Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2012, Article ID 196761, 9 pages
doi:10.1155/2012/196761

Research Article

Multidimensional Costas Arrays and Their Enumeration Using GPUs and FPGAs

Rafael A. Arce-Nazario and José Ortiz-Ubarri

Department of Computer Science, University of Puerto Rico, Río Piedras, San Juan, PR 00924, USA

Correspondence should be addressed to Rafael A. Arce-Nazario, rafael.arce@upr.edu

Received 4 May 2012; Accepted 17 September 2012

Academic Editor: René Cumplido

The enumeration of two-dimensional Costas arrays is a problem with factorial time complexity and has been solved for sizes up to 29 using computer clusters. Costas arrays of higher dimensionality have recently been proposed and their properties are beginning to be understood. This paper presents, to the best of our knowledge, the first proposed implementations for enumerating these multidimensional arrays in GPUs and FPGAs, as well as the first discussion of techniques to prune the search space and reduce enumeration run time. Both GPU and FPGA implementations rely on Costas array symmetries to reduce the search space and perform concurrent explorations over the remaining candidate solutions. The fine grained parallelism utilized to evaluate and progress the exploration, coupled with the additional concurrency provided by the multiple instanced cores, allowed the FPGA (XC5VLX330-2) implementation to achieve speedups of up to 30× over the GPU (GeForce GTX 580).

1. Introduction

A two-dimensional Costas array (2DCA) of size N is a permutation $f : N \to N$ such that for every integer h, i, and j, where $1 \le h \le N-1$ and $1 \le i, j \le N-h$, $f(i+h) - f(i) = f(j+h) - f(j)$, implies that $i = j$. Thus, informally, a size N Costas array is defined as $N \times N$ matrix containing exactly N dots, where every row and column contain exactly one dot and the vectors joining each pair of dots are distinct. Figure 1 illustrates a Costas array of size $N = 6$, both as a matrix and a permutation. The figure also shows the array's difference triangle, which organizes the differences between the various permutation digits in $N - 1$ rows where each row corresponds to a fixed h. By definition, each row in the difference triangle of a Costas array must consist of unique numbers.

Their definition implies that Costas arrays have perfect autocorrelation (autocorrelation = 1), which makes them useful in communications where signals must be recoverable even in the presence of considerable noise. Costas arrays are useful in many security and communication applications, such as remote object recognition and optical communications [1]. Furthermore, some special cases of Costas arrays can be used for digital watermarking [2].

Costas arrays with dimensions higher than two were introduced in 2008 by Drakakis [3]. These arrays maintain perfect autocorrelation, which broadens their applicability in optical communications, for example, 3D optical orthogonal codes [4]. Multidimensional periodic Costas arrays (MPCAs) over elementary Abelian groups, introduced by Moreno and Tirkel , add the property of being periodic over all dimensions. This extends their applicability to digital watermarking of video and combined video and audio, where higher-dimensionality codes are desired [2]. A formal definition and some of their properties are presented in Section 3. In this paper, we focus on the latter kind of multidimensional periodic Costas arrays (MPCAs) due to their richer application range.

The enumeration of 2D Costas arrays has been a topic of interest since their discovery by Costas in the 1960s [5]. With each new size enumerated, new properties and generation techniques may be discovered [6]. Ortiz-Ubarri et al. presented MPCA transformations and their first enumeration in [7]. Given their relatively new discovery, it is expected that the enumeration of MPCAs will, as with 2DCAs, help researchers improve their understanding.

Both 2DCAs and MPCAs can be generated using algebraic constructions based on finite fields like the Welch and Lempel-Golomb constructions [7]. Small sizes can be enumerated using hand computation, yet the only known method to guarantee complete enumeration is by exhaustive exploration. The search space for complete enumeration of 2DCAs grows factorially with N, thus computer-based exploration is the only practical approach for medium and large sizes. The most common approach for the enumeration of 2DCAs is to use a backtracking algorithm that incorporates symmetries and other observations to further prune the search space. This paper presents the first discussion of search space pruning techniques for MPCAs. Both the FPGA and GPU implementations utilize these methods to significantly reduce the search space. Nevertheless, the worst case timecomplexity is still factorial, requiring tremendous run times even for small cases of N and m (the dimension).

This paper discusses our implementations for the enumeration of $(m+1)$-dimensional Costas arrays in GPUs and FPGAs and constitutes the first description of such an enumeration. We present the techniques chosen to prune the search space as well as the organization of our designs. Our FPGA implementation achieved speedups of up to 30 times faster than the GPU. Furthermore, the modules that were created as part of the design process can easily be adapted to other constraint satisfaction problems that use backtracking.

The rest of this paper is organized as follows. Section 2 presents the relevant previous work, while Section 3 defines $(m+1)$-dimensional Costas arrays and some of their symmetries. Section 4 describes the backtracking algorithm and its use for the enumerations. Section 5 introduces several techniques that allow us to prune the search space during the enumerations. Sections 6 and 7 discuss the GPU and FPGA designs, respectively. Section 8 reports and discusses the results and Section 9 provides our conclusions.

2. Previous Work

Most recent enumerations of 2DCAs have been completed using general purpose processors [8, 9]. The latest enumerations have been achieved by deploying many computer clusters (in all, thousands of cores) to concurrently explore disjoint parts of the search space. For $N = 28$ and $N = 29$, the time per single CPU was determined to be 70 and 366.55 years, respectively.

To the best of our knowledge, the only reported FPGA-accelerated 2DCA enumerations have been [10, 11]. Devlin and Rickard implemented sizes $N = 13$ through 19 on a Xilinx Spartan-3 XC3S1000 running at 25 Mhz and extrapolated their results to sizes up to $N = 32$ using the same device as well as the Virtex-5 XC5VLX110 [10]. The extrapolated execution times for $N = 28$ and 29 were 9.26 and 48.45 years, respectively. Arce-Nazario and Ortiz-Ubarri compared the execution of 2DCA enumeration in an FPGA (Virtex 5-XC5VLX330-2) and a GPU (GeForce GTX 580). The FPGA implementation achieved speedups of up to 40× over the GPU and 4.44× over the fastest reported software implementation [9].

