Automatic Hardware Synthesis of Nested Loops using UET Grids and VHDL

Nectarios Koziris, Theodore Andronikos, George Economakos, George Papakonstantinou and Panayotis Tsanakas

National Technical University of Athens
Dept. of Electrical and Computer Engineering
Computer Science Division
Zografou Campus, Zografou 15773, Greece

e-mail:{nkoziris, tedandr, papakon}@dsclab.ece.ntua.gr

Abstract: *This paper considers the automatic synthesis of systolic architectures from nested loop algorithmic specifications. The high level input is given in the form of uniform dependence loops with unit dependencies and the target architecture is a multidimensional systolic array with unbounded number of cells. A complete methodology for the hardware synthesis of the resulting architecture, based on VHDL specifications, is presented. This methodology automatically detects all necessary computation and communication elements and produces optimal layouts. The theoretical framework of our method is based on the properties of the generalized UET grids. First, we calculate the optimal makespan for the generalized UET grids and then we establish the minimum number of systolic cells required to achieve the optimal makespan. The complexity of the proposed scheduling algorithm is completely independent of the size of the nested loop and depends only on its dimension, thus being the most efficient (in terms of complexity) known to us. All these methods were implemented and incorporated in an integrated software package which provides the designer with a powerful parallel design environment, from high level algorithmic specifications to low-level (i.e., actual layouts) optimal implementation.*

Index terms: *UET grid index space, optimal makespan, optimal mapping, number of systolic cells, uniform unit dependence vectors, VHDL based design automation.*

1. Introduction

One of the most tedious tasks for a lot of sequential algorithms is the execution of nested FOR-loops with data dependencies among their computations. If a computation in one iteration depends on a computation in another iteration, this dependence is presented as the vector difference of these two iteration indices. The majority of such algorithms present a regular vector pattern (uniform data dependencies). This means that the values of all dependence vectors are constants, i.e., they are independent of the indices of computations. A subclass of the class of uniform nested loops is the class of the unit dependence nested loops, where every dependence vector has zero or unit coordinates. Well-known linear algebra methods such as matrix multiplication, LU decomposition and signal processing algorithms such as convolutions fall into this category. Since dependence vectors describe computations' flow, they are used to find the optimal parallel execution time. The widely used method is based on Lamport [6] who introduced the term "hyperplane". The idea is to find a time schedule that partitions computations into different sets, which are called hyperplanes. All index points belonging to the same set can be executed concurrently. Moldovan [8], Shang [10] and others [4] applied this method to find a near optimal execution schedule for the FOR - loop algorithmic model.

The major problem after having found a time schedule, is to organize computations in space, i.e., assign indexed computations to processors. We are particularly interested in systolic structures that, due to their uniformity, are suitable for massive parallelism and low cost implementation. Lee [7], Moldovan [9] and Shang [11] presented techniques for automatic mapping of indexed computations into systolic arrays, while preserving the time schedules. One of the most difficult issues when using a systolic array is the efficient use of its cells. Most of the time systolic arrays have poor performance, due to the small utilization of their processing nodes. The regularity of the systolic array structure imposes serious obstacles in organizing computations efficiently and thus increasing the utilization of cells. The so far presented methods for mapping loop algorithms onto systolic arrays have poor cell utilization, and use exhaustive search-based mapping techniques.

When considering general loop scheduling, Andronikos et al. in [1], have given a polynomial time optimal scheduling and efficient mapping into systolic architectures. In addition to this, in UET scheduling of arbitrary index spaces, which are produced by general nested loops, Koziris et al in [5] have given a polynomial time schedule and polynomial time efficient mapping onto a clique of processors.

