 0 & 0 \\ 1 & 3 & 0 & 1 & 4 & 1 \end{bmatrix}. \quad (9)$$

4.7. Direct Method-Edge Connectivity and Delay Registers. The direct method in deriving the delay edge connectivity is obtained from the dependence vector as given in Table 6.

(1) The delay edge matrix based on the heuristic search is used to calculate the cost as given in Table 4(a) and the above can be used to pick up the good solution based on minimum cost, but does always guarantee the feasibility. So we do not consider the delay edge obtained from this method.

(2) Using the proposed modified search algorithm, 9, 9 or 12, 12 are the number of PEs and number of clock cycles in Table 4(b) (for assumed window size in Table 4(b)) are arrived at after pruning down the search results using the PE-plane subspace based on CTV.

(3) As mentioned above, the delay edge connectivity is obtained directly from the dependence matrix directly by considering the scheduling directions for delays and considering the PE directions for the edges as discussed in Section 4.5 and as shown in Table 2 and the architecture is obtained using the mapping results and direct delay_edge connectivity.

```
   For i = 1 to k1
   For j = 1 to k2
For k = 1 to 4 //window size = 4 × 3
                                    For l = 1 to 3
                                    If (i == 1 && j == 1)
                                          w(i, j, k, l)= Wnew
                                    Else if(j == k2)
                                          w(i, j, k, l) = w(i − 1, j, k, l);//next i
                                    //[Dvw1 = 1 0 0 0]
                                    Else
                                    w(i, j, k, l) = w(i, j − 1, k, l);
                                    // next j [Dvw2 = 0 1 0 0]
                                    End if
      If (i == 1 && j == 1)
      I(i, j, k, l) = Inew;
      Else if(i == 1 && j > 1) // first row—second window calculation
      I(i, j, k, l) = I(i, j + 1, k, l + 1); move to the next j pixel −j + 1; pixel data—reads in next column of pixel and
      old data is moved in the (k, l) plane-PE_array from (k, l) to (k, l + 1);//[DVx2 = 0 1 0 1]
      Else if (j==k2)          // for next i
      I(i, j, k, l) = I(i + 1, j, k + 1, l); // move to the next i pixel i + 1; pixel data—reads in next row of
      // pixel and old data is moved in the (k, l) plane-PE_array from (k, l) to (k + 1, l)
      //[DVx1 = 1 0 1 0]
      End if
      If (l ==1 && k == 1)
      O(i, j, k, l) = 0;
      Else if (k < 3)
      O(i, j, k, l) = O(i, j, k, l − 1) + I(i, j, k, l) ∗ W(I, j, k, l);
      Else
      O(i, j, k, l) = O(i, j, k − 1, l) + I(i, j, k, l) ∗ W(I, j, k, l);
      End;
End For l, k, j, i;
```

Listing 3: URE algorithm for 2D filtering.

4.8. Mapping Results. The cost function is defined as (10) and is used as an additional constraint mentioned to Step 9 in Section 3.2 for selecting architecture according to the modified heuristic method heuristic search

$$\text{Cost} = a * \text{processors} + b * \text{cycles} + c * \frac{\text{delays}}{\text{reg}}. \quad (10)$$

Here a, b, c are the scalar coefficients which represent weights for the corresponding costs to minimize the overall cost function.

4.9. Architecture. Figure 4 shows the architecture for edge-detection algorithm. It consists of 2 ports, one for accessing the image data and the other for the output. The architecture consists of $w_1 \times w_2$ PEs, where $w_1 \times w_2$ is the size of the window used. The intermediate output is propagated to the successive PEs within a row but has to be passed through a line buffer when passing the intermediate output between rows of PEs. The buffer width is equal to the number of pixels per row. The final output is at the $w_1 \times w_2$ PE.

Figure 5(a) shows the search results giving the possible solutions including the register cost. Registers represent the delays in the connecting edges which are the result of heuristic search, but which may not be feasible or realizable. The Pareto optimal and near optimal solutions are shown in the plots Figures 5(a) and 5(b) based on the heuristic search and the modified heuristic search, respectively. The modified heuristic search developed by us picks up the good solutions with respect to the number of PEs and cycles concerned, but we see that the register cost does not reflect the Pareto optimal solution and does not guarantee feasibility. The delay-edge connectivity is obtained directly from the dependency vectors as explained in Section 3.3 and in Table 2 for 2D filtering and leads to the feasible architecture in Figure 4.