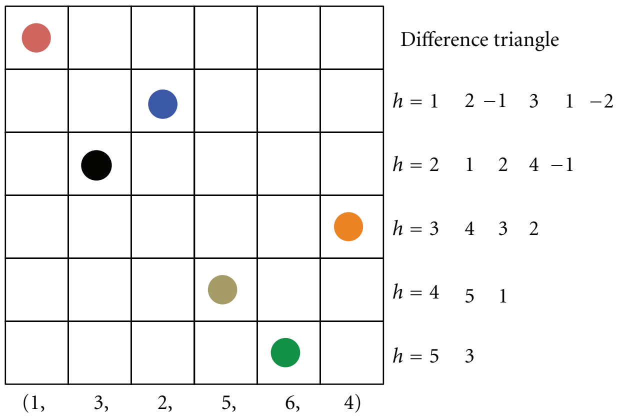


FIGURE 1: The Costas array (1, 3, 2, 5, 6, and 4), its matrix and permutation representations. The difference triangle confirms that this is a Costas array since each row h consists of unique numbers.

The first generalizations of Costas arrays were given by Drakakis in [12]. He defined various classes of multidimensional arrays by modifying the Costas property constraints and presented examples. Moreno et al. defined the $(m+1)$-dimensional periodic Costas arrays over elementary Abelian groups [13] and described algebraic Welch Costas constructions. More recently, Ortiz-Ubarri et al. presented symmetries over the MPCAs that allow the expansion of the families discovered by Moreno [7]. They reported the first enumeration for sizes $(\mathbb{Z}_2)^2$, $(\mathbb{Z}_3)^2$, and $(\mathbb{Z}_5)^2$, yet no details are offered regarding enumeration or solution space pruning techniques.

3. $(m+1)$-Dimensional Costas Arrays

We begin by providing the definitions of the $(m+1)$-dimensional periodic Costas arrays over elementary Abelian groups.

Definition 1. A generic $(m+1)$-dimensional periodic Costas array over the elementary Abelian group $(\mathbb{Z}_p)^m$ is a permutation function $f : ((\mathbb{Z}_p)^m)^* \rightarrow (\mathbb{Z}_{p^m-1})$, where A^* means A-$\{0\}$. This function has the distinct difference property: for any $h \neq 0$, $a, b \in (\mathbb{Z}_p)^m$, $f(a+h) - f(a) = f(b+h) - f(b)$ implies $a = b$, where the addition and subtraction operations are performed in the corresponding Abelian group.

Remark 2. Since, by definition, the $(m+1)$-dimensional periodic Costas arrays over the elementary Abelian group are fully periodic; the periodic shifts of an MPCA on any of its $(m+1)$ dimensions result in a different $(m+1)$-dimensional periodic Costas array over the elementary Abelian group.

Example 3. The following is a grid defined over $\mathbb{Z}_3 \times \mathbb{Z}_3$:

$$\mathbf{W} = \begin{pmatrix} w_{2,0} & w_{2,1} & w_{2,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{0,0} & w_{0,1} & w_{0,2} \end{pmatrix}. \quad (1)$$

As a shorthand method, we may also enumerate the elements in a Costas array $(\mathbb{Z}_p)^m$ using the index mapping $(d_0, \ldots, d_{m-1}) \mapsto (d_0 + p \cdot d_1 + \cdots + p^{m-1} \cdot d_{m-1})$

$$\mathbf{W} = \begin{pmatrix} w_6 & w_7 & w_8 \\ w_3 & w_4 & w_5 \\ w_0 & w_1 & w_2 \end{pmatrix}. \tag{2}$$

The distinct difference property can be verified, in a manner similar to 2DCAs, by using difference matrices. The differences for a $(\mathbb{Z}_p)^m$ array are organized into $p^m - 1$ matrices. For instance, for $m = 2$, each difference matrix $h = (h_0, h_1)$ is $p \times p$ and contains at each cell (i, j) the result of $w_{i+h_0, j+h_1} - w_{i,j}$. The cells for differences that involve position $(0, 0)$ are represented using $*$.

Figure 2 shows a $\mathbb{Z}_3 \times \mathbb{Z}_3$ Costas array along with its corresponding difference matrices. For example, cell $(1, 1)$ of the difference matrix $h = (0, 1)$ contains the difference between $w_{(1+0),(1+1)} - w_{1,1} = w_{1,2} - w_{1,1} = 7 - 2 = 5$. A MPCA satisfies the distinct difference property if each of its $p^m - 1$ difference matrices contain each number in $\mathbb{Z}_{p^m-1}$-$\{0\}$ exactly once.

3.1. Addition and Multiplication (Modulo $p^m - 1$) Symmetries. Two algebraic symmetries introduced by Moreno et al. can be used to significantly reduce the search space for MPCA enumeration [13].

Theorem 4. *Multiplication (modulo $p^m - 1$) of a periodic Costas array by an integer less than and relatively prime to $p^m - 1$ generates a new periodic Costas array.*

Example 5. Multiplying $\mathbf{W}$, the array in Figure 2, by $7 \equiv -1 \bmod 8$ yields the following MPCA:

$$\begin{pmatrix} 3 & 5 & 2 \\ 7 & 6 & 1 \\ * & 0 & 4 \end{pmatrix}. \tag{3}$$

Theorem 6. *Addition (modulo $p^m - 1$) of any integer less than $p^m - 1$ to a periodic Costas array generates a new periodic Costas array.*

Example 7. Adding 3 to $\mathbf{W}$ yields the MPCA:

$$\begin{pmatrix} 0 & 6 & 1 \\ 4 & 5 & 2 \\ * & 3 & 7 \end{pmatrix}. \tag{4}$$

4. Backtracking

Backtracking is a general algorithm for solving a computational problem by incrementally generating all possible solutions. The execution of a backtracking algorithm can be modelled as a search tree where every node is a partial solution. Moving forward corresponds to approaching a valid solution, and going backward corresponds to abandoning a partial candidate that cannot possibly generate a valid solution.