This paper presents a methodology and an integrated tool for the optimal scheduling for nested loops with unit dependencies on unbounded number of systolic cells. We not only find an optimal time schedule for loop iterations, but we also assign the concurrent iterations onto the less possible number of cells. We present an integrated tool, which accepts as input the nested loop specifications and produces optimal systolic designs for the subclass of loops with unit uniform dependencies. Our tool integrates the methods for optimal time and space scheduling onto unbounded number of processors presented in [2], and produces VHDL descriptions for the resulting architecture. In particular, a VHDL preprocessor called GENVHDL has been implemented, which translates optimal scheduling and mapping results into VHDL code, which can be afterwards fed into VHDL entry CAD tools for synthesis and simulation (e.g., XILINX, WorkView Plus from VIEWlogic etc). The use of VHDL is not a novel aspect, since many systems performing algorithmic transformations of high level input descriptions, incorporate VHDL. However, they use mainly behavioral descriptions, or both behavioral and structural. Our system uses strictly structural descriptions, because it determines the exact structure of the systolic architecture through the scheduling methodology and leaves no freedom to the CAD tool to reconstruct it.

As far as the theoretical framework behind the optimal mapping onto systolic arrays is concerned, we elaborate on the following: We model the nested loop index space with UET grids. We calculate the *optimal makespan* for UET grid-index spaces. Having established the optimal makespan, we proceed to calculate the *optimal number of cells*, i.e., the minimum number of cells required to achieve the optimal makespan. We present an optimal time and space scheduling policy for index spaces with specific space bounds. Our schedule partitions the iterations of the index space into disjoint sets. Each set contains vertices, which are executed on different cells at the same time. Our scheduling policy guarantees that neighboring iterations of the n-dimensional index space will be assigned to neighboring cells of the n-1 dimensional target systolic array. The time complexity of the scheduling and mapping method is independent of the index space size and depends only on the dimension n of the index space.

The rest of the paper is organized as follows: Section 2 contains useful notations and definitions; Section 3 establishes the optimal makespan and the minimum number of processors adequate for scheduling the iterations of a nested loop. Finally, in Section 4 the GENVHL preprocessor is analyzed and a complete nested loop example, which is optimally mapped onto a VLSI architecture and implemented using XILINX FPGA devices, is presented.

2. BASIC CONCEPTS AND DEFINITIONS

2.1. Notation - Model Description

Throughput this paper we use the following notation: N is the set of naturals, Z is the set of integers, n is the number of nested DO(FOR)-loops of the algorithm, m is the number of dependence vectors of the algorithm and DS is the set of the dependence vectors. Our work is focused on algorithms that exhibit regular computation patterns because they are especially suitable for mapping into systolic architectures. The exact algorithm model we use is depicted in Fig. 1.

```
FOR i1=l1 TO u1 DO
...
FOR in=ln TO un DO
    AS1(I)
    ...
    ASp(I)
END in
...
END i1
```

(I)

```
DO 10 i1=l1, un
...
DO 10 in=ln, un
     AS1(I)
     ...
     ASp(I)
10 CONTINUE
```

(II)

Fig. 1: The Algorithmic model.

In Fig. 1:

(1) l_i, $u_i \in Z$, $1 \le i \le n$.

(2) $\mathbf{I}=(i_1, ..., i_n)$.

(3) AS_1, ..., AS_p are assignment statements of the form $V_0(\mathbf{I})=E(V_1(\mathbf{J}_1), ..., V_k(\mathbf{J}_k))$, where V_0 is an output variable, E is an expression of the input variables V_1, ..., V_k, and

$\mathbf{d}_i=\mathbf{I}-\mathbf{J}_i$, $1\leq i\leq k$, are constant vectors, with *zero or unit coordinates* only.

The *index space* $J^n\subset Z^n$ is the set of indices $\{(i_1, ..., i_n) \mid i_i\in Z \wedge l_i\leq i_i\leq u_i,\ 1\leq i\leq n\}$. Each point in this n-dimensional integer space is a distinct instantiation of the loop body. In general, the geometrical form of the index space is that of a *hyperparallelepiped*; the exact shape depends on the indices' lower and upper bounds. The cardinality of J^n is $\prod_{i=1}^{n}(u_i-l_i+1)$. However, in the rest of this paper we shall assume that $l_i=0$, $1\leq i\leq n$, i.e., $J^n\subset N^n$. This assumption is not restrictive in any way because, given an arbitrary algorithm of this form, we can find an equivalent algorithm that satisfies the above criterion by means of a simple *translation*. $(u_1, \ldots, u_n)$ is called the *terminal point* of the algorithm and is denoted P_n. In the rest of the paper, we shall abstract an algorithm A by the ordered pair (DS, J^n).