5. Mapping of 6D FSBM

The main objective is to find the M matrix which consists of the processor allocation vector (P^t) and the scheduling vector (S^t). The method used is same as explained in Section 3.

5.1. Dependencies for 6-Level FSBM Algorithm. Dependence vectors formulations have been presented for a reduced index space 4D FSBM algorithm [11]. Due to lack of space, it is not presented.

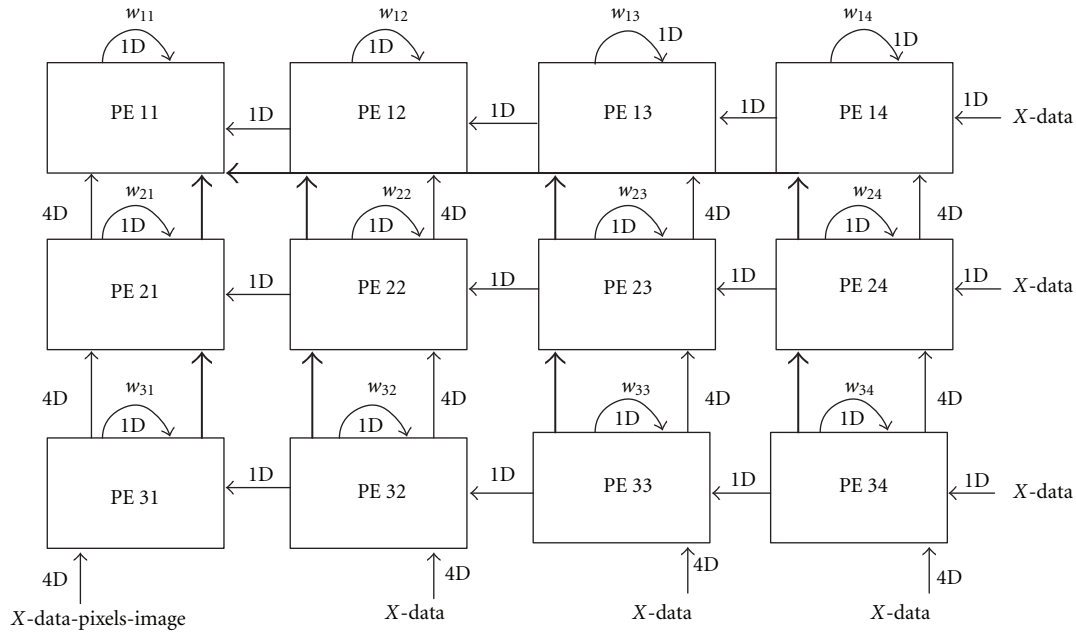


Figure 4: Architecture for 2D filtering algorithm for window size 4×3.

5.2. Results of Modified Method for FSBM Algorithm. The mapping results after the search are presented here.

The heuristic search results of Tables 5 and 8(a) (using MATLAB) for $p = 1$ and 2, respectively, are shown in the graph in Figure 6.

5.3. Delay-Edge Connectivity for FSBM Algorithm Using Table 5 Results

(1) $\begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix} * D_v = 3\ 1\ 1\ 3\ 3\ 1\ 0\ 0\ 3\ 1$; the edges we get are same as the elements in D_v at the p-direction (hence verified), and the delays $[0\ 1\ 0\ 0] * D_v$ = ans = 3 1 16 4 3 1 1 0 16 16 ans = register/delays for the variables x, y, MAD, Dmin. This is obtained as a good solution from Table 5 by selecting the optimum cost taking into consideration the feasibility.

