Memory Access Pattern Enumeration in GTM Mapping on Reconfigurable Computers

Xuejun Liang and Jack Jean
Department of Computer Science and Engineering
Wright State University
(xliang, jjean)@cs.wright.edu

Abstract: Generalized template matching (GTM) operations can be accelerated using reconfigurable systems with field programmable gate array (FPGA) hardware resources. The GTM operation involves intensive memory accesses when image is stored off-chip. FPGA designs that use different memory access patterns (MAPs) tend to have big differences in performance and hardware resource requirements. To obtain an optimal GTM FPGA design, there is a need to enumerate design options based on the MAPs. Some measures to evaluate MAPs in terms of the performance and the required hardware resources are introduced. Algorithms to prune MAPs and to enumerate all non-dominated MAPs are proposed so that the design space can be explored more efficiently.

Keywords: Generalized Template Matching, Configurable Computing, Field Programmable Gate Array (FPGA), Design Automation and Optimization

1 Introduction

The generalized template matching (GTM) [1] operations include image-processing algorithms for 2D digit filtering, morphologic operations, motion estimation, and template matching. Mapping GTM operations on reconfigurable computers automatically and optimally is desirable.

The computation of a GTM operation is to move a template (or mask) pixel by pixel in scanning line order, similar to the “Sliding Window-Based Operations” (SWO) as in [4]. In GTM a template may be quite “sparse” and only a low percentage of pixels in a “window” is involved in the computation. The GTM computation can be formulated as a nested loop. The computation iterates through image regions, templates, and pixel locations. The loop body computation, called the Basic Function (BF), involves applying one template window at one image pixel location. The image region is a set of consecutive image rows. The image pixels in the template window that are used for the BF evaluation are called active points. A functional unit design, called the unit function (UF), for the GTM operation consists of one or more copies of BFs as shown in Figure 1. The BFs compute in parallel either at several consecutive pixel locations or for several templates. It is also possible to have multiple UFs inside a single FPGA chip.

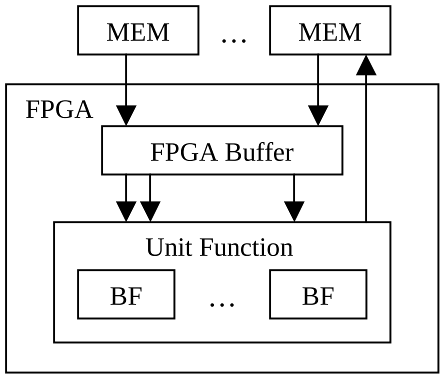


Figure 1: Buffer and Unit Function

In Figure 1, using buffers to buffer image pixels inside the FPGA chip serves two purposes: (1) it can reduce redundant memory accesses as the template window moves, and (2) it can provide more data in parallel to a UF than limited memory ports can. However, the FGPA buffer requires more FPGA area. This tradeoff can be explored by examining different memory access patterns.

There have been many research works about the use of data buffers to speed up applications. For example, a reconfigurable pipeline 2-D convolver design in [5] includes storage (shift registers) for pixel values contained in delay line and for convolution window. Because of these shift registers, the convolution can be carried out for one pixel location each clock period. For another example, the researchers [6] at BYU in their implementation of an automated target recognition algorithm uses data buffers so that the correlations can be computed column by column, and sums up the partial sums for all columns of template. In this method, all column correlations are computed in parallel but only one column of data needs to be available for processing. Note that the former approach buffers more data, and therefore has higher throughput, than the latter approach. But, the former approach needs more circuit area. The tradeoff of speed and area is not addressed in their works.

Since for many applications (including GTM operations) the throughput of the reconfigurable coprocessor is determined by the external memory accesses, the external memory access optimization is very important. The paper in [7] presents a definition of Equivalence of Vector Access in order to infer the on-chip storage for the data that need to be accessed redundantly from external memory. The paper in [8] has a similar work on the automatic derivation of data FIFO. Although two metrics are created in [8] to guide the selection of which direction vector to exploit reuse and determine the number and length of data FIFO, the on-chip storage inferred for a particular application may require a very large chip area, thus it is possible that the chip may not have an area large enough to accommodate such on-chip storage. Also, both works do not consider the computation cost.