Notice that the points of J^n are ordered *lexicographicaly*; the usual symbol < is used to denote this (linear) ordering. The following two things should be emphasized: (1) By definition, dependence vectors are always > than $\mathbf{0}$, where $\mathbf{0}=(0, ..., 0)$ and > is the *lexicographic* ordering and (2) In our model all dependence vectors are *uniform*, i.e., their elements are constants and not functions defined over index sets.

2.2. Definitions

Definition 2.1. For every point $\mathbf{j}$ of J^n, we define the following sets:

(1) $IN(\mathbf{j}) = \{\mathbf{i}\in J^n \mid \mathbf{j}=\mathbf{i}+\mathbf{d}$, where $\mathbf{d}\in DS\}$, and

(2) $OUT(\mathbf{j}) = \{\mathbf{i}\in J^n \mid \mathbf{i}=\mathbf{j}+\mathbf{d}$, where $\mathbf{d}\in DS\}$. ■

The unit dependence vectors induce a partial ordering over the points of the index space in a natural way. If $\mathbf{i}$ and $\mathbf{j}$ are two index points, we write $\mathbf{i}<\mathbf{j}$ iff $\exists\mathbf{d}_1\ldots\exists\mathbf{d}_k\in DS$ such that $\mathbf{j}=\mathbf{i}+\mathbf{d}_1+\ldots+\mathbf{d}_k$. The intuition behind the partial ordering notion is that the dependence vectors represent *precedence constraints* that have to be satisfied, in order to correctly complete the iterations represented by the index points. The formal definition of the schedule must reflect our intuition that a point $\mathbf{j}$ correctly begins its execution at instant k iff *all the points* $\mathbf{i}\in IN(\mathbf{j})$ *have completed their execution and communicated their results (if needed) to* $\mathbf{j}$ *by that instant.*

Definition 2.2.

● A schedule for algorithm A, denoted S(A), is an ordered couple (S_{TIME}, S_{CELL}), where S_{TIME} and S_{CELL} are the time and processor schedules, respectively, defined as follows:

(1) S_{CELL}: $J^n\rightarrow\{0, \ldots, m\}$, $m\in N$, such that $\mathbf{j}$ is assigned to processor $S_{CELL}(\mathbf{j})$, and

(2) S_{TIME}: $J^n\rightarrow N$ such that for every vertex $\mathbf{j}\in J^n$ we have

$$\forall\ \mathbf{i}\in IN(\mathbf{j})\ \ S_{TIME}(\mathbf{j})-S_{TIME}(\mathbf{i})\geq\begin{cases} p, & \text{if } S_{CELL}(\mathbf{i}) = S_{CELL}(\mathbf{j}) \\ p+c, & \text{if } S_{CELL}(\mathbf{i}) \neq S_{CELL}(\mathbf{j}) \end{cases},$$

where p is the processing time and c the communication delay.

● The *makespan* of a time schedule S_{TIME}, denoted $M(S_{TIME})$, is $\max\{S_{TIME}(\mathbf{j})+p \mid \mathbf{j}\in J^n\}$, where p is the processing time.

● Given the schedule S(A)=(S_{TIME}, S_{CELL}), we define N_{CELL} as $\max\{|\{\mathbf{j}\in J^n : S_{TIME}(\mathbf{j})=k\}| : 0\leq k\leq M(S_{TIME})\}$. ■

The makespan gives the completion time of the last task and, therefore, determines the time required for the completion of the whole algorithm. N_{CELL} gives the maximum number of cells required by the specific schedule. In our case, and for mapping onto systolic cells which operate synchronously without extra communication delays, we assume c=0 and p=1.

In order to achieve our final objective, which is to design optimal layouts, we must first achieve the following objectives:

(1) To find an *optimal time schedule* $S_{TIME_{OPT}}$, i.e., a schedule whose makespan is *minimum*.

(2) To establish the *optimal number of cells* $N_{CELL_{OPT}}$, i.e., the minimum number of systolic cells that are required to execute an optimal time schedule.