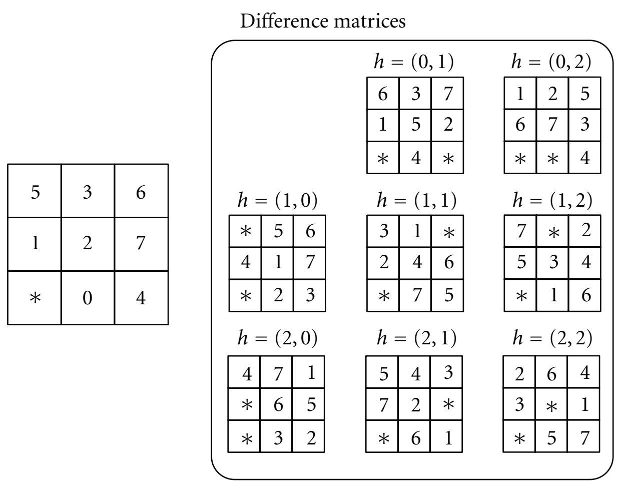


FIGURE 2: A $\mathbb{Z}_3 \times \mathbb{Z}_3$ MPCA and its difference matrices. The $*$ in the MPCA symbolizes that the mapping for $(0, 0)$ is not defined, that is, recall the mapping is $f : ((\mathbb{Z}_p)^m)^* \to (\mathbb{Z}_{p^m-1})$.

For the purpose of this discussion, we define a subpermutation $P_\ell^X = (p_1, p_2, \ldots, p_\ell)$, where $p_i \in X$. For MPCAs, $X = \mathbb{Z}_{p^m-1} + \{*\}$. The next subpermutation of size $\ell + k$ of P_ℓ^X, where $k \in \{-1, 0, 1\}$, expressed as $\aleph(P_\ell^X, \ell + k)$ is defined as the next subpermutation in lexicographical order of size $\ell + k$ that conserves the first $\ell + k - 1$ elements.

Example 8. For $X = \mathbb{Z}_{3^2-1} + \{*\}$, let $P_4^X = (*, 0, 4, 3)$, the next subpermutation of size 4, $\aleph(P_4^X, 4) = (*, 0, 4, 5)$. The next subpermutation of size 5, $\aleph(P_4^X, 5) = (*, 0, 4, 3, 1)$. The next subpermutation of size 3, $\aleph(P_3^X, 3) = (*, 0, 5)$.

Example 9. For $X = \mathbb{Z}_{3^2-1} + \{*\}$, let $P_4^X = (*, 0, 4, 7)$. $\aleph(P_4^X, 4) = \epsilon$, that is, there is no next subpermutation beginning with $(*, 0, 4)$. $\aleph(P_4^X, 5) = (*, 0, 4, 7, 1)$. $\aleph(P_4^X, 3) = (*, 0, 5)$.

Algorithm 1 illustrates the backtracking algorithm used for enumerating all MPCAs in $(\mathbb{Z}_p)^m$ given a seed permutation P_{init}. Figure 3 illustrates the steps in the computational tree of the backtracking approach, given the seed $(*, 0, 4, 1, 2, 7)$ for $(\mathbb{Z}_3)^2$.

5. Techniques for Pruning the Search Space and Efficient Evaluation of Candidate Arrays

MDCA symmetries can be leveraged to reduce the search space in their enumeration. For instance, it can be deduced from Theorems 4 and 6 that backtracking exploration must proceed only through permutations lexicographically smaller than $(*, 0, \lfloor p^m/2 \rfloor, \lfloor p^m/2 \rfloor + 1, \ldots)$.

(1) MPCAs with the $*$ in any position other than $(0, 0)$ are generated by periodic shifts of the arrays with the $*$ in position $(0, 0)$. These include all the geometric symmetries (horizontal flip, vertical flip, and 90^0 rotations) of the MPCAs.

```
Inputs: $p, m$: MPCA dimensions
        $P_{\text{init}}$: initial permutation
Output: Displays all MPCAs in $(\mathbb{Z}_p)^m$
Perm = Perm$_{\text{init}}$
$\ell$ = length of Perm$_{\text{init}}$
while ($\ell$ >= length of Perm$_{\text{init}}$){
    if (IsCostas (Perm) {
        if ($\ell == p^m$) {
            display Perm
            do {
                $\ell--$;
            } until (Next (Perm, $\ell$)!= empty)
        }
        else
            $\ell$++; //explore a deeper level
    }
    else {
        // backtrack
        do {
            $\ell--$;
        } until (Next (Perm, $\ell$)!= empty)
    }
    Perm = Next (Perm, $\ell$);
}
```

ALGORITHM 1: Backtracking algorithm to enumerate all MPCAs in $(\mathbb{Z}_p)^m$ beginning with permutation Perm$_{\text{init}}$.

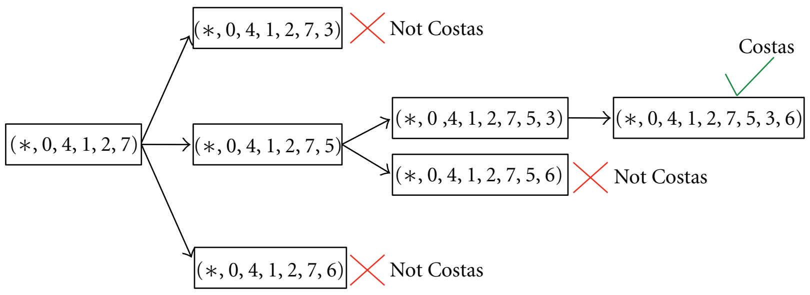


FIGURE 3: Backtracking search tree for MPCAs in $(\mathbb{Z}_3)^2$ with $P_{\text{init}} = (*, 0, 4, 1, 2, 7)$.