Two basic steps of the optimal GTM design methodology are to first enumerate, evaluate, and list enough number of design options of UFs and FPGA buffers, and then to select and combine some of them to build the final design [2]. The design goal is to obtain a GTM design on reconfigurable computer as fast as possible in term of throughput under the constraints of FPGA broad. This paper focuses on the enumeration process of FPGA buffers. A way to list these design options is to order them from the fastest one to the slowest one in terms of throughput. A faster UF has a higher throughput, or a smaller global data introduction interval (II) in its pipelined design. A procedure therefore has been developed to enumerate and evaluate design options based on increasing II divided by packing factor (PF). (PF is defined as the number of image pixels in one memory location. When packing, PF copies of BFs can be used to compute at PF consecutive pixel locations.) During the enumeration process, for a given pair of PF and II, there are usually many designs that can be grouped based on their memory access patterns (MAPs). Therefore, there is a need to enumerate all MAPs for all PFs and IIs.

This paper shows the design enumeration process by relating FPGA buffers and UF synthesis constraints, such as PF and II, to the MAP. A domination relation is defined among MAPs based on measures such as FPGA buffer size, number of memory ports, and memory size. If a MAP is dominated by other MAPs, it is not a candidate for further examination. Algorithms to prune MAPs and to enumerate all non-dominated MAPs are developed so that the design space can be explored systematically and efficiently.

Section 2 describes MAPs and FPGA buffer designs. Section 3 presents measures of MAPs and an algorithm to enumerate all the non-dominated MAPs. Section 4 presents two pruning processes to reduce the number of MAPs that need to be subjected to the domination consideration. Section 5 gives experimental results that demonstrate the effectiveness of the proposed techniques, and a simple example that shows the whole optimization process of GTM design. Section 6 concludes the paper.

2 MAPs and FPGA Buffers

As shown in Figure 1, there are two basic components in a GTM design, an FPGA buffer and a UF. An FPGA buffer can be measured by its area and the initial buffer filling time. A UF can be measured by its FPGA area and the computation time in terms of scheduling length (SL), II and PF. The measures of both components can be affected by different MAPs. In addition, different MAPs may lead to significantly different memory sizes.

2.1 MAPs

For the readability, notations used in the paper are listed in the following table. Some of them are explained latter.

α_W	Frequency of a Write Operation
B_{DATA}	No. of Bits on Image Pixel
C_{IMG}	No. of Image Columns
C_{WIN}	No. of Columns of Template Window
II_{MIN}	Minimum II
II_{UB}	Upper Bound of II
N_{AP}	No. of Active Points of BF
N_{LB}	No. of Line Buffers
$N_{LB}(P)$	Number of Line Buffers for Port P
N_{MP}	No. of Memory Ports on Chip
N_{MW}	No. of Memory Writes of BF
N_{NMAP}	Number of Non-Dominated MAPs
N_P	No. of Ports used ($N_P \le N_{MP}$)
$N_R(P)$	No. of Reads from Port P.
N_{PMAP}	No. of Pruned MAPs
N_R	No. of Reads in II Cycles
N_W	No. of Writes in II Cycles
$N_W(P)$	No. of Writes to Port P.
R_{IMG}	No. of Image Rows

R_{WIN}	No. of Rows of Template Window
S_M	Size of Required Memory
$S_M(P)$	Size of Required Memory by Port P
$S_{MAP}(PF,II)$	Set of MAPs for Given PF and II
$S_{MAP}(PF)$	Set of MAPs for Given PF
$S_{NMAP}(PF)$	Set of Non-Dominated MAPs
W_{MP}	Width of Memory Port in Bits

For a given pair of PF and II, a MAP includes the following information.

- The number of memory ports used (N_P), which should be less than or equal to N_{MP}.
- The number of memory reads ($N_R(p)$) from each memory port, p, ($1 \le p \le N_P$) in the II clock cycles. The vector ($N_R(1)$, $N_R(2)$,.., $N_R(N_P)$) is called the memory reading pattern.
- The number of memory writes ($N_W(p)$) to each memory port, p, ($1 \le p \le N_P$) in II clock cycles. The vector ($N_W(1)$, $N_W(2)$,.., $N_W(N_P)$) is called the memory writing pattern.