(3) To find an *optimal space schedule* $S_{CELL_{OPT}}$ that realizes $S_{TIME_{OPT}}$ using $N_{CELL_{OPT}}$ systolic cells from a multidimensional systolic structure.

A schedule ($S_{TIME_{OPT}}$, $S_{CELL_{OPT}}$) is called *optimal* and is denoted $S_{OPT}(A)$.

Theorem 2.1. For every algorithm and every time schedule S_{TIME} we have:

$M(S_{TIME}) = S_{TIME}(P_n)+1$.

Theorem 2.1 confirms our intuition that every loop will execute iteration P_n after all other iteration points have been completed. This implies that an optimal time schedule $S_{TIME_{OPT}}$ schedules P_n to be executed at the least possible time. In the following sections, we shall first establish the least possible time at which P_n can be correctly executed, and then the scheduling policy, which organizes the execution of the other nodes so as to achieve the optimal execution time for P_n.

3. OPTIMALITY IN TIME AND PROCESSORS

3.1. Theoretical Results

In this section we first calculate the optimal makespan and then we establish the optimal time schedule which achieves the optimal makespan. Finally, for the optimal time schedule we determine the optimal (i.e., the minimum) number of cells required to implement it. In [2] the following theorems have been shown.

Theorem 3.1. For every $\mathbf{j}=(k_1, \ldots, k_n)\in J^n$ $S_{TIME_{OPT}}(\mathbf{j})=k_1+ \ldots +k_n$.

Now, we can establish the optimal execution time for any uniform loop A.

Theorem 3.2. For every loop A with terminal point $P_n=(u_1, \ldots, u_n)$, $M(S_{TIME_{OPT}})=u_1+ \ldots +u_n+1$.

Definition 3.1. Let $\Pi(k)$, $0\le k\le u_1+ \ldots +u_n$, be the number of points of the loop that can be executed at instant k. Let Π_{MAX} be $\max\{\Pi(k) \mid 0\le k\le u_1+ \ldots +u_n\}$. ■

The following theorem gives the value of $\Pi(k)$.

Theorem 3.3. For every uniform loop with terminal point $P_n=(u_1, \ldots, u_n)$, $\Pi(k)=\binom{k+n-1}{n-1}+\sum_{r=1}^{n}(-1)^r\sum\binom{k+n-(u_1+1)-\ldots-(u_r+1)-1}{n-1}$, $0\le k\le u_1+ \ldots +u_n$.

A trivial consequence of Theorem 3.3 is the following corollary:

Corollary 3.1. For every uniform loop with terminal point $P_n=(u_1, \ldots,$

u_n), $\Pi_{MAX}=\Pi(\left\lfloor \frac{u_1+...+u_n}{2} \right\rfloor)$.

Theorem 3.4. For every uniform loop $N_{CELL_{OPT}}=\Pi_{MAX}$.

Every vertex **j**=(k_1, …, k_n) has a *maximal coordinate*, i.e., a coordinate k_i for which $k_i \geq k_r$, $1 \leq r \leq n$.

Theorem 3.5. Let u_i be a maximal coordinate of the terminal point $P_n=(u_1, \ldots, u_n)$. If $u_i \geq u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n$, then $\Pi(\left\lfloor \frac{u_1+...+u_n}{2} \right\rfloor)=(u_1+1)\times \ldots \times(u_{i+1}+1)\times(u_{i-1}+1)\times \ldots \times(u_n+1)$.

Theorem 3.6. Let u_i be a maximal coordinate of the terminal point $P_n=(u_1, \ldots, u_n)$. If $u_i \geq u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n$, then $\Pi_{MAX}=(u_1+1)\times \ldots \times(u_{i+1}+1)\times(u_{i-1}+1)\times \ldots \times(u_n+1)$.

In the rest of this paper we assume, without loss of generality, that u_1 is always a maximal coordinate of P_n.