(2) Any permutation starting with $*$, 0, and followed by a $\lfloor p^m/2 \rfloor$, $\lfloor p^m/2 \rfloor + 1$ can be obtained by multiplying a lexicographically smaller subpermutation by -1, that is, using Theorem 4. For example, for $(\mathbb{Z}_3)$, any permutation including and higher than $(*, 0, 4, 5, 1, 2, 3, 6, 7)$ can be obtained by multiplying a smaller permutation by -1, for instance, $(*, 0, 4, 5, 1, 2, 3, 6, 7)$ by multiplying $(*, 0, 4, 3, 7, 6, 5, 2, 1)$ by -1.

(3) Any permutation starting with $*$, followed by a nonzero element can be obtained by adding to a permutation that starts $*, 0$, that is, using Theorem 6. For example, $(*, 1, 5, 4, 0, 7, 6, 3, 2)$ is obtained from the addition of 1 to $(*, 0, 4, 3, 7, 6, 5, 2, 1)$.

Thus, the required exploration is reduced from $(p^2 - 1)!$ to approximately $(p^2 - 2)!/2$ permutations in the worst case.

5.1. Evaluation of Candidate Arrays. Computationally, we determine if a permutation is an MPCA by using the distinct difference property. In the backtracking algorithm, every time the algorithm moves forward by adding a new element p_k to the permutation, the new differences generated by subtracting p_k and $p_1, \ldots, p_{k-1}$ are added to the corresponding arrays. Thus, the difference arrays fill up as the permutation in the backtracking tree grows and deplete as the backtracking algorithm moves backward.

From the MPCA definition it can be deduced that only half of the difference matrices must be maintained. To understand this, let us define the negative of a distance vector.

Definition 10. Let $h = (h_0, \ldots, h_{m-1})$ be a distance vector of $(m+1)$-dimensional periodic Costas array over the elementary Abelian group $(\mathbb{Z}_p)^m$. The negative of the distance vector h, expressed as $-h$, is defined as $(-h_0, \ldots, -h_{m-1})$. In other words, h is a vector in the direction $(h_0, \dots, h_{m-1})$ then $-h$ is a vector of same length in the opposite direction.

Example 11. The negative of the difference vector $h = (1, 0)$ over the elementary Abelian group $(\mathbb{Z}_3)^2$ is $-h = (-1, 0) = (2, 0)$.

The difference matrix for a distance vector $-h$ accumulates the negatives of the differences collected by the matrix for the distance vector h. This is illustrated in Figure 4. The current subpermutation being evaluated $(*, 0, 4, 1, 2, 7)$ has produced the differences shown in the matrices. Notice how the differences in $h = (0, 1)$ and $h = (0, 2)$ are the negatives of each other, for example, $-4 = 4$, $-1 = 7$, $-5 = 3$, and $-2 = 6$. The same behavior is obtained for the rest of the h and $-h$ pairs, for example, $(0, 1)$ and $(0, 2)$, $(1, 1)$ and $(2, 2)$, and $(2, 1)$ and $(1, 2)$. Therefore, we only need to keep track of either the h or $-h$ of each $h, -h$ pair, that is, the other matrix contains redundant information.

Furthermore, using the index mapping $(d_0, \dots, d_{m-1}) \mapsto (d_0 + p \cdot d_1 + \cdots + p^{m-1} \cdot d_{m-1})$, we can demonstrate that the $(p^m - 1)/2$ matrices can be completed by computing all $w_b - w_a$, $a, b \in \mathbb{Z}_{p^m-1}$ where $b > a$.

Theorem 12. *The differences $w_b - w_a$, $a, b \in \mathbb{Z}_{p^m-1}$, where $b > a$ complete all the differences matrices.*

Proof. Without loss of generality, we consider $m = 2$, that is, $a, b \in \mathbb{Z}_{p^2-1}$. The matrix (h_0, h_1) collects all differences $f((i, j) + (h_0, h_1)) - f(i, j)$. If $(i + h_0) + p \cdot (j + h_1) > i + p \cdot j$ then $b > a$. Else, $b < a$, that is, $(i + h_0) + p \cdot (j + h_1) < i + p \cdot j$ which implies one of two cases.

(1) $j > j + h_1$. This implies that $h_1 \geq p - j$, in which case, for the negative, $h_1 < p - j$ and $j < j + h_1$. Thus, the negative of this case can be found using a difference covered by $b > a$.

(2) $j = j + h_1$ and $h_0 > i + h_0$. This implies that $h_0 \geq p - h_0$, in which case, for $-h$, $h_0 < p - i$ and $i < i + h_0$. Thus, the negative of this case can be found using a difference covered by $b > a$. □

Example 13. For the difference matrix $h = (0, 1)$ the computation for $f((2, 2) + (0, 1)) - f(2, 2)$, that is, $f(2, 0) - f(2, 2)$ can be obtained from $-(f(2, 2) - f(2, 0))$, that is, $-(w_8 - w_6)$.

In our implementations, the difference matrices are managed as follows.

(i) A hash table of size $p^m - 1$ is used for each of the $(p^m - 1)/2$ matrices to keep track of its differences.

(ii) Whenever the permutation length increases (P^X_ℓ to $P^X_{\ell+1}$), the differences between the last added digit and the rest of the digits are computed and compared to the contents of the corresponding hash tables. A hit in any of the tables indicates a repeated difference and hence a non-Costas permutation. Otherwise, the differences are registered in the corresponding tables.