The set of MAPs for given PF and II is denoted as $S_{MAP}(PF,II)$. Intuitively, a MAP in $S_{MAP}(PF,II)$ can be represented as a rectangle with $N_P \times II$ cells. Each cell is labeled with R for reading, W for writing, or I for idling. A row of cells stands for memory accesses of one memory port, and a column of cells stands for the memory accesses in one particular clock cycle. For example, when PF=2, II=4, N_{MW}=1, and N_{MP}=4, there may exist a MAP as shown in the following table, in which two memory ports are used (N_P=2), the memory reading pattern is (3,2) and the memory writing pattern is (1,1), where the first row is for memory port 1 and the second for memory port 2.

$$\begin{array}{c|c|c|c|c|} \cline{2-5} \big\updownarrow & R & R & R & W \\ \cline{2-5} N_P & R & R & W & I \\ \cline{2-5} \multicolumn{1}{c}{} & \multicolumn{4}{c}{\longleftarrow\; II \;\longrightarrow} \end{array}$$

Assume that N_R and N_W are the total number of memory reads and writes in II clock cycles of a MAP, respectively. Then the MAP has to satisfy the following constraints.

$$N_R = \sum_{p=1}^{N_P} N_R(p) \text{ and } N_W = \sum_{p=1}^{N_P} N_W(p)$$

$$N_R(p) \ge 0 \text{ and } N_W(p) \ge 0, \; p = 1,2,...N_P$$

$$0 < N_R(p) + N_W(p) \le \text{II}, \; p = 1,2,...N_P$$

$$1 \le N_P \le N_{MP} \text{ and } N_W = N_{MW} \times PF$$

$$1 \le N_R \le R_{WIN} \text{ or } N_R = N_{AP}$$

The requirement of $N_R(p)+N_W(p)>0$ means that the memory port considered in a MAP has at lease one read or one write operation. Note that the total number of writings, N_W, is fixed for given PF, while the total number of readings, N_R, can be changed in a specific range. Also note that different N_R and reading patterns require different FPGA buffers, which is explained in the FPGA buffer section.

PFs can be calculated by

$$PF = 2^b, \; 0 \le b \le \log_2 (W_{MP} / B_{DATA})$$

All possible II also can be determined by

$$II_{MIN} \le II \le II_{UB}$$

$$II_{MIN} = \lceil (1 + N_{MW} \times PF) / N_{MP} \rceil$$

$$II_{UB} = N_{AP} + N_{MW} \times PF$$

The minimum II_{MIN} corresponds to using all memory ports and reading only once in a BF evaluation. The upper bound II_{UP} corresponds to using only one memory port and reading all active points in a BF evaluation.

2.2 FPGA Buffers

When $N_R < N_{AP}$, the computation unit has to get some of the image data from inside the FPGA chip. Therefore there is a need for FPGA buffers. In order to support the function-level parallelism, the data within the template window are buffered inside FPGA as window moves. When N_R is less than R_{WIN}, (R_{WIN} - N_R) rows (lines) of image data must be also buffered inside FPGA. The buffer for an image line is called an image line buffer.

An FPGA buffer example is shown in Figure 2. It uses two memory ports. Port one supports two rows of image data and provides one image pixel during II clock cycles. Port 2 supports three rows and provides two pixels during II clock cycles. Both memory ports need to buffer one image line aside from data inside the template.

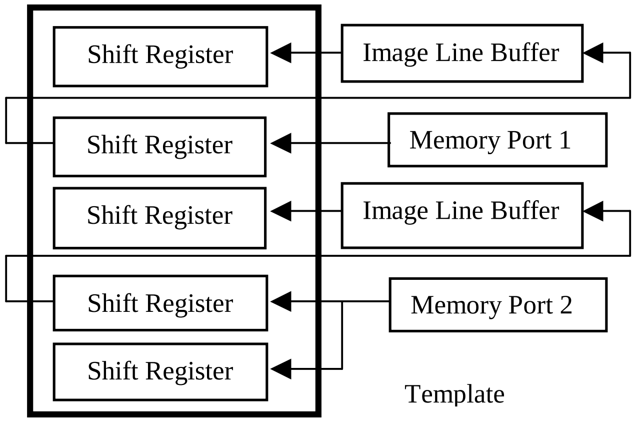


Figure 2: An FPGA Buffer Example

The buffer design is to provide such FPGA buffers when $1 \le N_R \le R_{WIN}$. When N_R=R_{WIN}, there is no need for image line buffers. When N_R=N_{AP}, there is no need for any buffering unless the memory packing scheme is considered. In that case, an FPGA buffer is still needed to support multiple BF copies for "random" accesses of active points [2]. No buffering strategy is considered for $R_{WIN} < N_R < N_{AP}$ because of the complicated memory controller design and relatively few performance benefits.