(2) The final delay edge is given as follows:

$$\begin{bmatrix} 0 & 0 & h \times N^2 & N & N & 1 & 1 & 1 & N^2 & N^2 \\ 2P+1 & 1 & 1 & 2p+1 & 2P+1 & 1 & 0 & 0 & 2P+1 & 1 \end{bmatrix}. \tag{11}$$

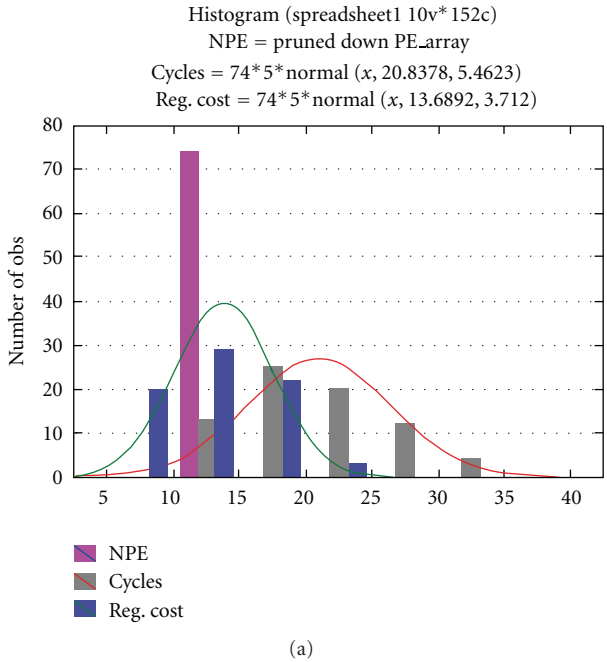
The second row is the edge, and the first row is the registers connected obtained as the highest nonzero value, in the D_v values along other indices other than p-direction in Listing 2. p-direction is the direction of orientation of the systolic array (PE array) in the n-D problem space. The above gives a minimal cost connectivity and register delay elements simultaneously satisfying the feasibility and implementability checked by the direct method.

Table 5: 4D FSBM—heuristic search.

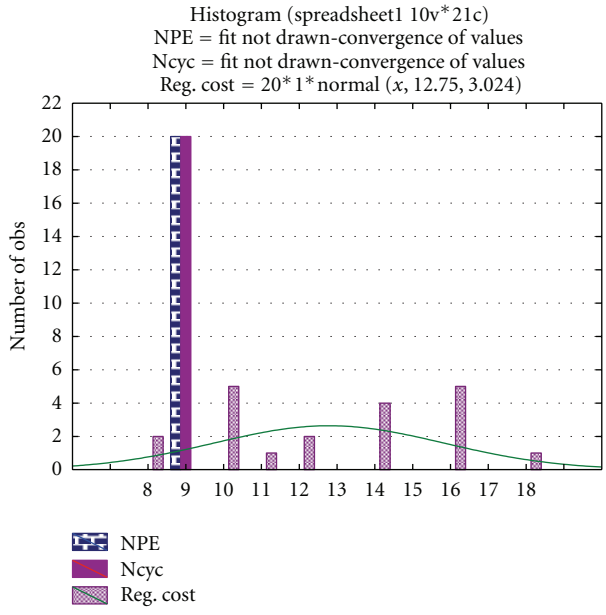
Mmat	NPE[I]	Ncyc[II]	Reg cost[III]	Total cost = 0.4 ∗ I + 0.4 ∗ II + 0.2 ∗ III
0 1 0 1	9	24	16	15.35
0 1 0 0	9	27	Edge	
0 1 1 1	9	24	19	15.5
0 1 0 0	9	27	Edge	
1 1 0 1	9	16	68	14.75
0 1 0 0	9	19	Edge	
1 1 1 1	9	24	71	18.1
0 1 0 0	9	27	Edge	
1 0 0 1	9	16	52	13.95
0 1 0 0	9	19	Edge	
1 0 1 1	9	24	54	17.25
0 1 0 0	9	27	Edge	
3 1 0 1	9	16	172	19.95
0 1 0 0	9	19	Edge	
3 1 1 1	9	24	174	23.25
0 1 0 0	9	27	Edge	
3 0 0 1	9	16	158	19.25
0 1 0 0	9	19	Edge	
3 0 1 1	12	24	160	24.2
0 1 0 0	12	27	Edge	
9 1 1 1	12	16	500	38
0 1 0 0	12	19	Edge	

6. Architecture of the FSBM Algorithm

The architecture is arrived at, based on above is in Figure 7.



(a)



(b)

Figure 5: (a) Plot for Table 4(a)—heuristic search. (b) Plot for Table 4(b)—modified heuristic search.

6.1. Design Space Exploration Using High-Level Synthesis. The design space exploration results are presented in the following based on the architecture arrived at.