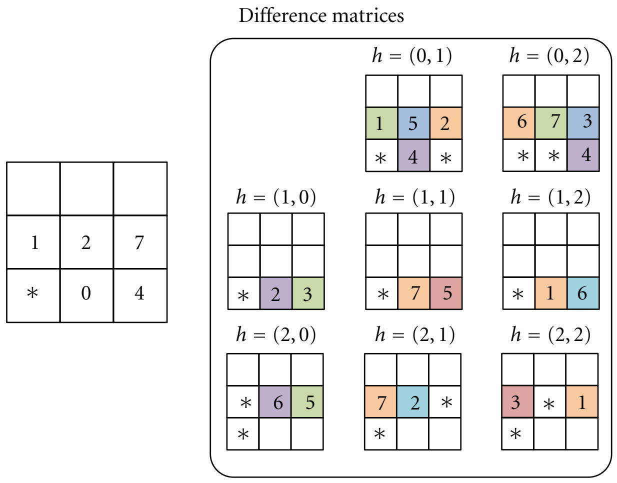


Figure 4: Difference matrices for a subpermutation during enumeration of $\mathbb{Z}_3 \times \mathbb{Z}_3$ MPCAs, to illustrate the behavior of h and $-h$ matrix pairs.

(iii) Whenever the permutation length decreases (P^N_ℓ to $P^X_{\ell-1}$), the differences between the dropped digit and the rest of the digits are deleted from the hash tables.

6. GPU Design

We perform our computations in a GeForce GTX 580 with 16 multiprocessors, each with 32 cores at 1.54 GHz clock rate, 1.5 GB of global memory, and 48 K of shared memory using the CUDA parallel computing architecture. Similar to many other computational problems where GPUs are used to accelerate algorithms in parallel, our implementation essentially decomposes the enumeration into many disjoint subspaces, which are deployed as threads to the GPU. Figure 5 illustrates the workflow of the GPGPU implementation. The Host (CPU) generates a set of K subpermutations P^N_m of size $m < N$ that comply with the Costas property. The set is then passed to the Device (GPU), where for each subpermutation a thread is generated to complete the exploration, that is, each thread determines all (if any) Costas arrays that begin with its given subpermutation. While the threads are executing, the Host is generating the next set of K subpermutations. When all threads complete, the results are passed to the Host, and the next set of K subpermutations is transferred to the GPU.

Two quite similar versions of Algorithm 1 run in the Host and each of the threads of the GPU. In the Host, Algorithm 1 is used to generate all subpermutations of length M compliant with the Costas property. As each subpermutation is found it is added to an array of size K of the data type shown in Algorithm 2 . When the array is full it is copied to the GPU global memory and the K CUDA threads are deployed to process the copied subpermutations.

Each CUDA thread runs a version of Algorithm 1 that takes one of the subpermutations as P_{init} and copies any found Costas arrays back to the GPU global memory. When all the GPU threads are done, the found Costas arrays of size

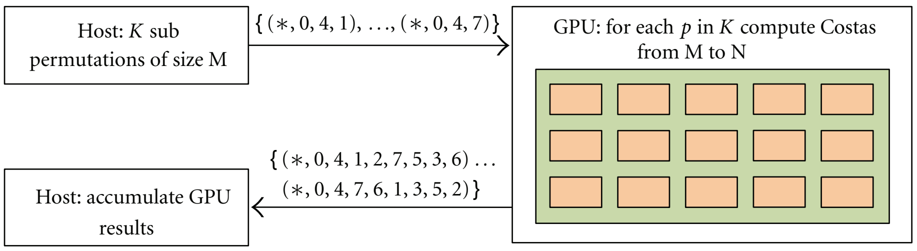

Host: K sub permutations of size M
{(∗, 0, 4, 1), …, (∗, 0, 4, 7)}
GPU: for each p in K compute Costas from M to N
{(∗, 0, 4, 1, 2, 7, 5, 3, 6) … (∗, 0, 4, 7, 6, 1, 3, 5, 2)}
Host: accumulate GPU results

Figure 5: Simplified workflow of the GPGPU implementation. The Host generates a list of subpermutations, the GPU completes the search for the full permutations, then sends them back to the Host. The process repeats until no more subpermutations exist.

```
typedef struct{
    char counter;
    char subcostas[M];
    char costas[MAX_COSTAS_PT][N–M];
} CostasData;
```

Algorithm 2: The array of K permutations used between the Host and GPU is an array of size K of type CostasData counter stores the number of Costas array of size N found in each thread of the GPU, subcostas stores one of the subpermutations of size M computed in the Host, and costas stores a maximum number of the remaining $(N–M)$ elements of the Costas arrays of size N found in each thread of the GPU.

N are copied to the Host. The number of subpermutations (K) to be passed to the GPU is determined by the number of cores of the GPU. In our experiments we obtained the best performance with K = number of cores $\times$ 128.

7. FPGA Design

Many FPGA implementations obtain their impressive performance by exploiting fine-grained parallelism through deep, custom pipelines. However, backtracking is in essence a serial process, that is, generates permutation, then evaluates, then accepts or backtracks. Given this scenario, we opted to implement a highly tuned, low-resource serial MPCA enumeration (MPCAEn) core that can be instantiated many times inside the FPGA. Hence, the acceleration provided by our design comes mainly from two factors: (a) the rapid generation of candidate permutations and their evaluation within each core and (b) the high number of cores working in parallel on subsets of the enumeration.

7.1. Backtracking Functionality and Array Evaluation. Figure 6 illustrates the basic blocks of our MPCAEn core. A shift register is used for constructing or reversing the candidate permutation. The candidate permutation is constructed by shifting left and concatenating a new digit in the right-most position. As a permutation is generated, its compliance with the Costas array definition is verified by the Costas evaluation block. If the candidate complies, the next permutation P^N_{m+1} is generated by shifting left and concatenating the lowest available digit to the right-most position. If not, then one of two cases may occur.

(1) There is an available digit d that is higher than the rightmost digit. In this case, the rightmost digit is substituted by d. For example, $(\ast, 0, 4, 1, 2, 6)$ is evaluated and does not comply. Since the available digits are 3, 5, and 7, then 6 is substituted by 7 to compose the next candidate permutation $(\ast, 0, 4, 1, 2, 7)$.