3.2. Optimal Scheduling Policy

It is apparent that in order to achieve the optimal makespan, all vertices of the grid must be executed at the earliest possible time. This fact implies that the optimal time schedule is actually *unique*: $S_{TIME_{OPT}}(\mathbf{j})=k=k_1+ \ldots +k_n$, where **j**=($k_1$, …, k_n). This point can be executed by any of the required Π(k) processors for the k time instant. Thus, for every point of the grid we know in time independent of the size of the grid when and by which group of processors will be executed. In principle, we can find a schedule that will implement the optimal time schedule utilizing exactly $N_{CELL_{OPT}}$ cells (see [2] for such a schedule). However, although in that way the resulting schedule will be optimal in all accounts, it will not be suitable for mapping into systolic architecture, mainly due to complicated interconnection network among the processors. Therefore, what we propose here is a mapping that reflects the regular computation pattern of the algorithm and uses the least possible number of links: Every index point **j**=(k_1, …, k_n) is assigned to cell **c**=(k_2, …, k_n). If $P_n=(u_1, \ldots, u_n)$ is the terminal point of the loop, the above mapping requires exactly $(u_2+1)\times(u_3+1)\times \ldots \times(u_n+1)$. Optimality in terms of parallel time is always guaranteed and if $u_1 \geq u_2+u_3+ \ldots +u_n$ (recall that according to our assumption u_1 is a maximal coordinate of P_n), then optimality in terms of number of cells is also achieved. The worst case scenario with respect to use of redundant processors is when $u_1=u_2=u_3= \ldots =u_n$. The greatest advantage of this methodology is the simplicity of its implementation and its efficiency in terms of time complexity.

Example 3.1. Consider the following algorithm A:

```
program test;
int a[5][3][3];
int i1, i2, i3;
{
  for (i1=0; 4;)
   for (i2=0; 2;)
```

```
        for (i3=0; 2;) {
          a[i1][i2][i3]=a[i1][i2-1][i3]+a[i1][i2][i3-1]+
          a[i1-1][i2][i3];}
}
```

Fig. 2: The Algorithm A.

The dependence set DS is {(0, 1, 0), (1, 0, 0), (0, 0, 1)} and the index space J^3 is $\{(i_1, i_2, i_3) \mid 0 \le i_1 \le 4,\ 0 \le i_2 \le 2,\ 0 \le i_3 \le 2\}$ with $P_3=(4, 2, 2)$ (see Fig. 3). Theorem 3.3 gives $\Pi(k)=\binom{k+2}{2}-\binom{k-3}{2}-\binom{k-1}{2}-\binom{k-1}{2}+\binom{k-6}{2}+\binom{k-6}{2}+\binom{k-4}{2}-\binom{k-9}{2}$, $0 \le k \le 8$. As we see in Table 2, $\Pi_{MAX}=9$ for k=4, which means that $N_{CELL_{OPT}}=9$.

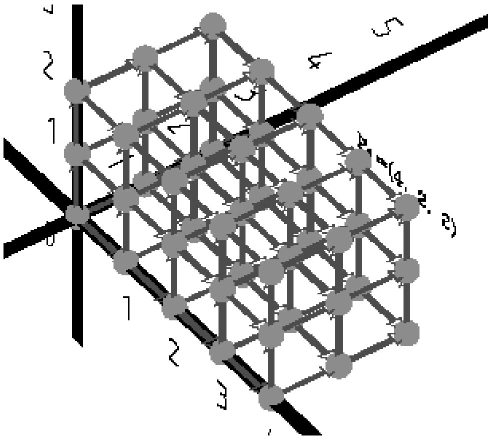

Fig. 3: The index space J^3.

Table 1: The cardinality of Π(k).

k	0	1	2	3	4	5	6	7	8
num of procs:	1	3	6	8	9	8	6	3	1

Table 2: The optimal schedule for A.