6.2. CDFG of the Design. The architecture in Figure 7 is input using a behavioural description using a C type language to the GAUT tool, and it generates the control data flow graph (CDFG) architecture as in Figure 8 and also integrates into ModelSim and Xilinx ISE.

Table 6: Results of modified method for 4D FSBM algorithm for $p = 1$.

>> [pe_arr Ncyc_arr, Mmat]	Reg. cost	Total cost = 0.4 ∗ I + 0.4 ∗ II + 0.2 ∗ III
0 0 1 1 4 0	35	17
9 16 1 0 0 1		10
0 0 1 3 2 0	56	21.2
9 16 1 0 0 1		10
0 0 1 2 3 0	43	18.6
9 16 1 0 0 1		10
0 0 1 4 1 0	69	23.8
9 16 1 0 0 1		10
0 0 1 1 4 0	30	16
9 16 1 0 0 1		10
0 0 0 1 4 0	28	15.6
9 16 1 0 0 1		10
0 0 0 3 2 0	54	20.8
9 16 1 0 0 1		10
0 0 0 2 3 0	41	18.2
9 16 1 0 0 1		10
0 0 0 4 1 0	67	23.4
9 16 1 0 0 1		10
0 0 0 1 4 0	28	15.6
9 16 1 0 0 1		10
0 0 2 1 4 0	31	16.2
9 16 1 0 0 1		10
0 0 2 3 2 0	58	21.6
9 16 1 0 0 1		10
0 0 2 2 3 0	45	38.5

Table 7: Design space exploration of the FSBM for $p = 1$.

Cadency	Operators, stages	Area	% use rate	Number of muxes	FF	Latency
40	22, 2	88	100	48	336	60
50	8, 2	64	100	96	288	80
100	5, 2	40	60,90,10,10 10,10	160	224	120
150	2, 1	16	60	128	144	140
200	2, 1	16	45	128	144	140

6.3. Results of Design Space Exploration. The high-level synthesis tool allows the designer to input the timing constraint as the cadency values to obtain the tradeoff of allocation of hardware as obtained in Table 7 for $p = 1$ for FSBM algorithm.

6.4. Design Space Exploration for $p = 2$. The search range p in FSBM algorithm is increased to $p = 2$, and the design space exploration is done in MATLAB for the modified heuristic and also using the HLS GAUT tool.

The results of the above are shown in Figure 9.

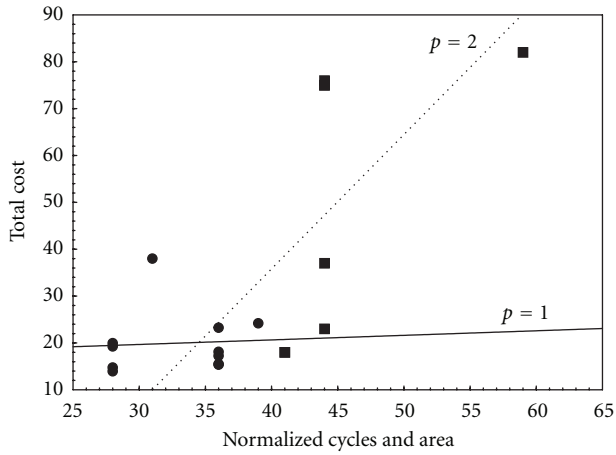


Figure 6: Graph-search results for reduced 4D FSBM heuristic search-cost function versus (normalized area and cycles) for Table 5 for search range $p = 1$ and Table 8(a) for search range $p = 2$.

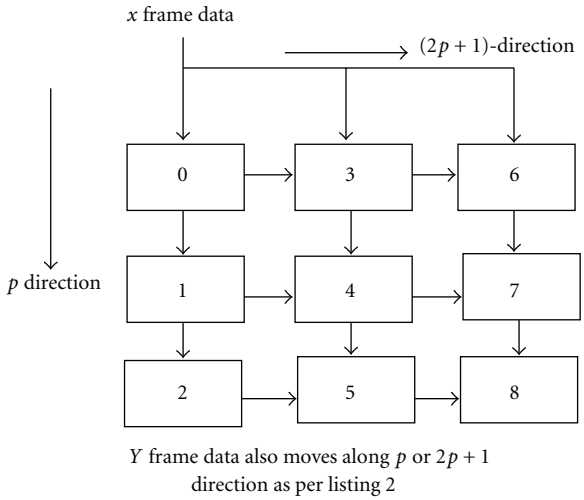


Figure 7: FSBM architecture after design space exploration.