(2) There are no digits available higher than the rightmost digit. In this case the shift register is shifted right and the digit that is shifted out is added to the available digits. This is repeated until there is an available digit that is higher than the current rightmost digit, at which case we perform the substitution described in the first case. For example, $(\ast, 0, 4, 3, 7)$ is determined to not comply with the MPCA difference property; the available digits are $1, 2, 5,$ and 6 but none of them is higher than 7 so the permutation must backtrack to $(\ast, 0, 4, 3)$. The available digits are now $1, 2, 5, 6,$ and 7, thus 3 is substituted by 5 to form the next permutation $(\ast, 0, 4, 5)$.

Figure 7 illustrates the operation of the MPCA evaluation block. When a new digit is added to the current permutation, for example, p_4 in the figure, the differences between the new digit and the rest are computed. The negatives of the differences are also computed, since they might be used to update some of the difference matrices (as explained in Section 5.1). Depending on the index of the newly added digit, the encoder/mux block routes the results of the differences to the corresponding hash tables. For the example in the figure, the result from Diff$_2$ corresponds to $p_4 - p_2$, that is, $p_{(1,1)} - p_{(0,2)}$; thus it will update the hash table for the difference matrix $h(1, 2)$. When p_5 is added to the permutation, Diff$_2$ corresponds to $p_5 - p_3$, that is, $p(1, 2) - p(1, 0)$; thus its negative will be used to update the hash table for the difference matrix $h(0, 1)$.

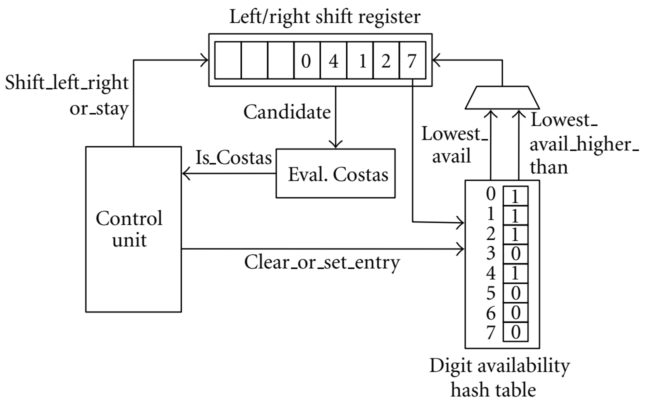


FIGURE 6: In the illustration, the permutation $(*, 0, 4, 1, 2, 7)$ has just been generated. The hash table keeps record of the digits that are in use in the current permutation (indicated by 1) and outputs two digits, the lowest available and the lowest available higher than the current input.

For 2DCAs, the deeper rows of the difference triangle are responsible for the detection of only a small percentage of the rejected arrays, as confirmed in [10, 11]. Our enumeration core for MPCAs is parameterizable in the number of distance vectors that are used to evaluate whether an array is an MPCA. The designer may choose to implement less than $(p^2 - 1)/2$ distance vectors to save FPGA resources. If so, the few false positives that will come out from the FPGA enumeration can be eliminated in software once they are communicated to the general purpose processor (GPP).

7.2. Resource Utilization and Core Organization. Figure 8 shows the amount of FPGA LUTs required by the MPCAEn core for $(\mathbb{Z}_5)^2$ and $(\mathbb{Z}_7)^2$, implementing various amounts of vector differences. LUTs are the most strained resource in the MPCAEn implementation (versus registers). The resource utilization results were obtained from the synthesis process using Xilinx ISE 11.5 targeting a Virtex5-XC5VLX330-2 FPGA. It was found through experimentation that keeping track of more than 7 differences for $(\mathbb{Z}_5)^2$ and 14 differences for $(\mathbb{Z}_7)^2$ did not significantly reduce enumeration times. The enumerators that were implemented for obtaining the results used those amounts of differences, for example, 7 for $(\mathbb{Z}_5)^2$ and 14 for $(\mathbb{Z}_7)^2$.

Since resource utilization per enumerator is low, multiple MPCAEn modules were instantiated in the FPGA as illustrated in Figure 9. The transfer of subpermutations and collection of results is performed by the transfer/collector module, which is connected through low-width data lines to the enumerators in order to save connection resources. MPCAs are so scarcely found during the enumeration that, regardless of the bandwidth between enumerators and transfer/collector module, these connections never become a bottleneck.

8. Results and Discussion

To compare GPU and FPGA performance, we implemented sizes 3×3, 5×5, and 7×7 of our MPCA enumeration designs in GPU (GeForce GTX 580) and FPGA (one Xilinx XC5VLX330-2 device of the four provided in a Convey HC-1 Server). Table 1 shows the number of found MPCAs starting with $(*, 0)$, the execution times for both designs as well as the growth rate as a function of p, and the speedup of FPGA versus GPU. Results for 3×3 and 5×5 are wall clock times while 7×7 is the worst case estimation of the run time based on sample runs.

We attribute the speedup mainly to the fact that the FPGA implementation was able to exploit the following two levels of parallelism, whereas the GPU could only make use of the highest level:

(1) *coarse-grained level* parallelism, which is achieved by splitting the solution space into multiple disjoint sets;

(2) *fine-grained* parallelism at the level of individual candidate evaluations, that is, the operations for the evaluation of each (sub)permutation are performed in parallel (as discussed in Section 7 and illustrated in Figure 7). An analogous, low-level technique could be used in general purpose processors by utilizing SIMD instructions. However, CUDA programs compile to the PTX instruction set, which does not contain SIMD instructions.

The GPU $(\mathbb{Z}_3)^2$ was greatly overshadowed by the data transfer times between FPGA/GPP; therefore, we only consider for fair comparisons the cases $(\mathbb{Z}_5)^2$ and $(\mathbb{Z}_7)^2$. For these larger cases we observe FPGA versus GPU speedups similar to those reported for 2DCAs in [11]. The enumeration of MPCAs exhibits a slower growth rate (approximately 3) per additional permutation digit as compared to the reported for two-dimensional Costas arrays (approximately 5) [8, 11]. We conjecture that the reason for the slower growth rate is that the conditions imposed in the MPCA definition are more strained, thus eliminating more candidate subpermutations earlier than the case of 2D Costas arrays.

