DATA REUSE EXPLORATION FOR FPGA BASED PLATFORMS APPLIED TO THE FULL SEARCH MOTION ESTIMATION ALGORITHM *

Qiang Liu, Konstantinos Masselos, George A.Constantinides

Department of EEE
Imperial College London
London SW7 2AZ
email: {qiang.liu2, k.masselos, g.constantinides}@imperial.ac.uk

ABSTRACT

Compilation of high level descriptions to field programmable gate array hardware forms a promising option for the efficient mapping of computationally intensive applications under tight development time constraints. In this paper data reuse exploration on top of an existing hardware compilation environment is discussed. The full search motion estimation algorithm for video processing is used as a test vehicle. The systematic approach adopted for the exploration of the data reuse space is described. Experimental results prove that the exploitation of data reuse may lead to more than 80% reduction of the execution time and up to 95% reduction of the off-chip memory accesses.

1. INTRODUCTION

As the number of logic resources included in FPGA devices keeps increasing HDL-based FPGA programming becomes more problematic from a development time point of view. The solution seems to be the moving up of the abstraction level using hardware compilation from less detailed descriptions than conventional HDLs.

A number of commercial tools for compilation of behavioral and cycle accurate description to FPGAs hardware exist [1, 2, 3, 4]. Some high level synthesis approaches existing in research community are described in [5, 6].

Exploitation of data reuse (data temporal locality), by keeping frequently used data in local memories, can lead to execution time and power consumption reduction. A systematic approach for data reuse exploration targeting ASICs is described in [7]. Data reuse is not exploited in most existing ASIC/FPGA hardware compilation environments. Data reuse exploitation targeting FPGAs limited to one level stored in registers is described in [8]. The approach described in [9] infers FPGA on-chip RAMs and shift registers to exploit data reuse.

*THIS WORK WAS SUPPORTED BY THE EPSRC (EP/C549481/1)

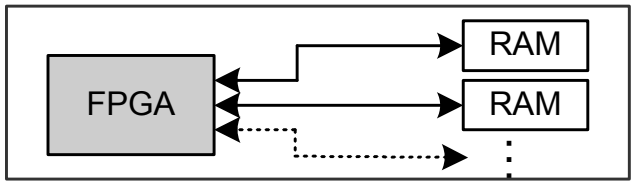

Fig. 1. Target platform.

In this paper data reuse exploration targeting implementation platforms based on FPGA and external memory (Fig. 1) is discussed for the full search motion estimation algorithm. A systematic approach has been adopted for the identification of the data reuse options and the selection of the most promising ones. In contrast to existing approaches for data reuse exploration, the following issues are considered:

- Combination of data reuse exploitation with coarse grained data-level parallelization.
- Exploitation of data reuse assuming storage of all the data of the targeted algorithm on chip.
- Explicit evaluation of the area overheads introduced by the data reuse exploitation and the impact they have on the device size and cost.
- Physical memory hierarchy limitations.

The rest of the paper is organized as follows: Section 2 briefly describes the full search motion estimation kernel. Section 3 describes the data reuse exploration approach adopted. Data reuse exploration results for the full search motion estimation algorithm are presented in Sections 4 and 5. Finally, conclusions are presented in Section 6.

2. FULL SEARCH MOTION ESTIMATION

The classical full search motion estimation kernel [10] has been used as test vehicle for the demonstration of the proposed methodology for data reuse exploration. The full search

```
for(x=0; x<N/B; x++)          /*for each current block */
 for(y=0; y<M/B; y++)
  for(i=-P; i<P+1; i++)         /* for each candidate block in the RW*/
   for(j=-P; j<P+1; j++)
    for(k=0; k< B; k++)         /* for each pixel of the blocks */
     for(l=0; l< B; l++)

   {
    if((B*x+i+k<0) || (B*x+i+k>N-1) || (B*y+j+l<0) || (B*y+j+l>M-1))
      ...
      ...= current_frame[B*x+k][B*y+l];
      ...= previous_frame[B*x+i+k][B*y+j+l];
      …
   }
```

Fig. 2. Structure of the full search motion estimation kernel.

motion estimation is widely used and it is a typical kernel for many video processing applications. A video frame under compression (`current_frame`) is divided into non-overlapping blocks of given size. Each block in the current frame is compared to a number of candidate blocks of the same size in the previous (in time) frame of the video sequence. The candidate blocks are included in a reference window around the (top left corner) coordinates of the block under consideration. The comparison is performed using a certain distortion criterion, normally the mean absolute difference. The top left corner coordinates of the candidate block that minimizes the distortion criterion define the motion vector for the block under consideration.