	0	1	2	3	4	5	6	7	8
c(0,0)	(0,0,0)	(1,0,0)	(2,0,0)	(3,0,0)	(4,0,0)				
c(0,1)		(0,0,1)	(1,0,1)	(2,0,1)	(3,0,1)	(4,0,1)			
c(0,2)			(0,0,2)	(1,0,2)	(2,0,2)	(3,0,2)	(4,0,2)		
c(1,0)		(0,1,0)	(1,1,0)	(2,1,0)	(3,1,0)	(4,1,0)			
c(1,1)			(0,1,1)	(1,1,1)	(2,1,1)	(3,1,1)	(4,1,1)		
c(1,2)				(0,1,2)	(1,1,2)	(2,1,2)	(3,1,2)	(4,1,2)	
c(2,0)			(0,2,0)	(1,2,0)	(2,2,0)	(3,2,0)	(4,2,0)		
c(2,1)				(0,2,1)	(1,2,1)	(2,2,1)	(3,2,1)	(4,2,1)	
c(2,2)					(0,2,2)	(1,2,2)	(2,2,2)	(3,2,2)	(4,2,2)

The schedule for the Algorithm A, which is sepicted in Table 2, is optimal in time and number of processors. The systolic array that implements this schedule is depicted in Fig. 4. ❐

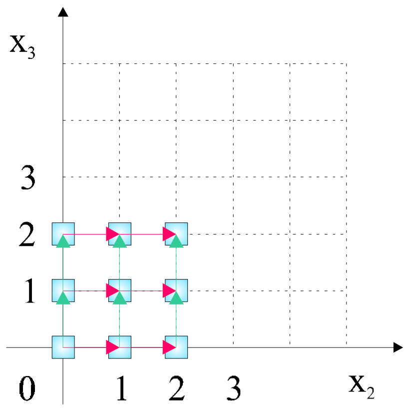

Fig. 4: The systolic mapping of A.

4. GENVHDL (VHDL PREPROCESSOR)

GENVHDL is a utility program developed to translate architectural descriptions into VHDL. By doing so, it provides a standard interface between the nested loop algorithmic tranformations presented earlier and current design automation tools. Great effort has been taken to make GENVHDL produce strictly structural descriptions, which define the exact placement of computation and communication elements of the proposed systolic architecture. This information is crucial for the VHDL synthesis phase implemented by design automation tools. Some practical results are presented in the following example.

GENVHDL accepts as inputs the original FOR loop statements and the transformation and dependence matrices. From these it constructs two VHDL files, one describing the structure of the basic systolic cell and one describing the map of the hole architecture. Computational elements are constructed by mapping primitive arithmetic operations (additions, multiplications etc.) into appropriate primitive hardware elements. The transformed dependencies are satisfied by implementing the appropriate links among the cell of the systolic architecture and by inserting delay elements, when needed. All elements are connected together in the map file.

Example 4.1. (Example 3.1 continued).

GENVHDL was used to produce the VHDL systolic hardware description corresponding to the algorithm A, presented in Example 3.1. The outcome was used as the input specification in VIEWlogic ViewSynthesis VHDL synthesis tool. The resulting schematic was given to XILINX XACTStep to produce an FPGA implementation. A floorplan of such an implementation can be seen in Fig. 5 (device used XC4010).

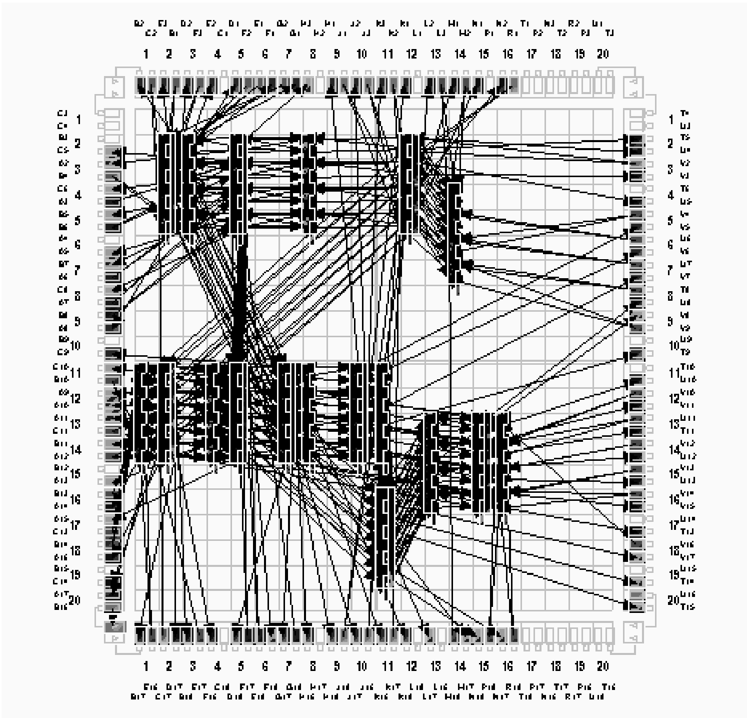

Fig. 5: FPGA Implementation.