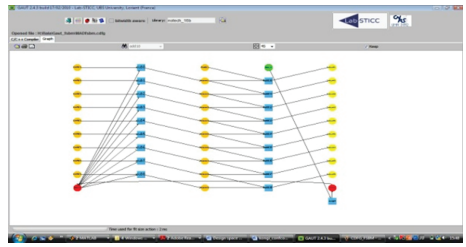


Figure 8: CDFG of the FSBM architecture in high-level synthesis tool.

Table 8

(a) Search results of MATLAB for $p = 2$

Npe	Ncyc	Reg. cost	Total cost
25	16	8	18
40	4	99	37
25	16	8	18
40	4	290	75
1	379	393	231
40	4	291	75
16	28	27	23
25	19	293	76
40	19	293	82

(b) Design space exploration GAUT-FSBM for $p = 2$

Cadency	Area	Number of operators	Latency
50	144	18	90
60	152	19	80
70	112	14	100
80	112	14	100
100	104	13	120
150	64	8	170
200	32	4	180
300	40	5	320
400	16	2	340

7. Complexity Analysis

The merit of the modified heuristic algorithm is measured in terms of the search space complexity.

7.1. Search Space Complexity. In general, in heuristic search procedures, the loop bounds are considered as the maximum values for searching. But as the loop bounds and the nested loop dimension increase, the search space will be huge if vectors are exhaustively generated. A graphical representation of search space expansion with respect to the different values of n for n-level nested loop algorithms is given in Figure 10.

The "a" bars show the search space obtained by taking the loop bounds, say $-U_i$ to $+U_i$, as the limit for each variable, and the "b" bars are obtained by using our proposed modified heuristic elaborated in Section 3, where it is observed from the plot in Figure 10 that the increase in cost is not high.

7.2. Search Space Complexity Tables. Tables 9(a) and 9(b) show the complexity calculations for 6D FSBM and 4D FSBM and the proposed modified heuristic method whose results are in Tables 4(b) and 5(b).

Table 9(a) shows the complexity calculations for varying values of n and gives a comparison between the general heuristic method and the method presented in this paper.

7.3. 6D Problem Reduced to 4D FSBM [11] and 4D Problem-2D FIR Filtering Problem. The reduction in search space by

Table 9

(a) 6D problem—full search block motion estimation (FSBM) problem

$n = 6$	S&K-2D array	Our work 2D array—use of sd	Use of direct determination of S vector of expression (3)	2D array considered as 1D array
$U_h, U_v, U_m, U_n, U_i, U_j$	Image file—image size—$U_i \times U_j = n \times U_h, n \times U_v$ subframe size $U_i \times U_j$	-do-*	-do-	-do-
I_space	[1,1,1,1,1,1] to $[U_h, U_v, U_m, U_n, U_i, U_j]$	-do-	-do-	-do-
S_space	0, 1, −1, $U_h, U_v, U_m, U_n, U_i, U_j, U_h \times U_v, U_v \times U_m, \ldots, U_i U_j, \ldots, U_h \times U_v \times U_m \times U_n \times U_i \times U_j$**	-do-	-do-	-do-
CTV	Nil	[0,0,0,1,0,0]; [0 0 1 0,0,0]	[0,0,0,0,0,1], [0,0,0,0,1,0], [0,1,0,0,0,0], [1,0,0,0,0,0]	-do-
Scheduling direction = sd	Nil	[1,0,0,0,0,0]; [0,1,0,0,0,0]	-do-	-do-
Search space complexity—P vector—size—$[1 \times 2]$	$66^{2n} = 66^{12}$ (Number of possible elements of P matrix)	Pruned down using $P^t \times \text{sd} = 0 = P^{2n-2-2}$ $66^{12-2-2} = 66^8$ sd along 2 directions	Pruned down using $P^t \times \text{sd} = 0$ P^{2n-2-2} $66^{12-4} = 66^8$	P^{n-2} $66^{6-2} = 66^4$
S vector – size – $[1 \times 2]$	$66^n = 66^6$	Pruned down using $S^t \times \text{sd} > 0 = 66^{6-2}$	Nil***	Nil
Example	$66^{12} + 66^6 = P^{2\times n} + P^n$	$66^8 + 66^4$	66^8	66^4

* -do-entry same as in previous column,*** nil: not defined/not applicable.