The general structure of the full search motion estimation kernel is described in Fig.2. The control flow of the algorithm includes six regularly nested loops with no data dependencies. The basic parameters of the algorithm are: the frame size $N \times M$ pixels, the number of candidate blocks $(2P+1) \times (2P+1)$ and the block size $B \times B$. The candidate blocks lie inside a reference window of size $(2P+B) \times (2P+B)$. The two major arrays (`current_frame, previous_frame`) corresponding to the current and the previous frames are accessed inside the six nested loops.

3. DATA REUSE EXPLORATION METHODOLOGY

In this section the methodology adopted for the data reuse exploration of the full search motion estimation algorithm is described.

a) Identification of data reuse levels

In the first step the data reuse levels are identified for each array of the given algorithm independently as in [7, 9]. A data reuse level of an array A is associated with an array RL_A of smaller size storing a subset of data of the original array that are potentially reused.

In a general case, considering a k-level nested loop with iterators $I_1, \ldots, I_k$ (where I_1 corresponds to the outermost loop and I_k corresponds to the innermost loop), an array A accessed inside the nested loop according to an address expression given by the function $f(I_1,\ldots,I_k)$ and a function *size()* returning the size of an array, the procedure for defining the data reuse levels for the array A can be described as follows:

For each I_j, $1 \leq j \leq k$, a reuse level array RL_A_j initialized inside the loop with iterator I_j, is introduced. Size(RL_A_j) is determined by the data of A accessed in one iteration of I_j and $size(RL\_A_{j+1}) < size(RL\_A_j) < size(A)$. The address function of RL_A_j is $f_j(I_{j+1}, \ldots, I_k)$, and RL_A_j replaces A inside the loops with iterators I_{j+1}, $\ldots, I_k$.

The procedure is repeated for all arrays in the given algorithm. In the case of the full search motion estimation algorithm the data reuse levels for the two major arrays (`current_frame, previous_frame`) are presented in Tables. 1 and 2.

Table 1. Data reuse levels of `current_frame` array.

Array	Comment	Size	Initializing iterator	Iterators on which accesses to the array depend after initialization
current_frame	Original array	N X M		x,y,k,l
line of current blocks (LCB)	First reuse level	B X M	x	y,k,l
current block (CB)	Second reuse level	B X B	y	k,l
current block line of pixels (CBLP)	Third reuse level	1X B	k	l
pixel of current block	Fourth reuse level - corresponds to copying data from the original array to a register before use	1	l	-

Table 2. Data reuse levels of `previous_frame` array.

Array	Comment	Size	Initializing iterator	Iterators on which accesses to the array depend after initialization
previous_frame	Original array	N X M		x,y,i,j,k,l
line of reference windows (LRW)	First reuse level	(B+2p) X M	x	y,i,j,k,l
reference window (RW)	Second reuse level	(B+2p) X (B+2p)	y	i,j,k,l
line of reference blocks (LRB)	Third reuse level	B X (B+2p)	i	j,k,l
reference block (RB)	Fourth reuse level	B X B	j	k,l
reference block line of pixels (RBLP)	Fifth reuse level	1X B	k	l
pixel of candidate block	Sixth reuse level - corresponds to copying data from the original array to a register before use	1	l	-

b) Identification of data reuse options

In the next step the data reuse options (RO) for each array are defined. The data reuse options for the array A can be defined as combinations of reuse levels (RL_A_j) of the array . The combinations can be represented as sets in the following way:

$$RO_A = \{RL\_A_j : 1 \leq j \leq k\}$$

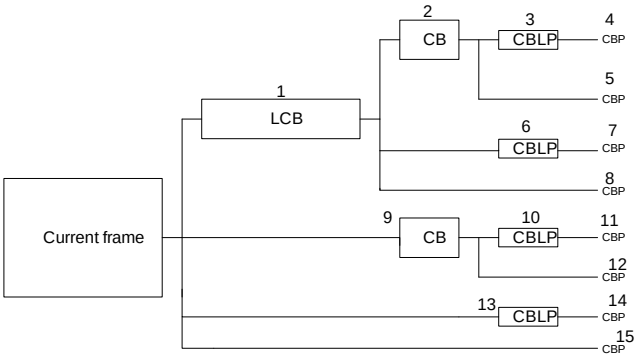

Fig. 3. Data reuse options tree for `current_frame` array.