One of the FPGA buffer design issues is to determine the number of image rows each memory port is supposed to provide. The basic idea is to distribute (R_{WIN}-N_R) image rows almost equally among memory ports to balance the workload of the initial buffer filling.

3 Non-Dominated MAPs

The MAPs can be evaluated with the following four quality measures. The first measure is the number of image line buffers (N_{LB}) defined as R_{WIN}-N_R. The second is the number of memory port (N_P). The third is the initiation interval (II). The last measure is the memory size requirement (S_M).

$$S_M = \max\{S_M(p) : 1 \le p \le N_P\}$$

where $S_M(p)$ is the required memory size for memory port p ($1 \le p \le N_P$) as defined as

$$S_M(p) = \begin{cases} 1 + N_W(p) \times \alpha_W & \text{when } N_R(p) \neq 0 \\ N_W(p) \times \alpha_W & \text{when } N_R(p) = 0 \end{cases}$$

Let $S_{MAP}(PF) = \{S_{MAP}(PF, II) : II = 1,2,3,...\}$

Definition: For two MAPs A and B in S_{MAP}(PF), A dominates B if A's $N_L \le$ B's N_L, and A's $N_P \le$ B's N_P, and A's $S_M \le$ B's S_M, and A's II $\le$ B's II.

From this definition, when A dominates B, A is better than B in all four measures. Therefore, there is no need to consider B for an optimal design. In other words, B can be eliminated during the enumeration of UF design options.

If a MAP in S_{MAP}(PF) is not dominated by any other element in S_{MAP}(PF), it is considered non-dominated. The set of all the non-dominated elements in S_{MAP} (PF) is denoted as S_{NMAP}(PF). The set of all the non-dominated elements in S_{MAP}(PF, II) is denoted as S_{NMAP}(pf, II). The following algorithm is a brute-force procedure of finding S_{NMAP}(PF) from a given S_{MAP}(PF).

Algorithm: Finding S_{NMAP}(PF) from S_{MAP}(PF).

```
$S_{NMAP}(PF) = \Phi$;
While $S_{MAP}(PF) \neq \Phi$ {
  pick $a \in S_{MAP}(PF)$; Temp_Set = $S_{MAP}(PF)$ - {a};
  nd = true;
  for each $e \in$ Temp_Set and nd == true {
    if a dominates e then $S_{MAP}(PF) = S_{MAP}(PF)$ - {e};
    else if e dominates a then
      {$S_{MAP}(PF) = S_{MAP}(PF)$ - {a}; nd = false;}}
  if nd == true then
    $S_{NMAP}(PF) = S_{NMAP}(PF) \cup$ {a}; }
```

Note that S_{MAP}(PF) needs to be found before the algorithm can be applied which in turn requires the enumeration of S_{MAP}(PF,II) for each II. The set S_{MAP}(PF,II) can be enumerated by iterating through all N_R and N_P, and in each iteration, finding reading patterns that satisfy the (3.1) constraints and finding writing patterns that satisfy the (3.2) constraints. That is,

for ($N_R = 1$ to R_{WIN}) or ($N_R = N_{AP}$)
for ($N_P = 1$ to N_{MP})

$$(3.1)\ \begin{cases} \sum_{p=1}^{N_P} N_R(p) = N_R \\ II \ge N_R(1) \ge N_R(2) \ge \cdots \ge N_R(N_P) \ge 0 \end{cases}$$

$$(3.2)\ \begin{cases} \sum_{p=1}^{p=N_P} N_W(p) = N_M \\ 0 \le N_W(p) \text{ and } 1 \le N_R(p) + N_W(p) \le II,\ p = 1,2,....N_P \\ N_R(p) = N_R(p+1) \Rightarrow N_W(p) \ge N_W(p+1), \\ p = 1,2,..., N_P - 1 \end{cases}$$