(b) Reduced index space

n = 4-4D FSBM	S&K-2D array	Our work-2D array I—use of sd	Use of direct determination of S vector	2D array considered as 1D array
$U_{h\text{new}}, U_{p\text{new}}, U_i, U_j$	Image file—image size—$U_i \times U_j = N \times U_{h\text{new}}, N \times U_v$ Sub-frame size size $U_i, \times U_j$	-do-	-do-*	-do-
I_space	[1,1,1,1] to $[U_{h\text{new}}, U_{p\text{new}}, U_i, U_j]$	-do-	-do-	-do-
S_space	0, 1, −1, $U_{h\text{new}}, U_{p\text{new}}, U_i, U_j U_{h\text{new}} \times U_{p\text{new}}, \ldots, U_i U_j, \ldots, U_{h\text{new}} \times U_{p\text{new}} \times U_i \times U_j$**	-do-	-do-	-do-
CTV	Nil***	[0,1,0,0]; $[0, 2p_{\text{new}} + 1, 0,0]$	[0,0,0,1], [0,0,1,0], [0,1,0,0], [1,0,0,0]	-do-
Scheduling direction = sd	Nil***	[1,0,0,0]; [0,1,0,0]	-do-	-do-
Search space complexity—P vector—size—$[1 \times 2]$	$17^{2n} = 17^8$ (Number of possible elements of P matrix)**	Pruned down using $P^t \times \text{sd} = 0 = P^{2n-2-2}$ $17^{8-2-2} = 17^4$ sd along 2 directions	Pruned down using $P^t \times \text{sd} = 0$ P^{2n-2-2} $17^{8-4} = 17^4$	P^{n-2} $17^{4-2} = 17^2$
S vector—size—$[1 \times 2]$	$17^n = 17^4$	Pruned down using $S^t \times \text{sd} > 0 = 17^{4-2}$	Nil***	Nil
Example	$17^8 + 17^4 = P^{2\times n} + P^n$	$17^4 + 17^2$	17^4	17^2

** note $4p_1 + 4p_2 + 4p_3 + 1 = 7 + 6 + 4 = 17$.
*** nil: not defined; * do entry same as in previous column.

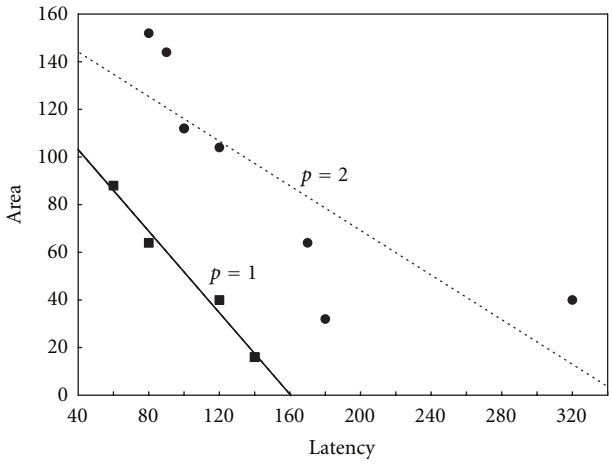


Figure 9: Design space exploration using HLS tool (Tables 7 and 8(b)).

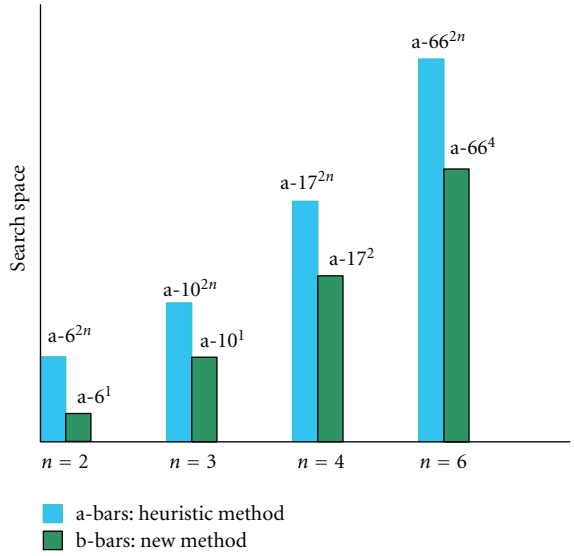


Figure 10: Plot showing the search space size and FSBM algorithm parameter (P) (with $N_v = N_h = 4$).

modifying the 6D algorithm to 4D as reported in [11] and also the benefit of the modified heuristic are reflected by the last entry in Table 9(b).