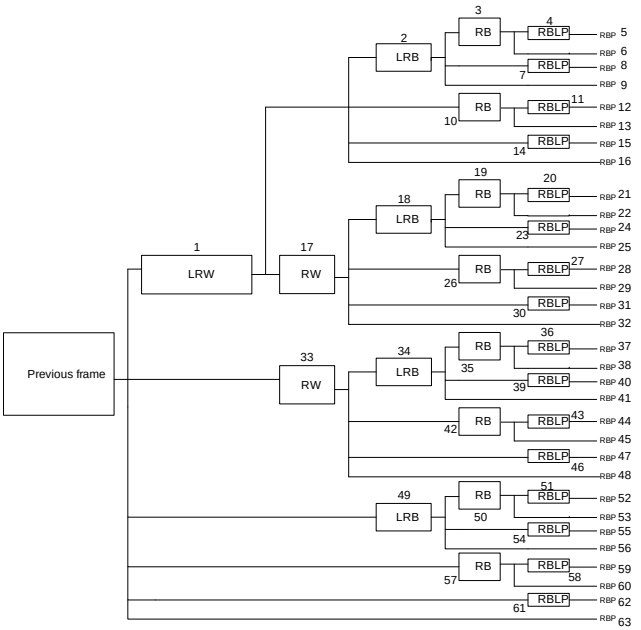

Fig. 4. Data reuse options tree for `previous_frame` array.

$$1 \leq |RO_A| \leq k$$

When data reuse levels are combined, the data of the original array are copied to the first reuse level involved (corresponding to the outermost loop) and the data to all other levels are copied from the previous reuse level involved. All the data reuse options for a given array can be represented using a data reuse tree [7]. The root of the tree is the original array and the leaf nodes correspond to the lower data reuse level (corresponding to the innermost loop). All the nodes in the tree correspond to a data reuse option that includes the given node and all the preceding nodes up to the root level. The data reuse options trees for the two arrays of the full search motion estimation algorithm are presented in Figs. 3 and 4 (the data reuse options are numbered).

c) Identification of the most promising reuse options

Table 3. Reuse metric values for representative reuse options.

Reuse option	Reuse metric	Reuse option	Reuse metric
LCB	81	LRW	27
CB	81	RW	9
(LCB,CB)	(1, 81)	(LRW, RW)	(3, 9)

This step involves the identification of the most promising data reuse options for each array. Only data reuse options in which the reuse level arrays can be stored in different levels of the targeted physical memory hierarchy should be considered. Otherwise, unnecessary copies between locations in the same level of the memory hierarchy will be introduced. Given that FPGA devices include two levels of memory hierarchy (RAMs and registers), the following cases could be considered : a) if the input video frames can fit in on-chip RAMs, only one level of data reuse in registers can be introduced; b) if the input video frames are stored in one level of off-chip memories (as shown in Fig. 1), two levels of data reuse can be introduced (one in on-chip RAMs and another in registers).

The different reuse options can be evaluated using a metric of the reduction of the accesses to the large arrays of the targeted algorithms. For a given reuse level RL_A_j of the array A, a simple definition for this metric is:

$$\frac{Num_{\_\ reads\ from\ RL\_A_j}}{Num_{\_\ writes\ to\ RL\_A_j\ from\ RL\_A_{j-1}}}$$

If $j = 1$ corresponding to the first data reuse level then the previous level is the original array A.

For reuse options with more than one reuse levels, the metrics between all pairs of levels must be considered. The reuse metric values for some representative reuse options of the main arrays of the full search motion estimation algorithm are presented in the Table 3.

d) Final data reuse decisions

To make final data reuse decisions, data reuse options for all the arrays must be selected and combined. The final assignment of the data reuse arrays to the levels of the memory hierarchy is made at this point. Two cases are identified:

1. No area constraints exist. In this case the best data reuse options can be selected for each of the arrays.

2. Area constraints exist (the design must be implemented using a specific FPGA device). In this case the selection of the best data reuse options for each of the arrays may not be feasible due to the limited amount of resources in each level of the memory hierarchy. Priority should be given to certain arrays based on the related data reuse metrics and total number of accesses to them.