In the above formulation, the memory port ordering is ignored because it does not influence design optimality. This assumption cuts down the number of reading patterns and leads to

$$II \ge N_R(1) \ge N_R(2) \ge \cdots \ge N_R(N_P) \ge 0$$

The following constraint is assumed for a similar reason. For $1 \le p \le N_P$-1,

$$N_R(p) = N_R(p+1) \Rightarrow N_W(p) \ge N_W(p+1)$$

Note that this enumeration process grows very fast. The equation in (3.1) corresponds to the integer partition problem. There is no general formula to express the number of such partitions. But, for a fixed N_R, the growth of the number is $\theta(N_R^{(N_P-1)})$ [3]. The equation in (3.2) can lead to even more partitions because the order of writing cannot be ignored after the order of reading is fixed. Therefore, although the solutions to the constraints (3.1) and (3.2) can be enumerated directly, it is a good idea to combine the S_{MAP}(PF) enumeration process with the following pruning process so that the enumeration process can avoid most of the dominated MAPs.

4 MAP Pruning

The MAP pruning process includes the reading pattern pruning and writing pattern pruning.

4.1 Reading Pattern Pruning

Some reading patterns can be pruned because they cannot lead to a non-dominated MAP.

Definition: A minmax decomposition of an integer n with respect to another integer q is (i_1, i_2, .., i_q) that minimizes i_1 subject to

$$\sum_{j=1}^{q} i_j = n$$
$$i_1 \ge i_2 \ge ... \ge i_q \ge 1$$

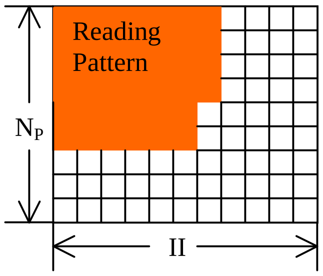
For an integer n, there are a total of n minmax decompositions with respect to q=1, 2, .., n, respectively. For example, assume n=4, the four minmax decompositions are (1,1,1,1), (2,1,1), (2,2), and (4). As another example, if n=17 and q=3, then the minmax decomposition is (6,6,5).

Given a MAP A, its reading pattern is denoted as $P_A(R)=(N_R(1), N_R(2), .., N_R(N_P))$ that satisfies (3.1). Assume that $N_R(q) \neq 0$ and $N_R(q+1)=0$ and that $(N'_R(1), N'_R(2), .., N'_R(q))$ is the minmax decomposition of N_R with respect to q. Then the MAP A can be transformed to another MAP T(A) such that the reading pattern $P_{T(A)}(R) = (N'_R(1), N'_R(2), .., N'_R(q), 0, 0, .., 0)$. Note that the writing pattern of T(A) may be different from that of A.

Theorem 1: T(A) dominates A.

The proof is omitted because of the space.

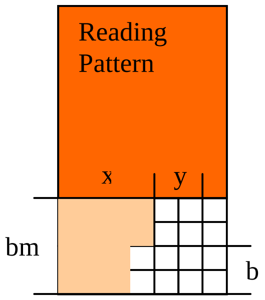
This theorem allows the pruning of huge number of reading patterns because, given N_R, only the minmax decomposition needs to be considered. Given N_P, N_R, and II, a typical MAP with a reading pattern that contains a minmax decomposition of N_R with respect to q ($1 \le q \le N_P$) is as follows.



4.2 Writing Pattern Pruning

After the reading pattern pruning, the next step is to prune writing patterns. For this purpose, reading patterns can be classified into seven types. The idea is to minimize the memory size (S_M) of a MAP by arranging writing patterns for each type of reading patterns separately, and then reducing II or N_P without changing S_M. The pruning algorithm is presented for only two types of reading patterns because of the space.

Case1: The reading pattern is shown as follows, where $N_R(i)$=II for $1 \le i \le q$ and $N_R(j)=0$ for $q<j \le N_P$ ($1<q<N_P$). Assume the writing pattern is as shown at the bottom of the figure. The pattern can be pruned if one of the following two conditions holds.

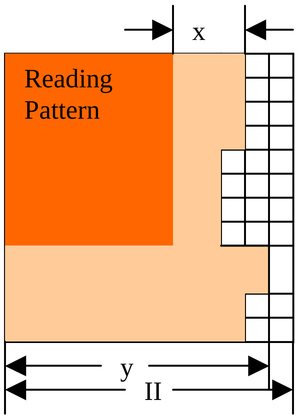


Otherwise, the MAP gets the minimum S_M for the given reading pattern.

(1) $x < b$

(2) There is a *y*>0 such that $x+y \le II$, $(x+y) \times \alpha \le 1$, and $x-b \le y*(bm-1)$.