For evaluation purposes, the same algorithm, in behavioral VHDL, where each iteration is expressed as a concurrent signal assignment, was given as input to the above tools. The resulting implementation uses twice as many logic elements as the one produced using GENVHDL (see comparisons in Table 3). This is a very promising result for our methodology. However, it should not be overestimated since none of the VHDL descriptions was specially written for FPGA synthesis. ❐

Table 3: XC4013 Comparisons.

	GENVHDL	Beh. VHDL
FGs	63	135
FGHs	-	45
DFFs	18	-
IOBs	122	169

Table Key:

FGs - 4 input Programmable Logic Blocks

FGHs - 3 input Programmable Logic Blocks

DFFs - D type Flip Flops
IOBs - Programmable I/O Blocks

5. CONCLUSION - FUTURE WORK

In this paper we have presented a complete methodology and a tool for optimal mapping of uniform nested loops with unit dependencies into multidimensional systolic arrays. We have given the theoretical framework for the optimal schedule and mapping which achieves the maximum cells utilization for a subclass of index spaces with specific bounds. Future work includes the optimal mapping into fixed size arrays and mapping into lower dimensional arrays.

REFERENCES

1. Andronikos, T., Koziris, N., Tsiatsoulis, Z., Papakonstantinou, G., and Tsanakas, P. Lower Time and Processor Bounds for Efficient Mapping of Uniform Dependence Algorithms into Systolic Arrays. To appear in *Journal of Parallel Algorithms and Applications.* 10, 3-4, 1997.
2. Andronikos, T., Koziris, N., Papakonstantinou, G., and Tsanakas, P. Optimal Scheduling for UET/UET-UCT Generalized n-Dimensional Grid Task Graphs, submitted to Journal of Parallel and Distributed Computing.
3. Bampis, E.,. Delorme, C., and Konig, J.C. Optimal Schedules for d-D Grid Graphs with Communication Delays. *Symposium on Theoretical Aspects of Computer Science (STACS96*). Grenoble France 1996.
4. Koziris, N., Papakonstantinou, G., and Tsanakas, P. Automatic Loop Mapping and Partitioning into Systolic Architectures. *Proceedings of the 5th Panhellenic Conference on Informatics*, Dec. 1995, pp. 777-790, Athens
5. Koziris, N., Papakonstantinou, G., and Tsanakas, P. Optimal Time and Efficient Space Free Scheduling For Nested Loops. *The Computer Journal*. 39, 5, pp. 439-448, 1996.
6. Lamport, L. The Parallel Execution of DO loops. *Commun. ACM*, vol.17, no.2, pp. 83-93, Feb. 1974.
7. Lee, P.-Z. and Kedem, Z.M. Mapping Nested Loop Algorithms into Multidimensional Systolic Arrays. *IEEE Trans. Parallel Distrib. Syst*., vol. 1, no. 1, pp. 64-76, Jan. 1990.
8. Moldovan, D.I. and Fortes, J.A.B. Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays. *IEEE Trans. Comput*., vol C-35, no 1, pp. 1-11, Jan. 1986
9. Moldovan, D.I. ADVIS: A Software Package for the Design of Systolic Arrrays, *IEEE Trans. Computer Aided Design*, vol CAD-6, no 1, pp. 33-40, Jan. 1987.
10. Shang, W. and Fortes, J.A.B., Time Optimal Linear Schedules for Algorithms with Uniform Dependencies, *IEEE Trans. Comput*., vol. 40, no. 6, pp. 723-742, June 1991.
11. Shang, W. and Fortes, J.A.B. On Time Mapping of Uniform Dependence Algorithms into Lower Dimensional Processor Arrays. *IEEE Trans. Parallel Distrib. Syst*., vol. 3, no. 3, May 1992.