8. Conclusion and Future Work

Many of the computationally intensive algorithms are of n-D deeply nested loop type. The methodology of mapping of algorithms involves heuristic search wherein the search complexity is large. The search space of the 2D filtering and 4D FSBM has been pruned down using the scheduling vector $\vec{sd}$ and the constraints imposed by it. The search has been performed using MATLAB, for the PE array assigned to the identified $(n - x)$-D subspace evolved with the nature of the CTV. The resultant mapping matrix is useful in determining the PE assignment and the exact clock cycle at which a particular node in n-D space represented by the DG is mapped onto a PE in the PE array. The search results are presented for 2 computationally intensive applications—2D filtering and the reduced index space 4D FSBM algorithm. The graph in Figure 5(a) corresponds to Table 4(a) showing the heuristic search results that show the distribution of PEs and cycles and cost. Figure 5(b) corresponds to Table 5(b) that gives the number of PEs and cycles pruned down after applying the modified heuristic algorithm. The delay edge connectivity is determined by the proposed direct approach as described in Sections 3.3 and 4.5 using Tables 2 and 4, instead of using the Mapping Transformation Matrix M or Tmat in Tables 4(a) and 5(a) as in [4]. The use of high-level synthesis tool is to obtain the CDFG. Also the design space exploration results obtained using high-level synthesis tool GAUT have been presented. The search have been performed for varying search ranges of P values $P = 1$ and $P = 2$ and the number of resources used, and latency for different input cadency values gives the design trade-off results presented in Tables 7 and 8(b) shown in the graph in the Figure 9. The output file of the GAUT tool could be used to interface with simulation tools and synthesis tools to build the RTL design and map it onto target FPGA architecture in the future for elaborate timing verification. The complexity comparison of our method with heuristic method is given in Tables 9(a) and 9(b).

References

[1] C. Lee, S. Kim, and S. Ha, "A systematic design space exploration of MPSoC based on synchronous data flow specification," *Journal of Signal Processing Systems*, vol. 58, no. 2, pp. 193–213, 2010.

[2] U. Bondhugula, J. Ramanujam, and P. Sadayappan, "Automatic mapping of nested loops to FPGAS," in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '07)*, pp. 101–111, San Jose, Calif, USA, March 2007.

[3] X. Zhang and K. K. Parhi, "High-speed VLSI architectures for the AES algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 9, pp. 957–967, 2004.

[4] S. Kittitornkun and Y. H. Hu, "Mapping deep nested do-loop DSP algorithms to large scale FPGA array structures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 2, pp. 208–217, 2003.

[5] D. Peng and M. Lu, "On exploring inter-iteration parallelism within rate-balanced multirate multidimensional DSP algorithms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 1, pp. 106–125, 2005.

[6] P. Lee and Z. M. Kedem, "Synthesizing linear array algorithms from nested for loop algorithms," *IEEE Transactions on Computers*, vol. 37, no. 12, pp. 1578–1598, 1988.

[7] L. Lamport, "The parallel execution of Do loops," *Journal of Communication ACM*, vol. 17, no. 2, pp. 83–93, 1974.

[8] P. Coussy and A. Morawiec, Eds., *High-Level Synthesis—From Algorithm to Digital Circuit*, Springer, 2008.

[9] D. D. Gajski, N. D. Dutt, A. C. H. Wu, and S. Y. L. Lin, *High Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Press, 1992.

[10] http://www-labsticc.univ-ubs.fr/.

[11] B. Bala Tripura Sundari, "Dependence vectors and fast search of systolic mapping for computationally intensive image processing algorithms," in *Proceedings of the International Multi-Conference of Engineers and Computer Scientists 2011 (IMECS '11)*, Kowloon, Hong Kong, March 2011.