Case 2: The reading pattern is shown as follows, where $N_R(i)$=r for $1 \le i \le q$ and $N_R(j)=0$ for $q<j \le N_P$ (1<r<II and $1<q<N_P$). The writing pattern can be viewed as two parts, one at the right of the reading pattern and the other below that. In order to get the minimum S_M, memory ports in each part has to have "balanced" numbers of memory writes (as shown in the figure). The ranges of x and y are $0 \le x \le II-N_R(1)$ and $1 \le y \le$ II. From those values of x and y, the (x, y) pair that leads to the minimum S_M is to be found. If the maximum value of x and y in that pair is less than II, then the pattern can be pruned.



Theorem 2: No MAP in the pruned S_{MAP}(PF, II) (after applying writing pattern pruning) can be dominated by an element in S_{MAP}(PF, II') with II' less than II.

The basic idea of pruning S_{MAP}(PF) is to prune each S_{MAP}(PF, II) in the increasing order of II. Theorem 2 ensures that each element in the pruned S_{MAP}(PF, II) cannot be dominated by an element in S_{MAP}(PF, II') with II' < II. Therefore, the S_{NMAP}(PF) can be obtained by computing each S_{NMAP}(PF, II) separately.

By using the above two pruning processes, one for reading patterns and the other for writing ones, enumerating the pruned S_{MAP}(PF, II) can be finished in polynomial time. For each N_R and N_P, the reading patterns can be enumerated in linear time to N_R, and for each reading pattern, the writing patterns can be determined in a polynomial time with respect to N_P, N_W and II. Therefore, the total computational time complexity is polynomial to the problem size.

5 Experimental Results

5.1 Efficiency of Pruning Algorithm

The proposed pruning and enumeration algorithm and the algorithm to find S_{NMAP}(PF) from S_{MAP}(PF) have been implemented in a C++ program. Also, a dynamic programming algorithm that solves the integer partition problem (Constraints (3.1) and

(3.2)) has been developed and implemented to compute S_{MAP}(PF). Two experiments were performed.

In the first experiment, assume that N_{MP}=4, W_{PORT} =32, B_{DATA}=8, N_{MW}=2, and α_W=1.0. For several different R_{WIN} and N_{AP}, the number of MAPs after pruning (N_{PMAP}) and the number of non-dominated MAPs (N_{NMAP}) are counted. The results are summarized in the following table, where the computation time is in seconds.

R_{WIN}	N_{AP}	N_{PMAP}	N_{NMAP}	Time	$\frac{N_{NMAP}}{N_{PMAP}}$
3	9	96	50	0.06	52%
9	20	288	130	0.17	45%
25	45	816	351	0.66	38%
45	65	1492	632	2.14	42%
65	75	2169	915	4.89	42%
85	95	2845	1199	10.22	42%
105	115	3522	1481	19.11	42%
125	135	4199	1765	33.28	42%

The table shows that the algorithm is very effective and the computation time is within tens of seconds for typical sizes of GTM operations. The last column of the table shows that after pruning, about 40%-50% of MAPs are non-dominated. Therefore, the algorithm to find S_{NMAP}(PF) removes around 50%-60% of MAPs after pruning.

The second experiment is to compare the algorithms with or without pruning. In this experiment, assume N_{MP}=4, W_{PORT} =32, B_{DATA}=8, N_{MW}=1, and α_W=1.0. In the following two tables, the first one lists the results with pruning and the second without. It can be observed that the number of MAPs (N_{MAP}) without pruning is much larger than the corresponding N_{PMAP}, and the ratio of N_{MAP} to N_{PMAP} increases significantly as the problem size increases. That would explain the same trend for the computation times.

W_{WIN}	N_{AP}	N_{PMAP}	Time
3	9	72	0.02
9	20	214	0.06
15	30	346	0.17

(a) With Pruning

W_{WIN}	N_{AP}	N_{MAP}	Time
3	9	4032	1.48
9	20	61133	30.59
15	30	285966	252.38

(b) Without Pruning

It is worth mentioning that the values of N_{MAP} shown in the second table are lower bounds of MAPs. When solving the constraints (3.1) and (3.2) for N_R's and N_P's, the following extra constraint is incorporated to reduce the size of the solution space (S_{MAP}(PF)).

$$N_R + N_W \geq II + N_P - 1$$

5.2 GTM Optimal Mapping

This section presents a simple example to show the steps to explore GTM design space and find the fastest (highest throughput) design. In this example, it is assumed that the FPGA board has only one FPAG chip (Xilinx 4000 series) with only one memory port, and that C_{IMG}=360, R_{WIN}=3, C_{WIN}=4, N_{AP}=9, N_{MW}=1, α_W=1.0, W_{MP}=16, and B_{DATA}=8. The BF is simply a function summing up nine active points in the template window.