All enumerated MPCAs turned out to be either Welch Costas constructions as presented in [13] or their symmetries introduced in [7], that is, no spurious MPCAs were found similar to some sizes of 2DCAs. These results support the conjecture that MPCAs (of all sizes and dimensions) are fully characterized by multidimensional Welch Costas arrays along with their symmetries.

9. Conclusions

In this work, we presented designs for the enumeration of multidimensional periodic Costas arrays in GPUs and FPGAs. Also, we introduced several MPCA symmetries and showed how they are used in our designs to significantly prune the search space. Both GPU and FPGA implementations rely on the concurrent exploration of multiple disjoint areas of the search space. In the GPU implementation, hundreds of threads are deployed to complete the search using the many GPU cores. For the FPGA, a multidimensional periodic Costas arrays enumeration core was designed taking into consideration pruning techniques while maintaining a low use of logic resources. Multiple cores were instantiated

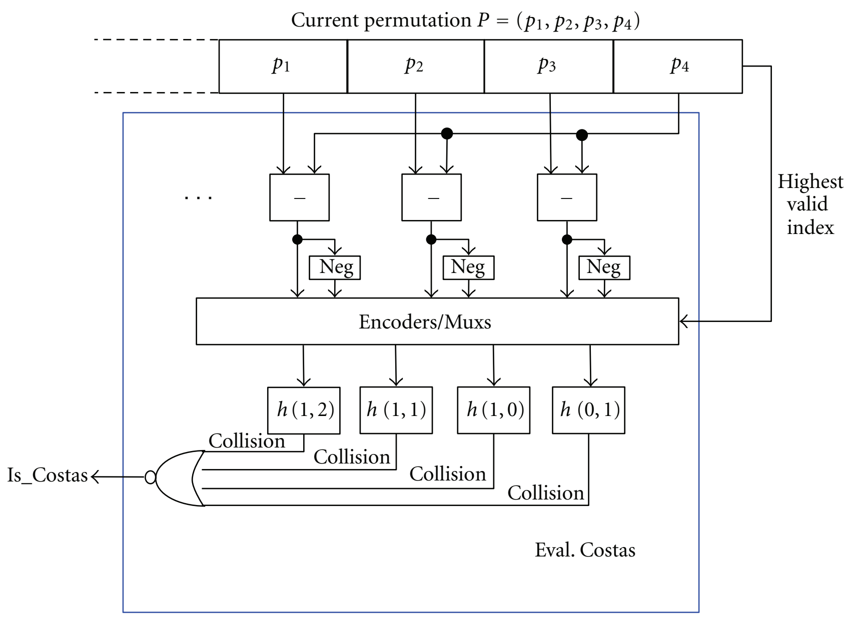


FIGURE 7: Detail of the Costas evaluation block.

TABLE 1: Experimental results for MPCA enumerations.

N		GPU		FPGA		Speedup
	MPCAs found[†]	Time (secs)	Growth rate[⋆]	Time (secs)	Growth rate	FPGA versus GPU
$(\mathbb{Z}_3)^2$	12	3.626	—	$1.24e-4$	—	29241
$(\mathbb{Z}_5)^2$	80	20389	1.72	662	2.63	30.76
$(\mathbb{Z}_7)^2$	336[‡]	$1.87e14$	3.15	$7.42e14$	3.18	25.21

[†] MPCAs starting with $(*; 0)$. As explained in Section 5, the rest of the MPCAs can be obtained using symmetries.
[⋆] Growth rate is computed as:
$(p^2_{(n+1)} - p^2_{(n)})\sqrt{^T(\mathbb{Z}_{p_{(n+1)}})^2/^T(\mathbb{Z}_{p_{(n)}})^2}$.
[‡] Lower bound of MPCAs based on Welch Costas construction and symmetries presented in [13].

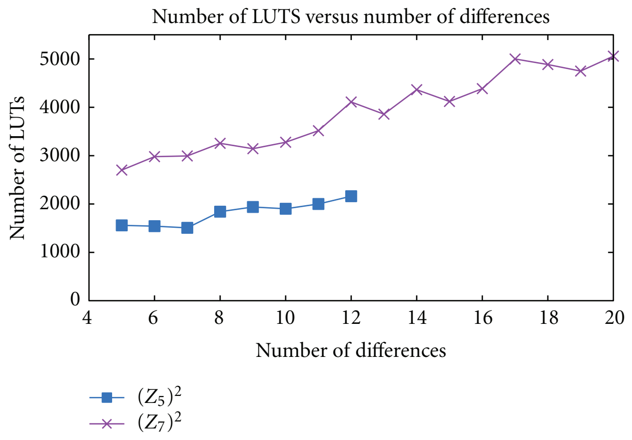


FIGURE 8: LUT utilization by the Costas enumeration module for various differences in the evaluation block. The target device is a XC5VLX330-2, which contains 207360 LUTs.

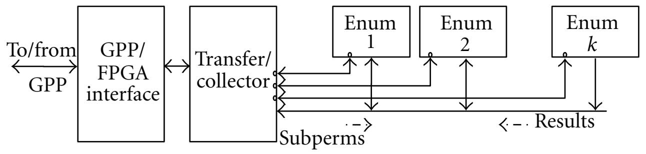


FIGURE 9: FPGA design. The transfer/collector (T/C) block receives a set of subpermutations from the GPP and schedules them among the various MPCA enumerators. Whenever an enumerator finds an MPCA it is sent back to the T/C and back to the GPP for validation.

in the FPGA to provide further acceleration. The fine grained parallelism utilized to evaluate and progress the exploration, coupled with the additional concurrency provided by the multiple cores allowed the FPGA implementation to achieve speedups of up to 30× over the GPU. The implementations completed the first reported enumeration for MPCAs. Furthermore, the MPCA properties and symmetries discovered throughout the process help improve our understanding of these new structures.

Acknowledgments

Dr. R. A. Arce-Nazario was partially supported by NSF Grant number CNS-0923152. Dr. J. R. Ortiz-Ubarri was partially supported by the UPR-RP FIPI funds. Thanks are due to Glen Edwards of Convey Computer for his engineering support in the HC1 implementations and Jonathan Vélez for his help in the GPU implementations.

References

[1] O. Moreno and J. Ortiz-Ubarri, "Group permutable constant weight codes," in *Proceedings of the IEEE Information Theory Workshop*, Dublin, Ireland, September 2010.

[2] O. Moreno and A. Tirkel, "Multi-dimensional arrays for watermarking," in *Proceedings of the IEEE International Symposium on Information Theory*, Saint-Petersburg, Russia, August 2011.

[3] K. Drakakis, "Higher dimensional generalizations of the Costas property," in *Proceedings of the 42nd Annual Conference on Information Sciences and Systems (CISS '08)*, pp. 1240–1245, Princeton, NJ, USA, March 2008.

[4] J. Ortiz-Ubarri and O. Moreno, "Three-dimensional periodic optical orthogonal code for OCDMA systems," in *Proceedings of the IEEE Information Theory Workshop*, pp. 170–174, October 2011.

[5] J. Costas, "Medium constraints on sonar design and performance," *FASCON Convention Record*, pp. 68A–68L, 1975.

[6] K. Drakakis, F. Iorio, and S. Rickard, "The enumeration of Costas arrays of order 28 and its consequences," *Advances in Mathematics of Communications*, vol. 5, no. 1, pp. 69–86, 2011.

[7] J. Ortiz-Ubarri, O. Moreno, A. Z. Tirkel, R. A. ArceNazario, and S. W. Golomb, "Algebraic symmetries of generic (m+1) dimensional periodic costas arrays," *IEEE Transactions on Information Theory*. In press.

[8] K. Drakakis, F. Iorio, S. Rickard, and J. Walsh, "Results of the enumeration of Costas arrays of order 29," *Advances in Mathematics of Communications*, vol. 5, no. 3, pp. 547–553, 2011.

[9] K. Drakakis, F. Iorio, and S. Rickard, "The enumeration of Costas arrays of order 28 and its consequences," *Advances in Mathematics of Communications*, vol. 5, no. 1, pp. 69–86, 2011.

[10] J. Devlin and S. Rickard, "Accelerated Costas array enumeration using FPGAs," in *Proceedings of the 42nd Annual Conference on Information Sciences and Systems (CISS '08)*, pp. 1252–1257, Princeton, NJ, USA, March 2008.

[11] R. Arce-Nazario and J. Ortiz-Ubarri, "Enumeration of Costas arrays using GPUs and FPGAs," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, pp. 462–467, 2011.

[12] K. Drakakis, "On the generalization of the Costas property in higher dimensions," *Advances in Mathematics of Communications*, vol. 4, no. 1, pp. 1–22, 2010.

[13] O. Moreno, A. Tirkel, S. Golomb, and K. Drakakis, "Multi-dimensional periodic Costas arrays over elementary Abelian groups," Preprint.

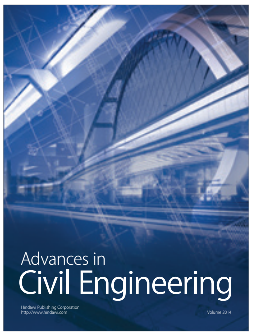
Journal of Engineering

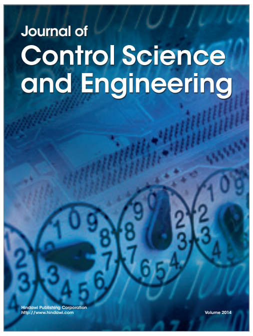
The Scientific World Journal

International Journal of Rotating Machinery

Journal of Sensors

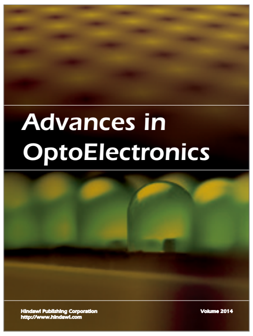
International Journal of Distributed Sensor Networks


Advances in Civil Engineering


Journal of Control Science and Engineering

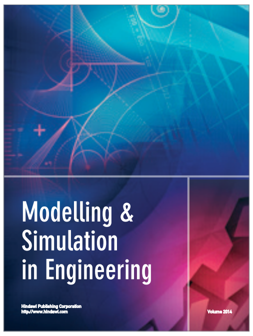
Journal of Robotics

Journal of Electrical and Computer Engineering

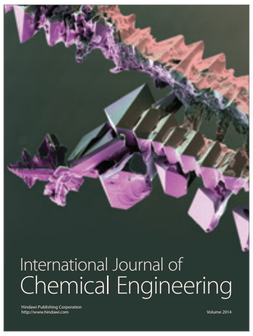

Advances in OptoElectronics

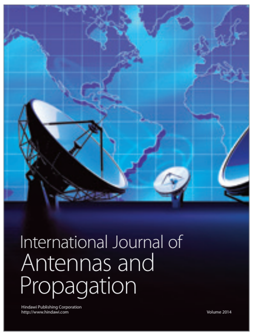
VLSI Design

International Journal of Navigation and Observation

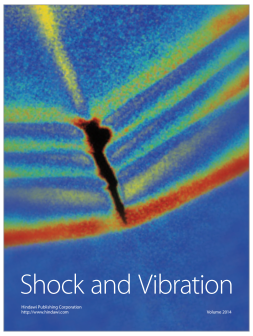

Modelling & Simulation in Engineering

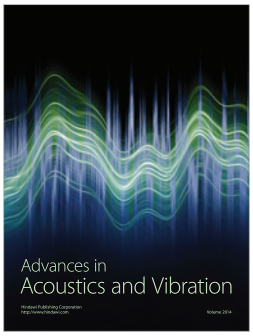
International Journal of Aerospace Engineering


International Journal of Chemical Engineering


International Journal of Antennas and Propagation

Active and Passive Electronic Components


Shock and Vibration


Advances in Acoustics and Vibration