The first step is to get all non-dominated MAPs and list them in the increasing order of II/PF as follows. Note that the smaller II/PF a design has, the higher the throughput is.

PF	II	MAP	Buffer Type
2	3	RWW	Full/Packing
2	4	RRWW	Hybrid/Packing
1	2	RW	Full
2	5	RRRWW	Partial/Packing
1	3	RRW	Hybrid
1	4	RRRW	Partial
2	11	RRRRRRRRRWW	Internal
1	10	RRRRRRRRRW	No

The second step is to compute the buffer and synthesize the BF dataflow graph to get the UF for each MAP in the list. The third step is to select UFs and assign FPGA board resources such as memory ports and FPGA chips them to get the final GTM design. Because there is a single FPGA and a single memory port, the GTM design is composed of only one UF associated with possible buffer. Therefore, in this simple example, the third step is trivial.

After computing the buffer area and the UF area for each MAP in the above order. The first MAP whose corresponding design's area is less than t he FPGA area is the fastest design. Assume the FPGA has 720 CLBs, the following area information implies the fourth map corresponds an optimal design.

Buffer Area	UF Area	Total Area
680	408	1088
474	404	878
548	216	764
264	376	640

The above buffer areas are computed according to an approximate formula. The UF area is obtained from the datapath generated by synthesizing the dataflow graph of the BF. Each node in the datapath corresponds to one component in the library that contains the area. Note the UF controller area is not added at the current stage.

6 Conclusions

This paper presents the concept of memory access patterns (MAPs) and the measures to evaluate the MAPs. The relations among MAPs, FPGA buffers, and unit function designs are studied. Algorithms to prune and enumerate all non-dominated MAPs are given so that the FPGA GTM design space can be explored systematically and efficiently.

This work represents an important step in GTM optimal mapping on reconfigurable computers. At Wright State University, a VHDL generator for FPGA buffers has be developed to generate VHDL file automatically according to several design parameters including MAPs. The unit function synthesis algorithms have also been created and implemented to provide pipelined designs of the unit function (UF). After getting these building blocks of GTM designs, an FPGA board resource binding and image region partitioning can be performed. Finally an interface tool still under development will generate steering logic and interface controllers to get an optimal GTM design.

7 References

[1] Xuejun Liang, Jack Jean and Karen Tomko, "Data Buffering and Allocation in Mapping Generalized Template Matching on Reconfigurable Systems", to appear in the Journal of Supercomputing, Special Issue on Engineering of Reconfigurable Hardware/Software Objects
[2] Xuejun Liang and Jack Jean, "Interface Design for the Matching of Generalized Template Matching on Reconfigurable Systems", in Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 159-165, June 2000
[3] Donald L. Kreher and Douglas R. Stinson, "Combinatorial Algorithms: Generation, Enumeration, and Search", CRC Press, 1998
[4] C. Thibeault and G. Begin, "A Scan -Based Configurable, Programmable, and Scalable Architecture for Sliding Window-Based Operations", in IEEE Transactions on Computers, pp. 615-627, 1999
[5] Bernard Bosi, Guy Bois and Yvon Savaria, "Reconfigurable Pipeline 2-D Convolvers for Fast Digital Signal Processing", in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 299-308, Vol. 7, No. 3, 1999
[6] M. Rencher, and B. L. Hutchings, "Automated Target Recognition on Splash 2," in IEEE Symposium on FPGA Custom Computing Machines, pp. 192-200, April 1997.
[7] Markus Weinhardt and Wayne Luk, "Memory Access Optimization and RAM Inference for Pipeline Vectorization", in Proceedings of FPL' 99, pp.61-70, 1999
[8] Pedro Diniz and Joonseok Park, "Automatic Synthesis of Date Storage and Control Structures for FPGA-based Computing Engines", in IEEE Symposium on Field Programmable Custom Computing Machines, 2000