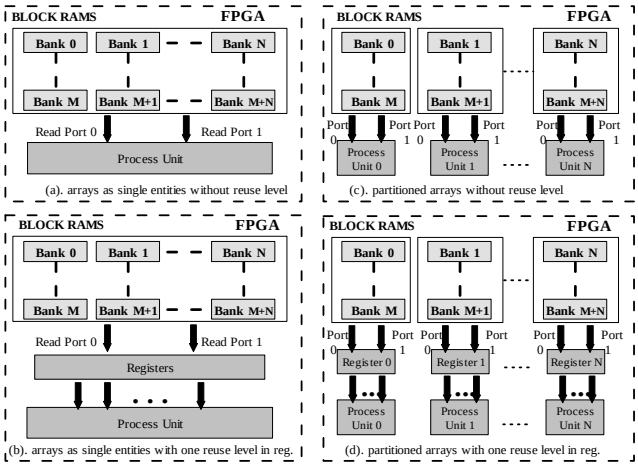

Fig. 5. Implemented data reuse options assuming on-chip storage of the input frames .

In the next sections data reuse exploration results for the full search motion estimation algorithm are discussed. These results correspond only to a part of the data reuse options described in this section. In all cases it is assumed that source data are already loaded into the largest memory level of the targeted memory hierarchy.

4. DATA REUSE EXPLORATION ASSUMING ON-CHIP STORAGE OF INPUT FRAMES

In this case it is assumed that the full search motion estimation algorithm operates on the luminance component of QCIF frames of 144×176 pixels. Given the relatively small size of the `current_frame` and `previous_frame` arrays it is possible to store them on on-chip RAMs of FPGA devices. Designs for the original full search motion estimation algorithm and algorithm's versions realizing some of the data reuse exploration options described in the previous section have been implemented in Handel C [4]. The Handel C descriptions of the designs have been synthesized using Celoxica's DK Design Suite 4.0 and mapped on Xilinx Virtex II devices using Xilinx ISE 6.3 software. The total on-chip power consumption is analyzed using Xpower (integrated in Xilinx ISE). All results are obtained using a Xilinx Virtex II XC2V6000-ff1152-4 device, however, the smallest device, in which each design can fit, is shown in the last column of the tables. Two different cases are considered: a) arrays are treated as single entities and b) arrays are partitioned into sub-arrays.

4.1. Arrays treated as single data entities

In this case the arrays in different design versions are either stored as single entities in RAMs and accessed through a single pair of ports or in registers. The results of this exploration are presented in Table 4.

Table 4. Data reuse exploration results assuming on-chip storage of the input frames without data partitioning: (a) original without data reuse exploitation; (b) one data reuse level for each input frame: (CB, RW) stored in registers .

Version	Block RAMs	Slices	Cycles /Frame	Freq. (MHz)	Time /Frame (s)	Power on-chip (mW)	Smallest device
(a)	32	411	2982742	101	0.030	1154.11	XC2V 500
(b)	32	980	622586	111.6	0.006	1216.59	XC2V 500

The design optimized for data reuse (shown in the second row of the Table 4) introduces the RW (reference window) and the CB (current block) levels of data reuse in registers for the `previous_frame` and the `current_frame` arrays respectively. The RW and the CB are copied into two sets of registers and then $(B \times B)$ pixels are read from each of these sets and processed in parallel (corresponding to unrolling of the two innermost loops). The original and the data reuse optimized designs are described in the left hand side of Fig.5. Data reuse exploitation leads to large execution time (time/frame) improvement, about 80%, resulting from the improvement of the clock frequency (by approximately 10 MHz) and the great reduction of the number of execution cycles. This improvement in performance comes with a small penalty in power consumption, less than 6%, due to extra transfers to copy data from the on-chip Block-RAMs to the registers. From the resource point of view, when data reuse is exploited, the number of slices used is more than doubled but this does not create the need to move to larger and higher cost devices.

4.2. Arrays partitioned into sub-arrays

In this case the arrays are partitioned at the source code level to form a number of sub-arrays that are distributed to different memories (on-chip RAMs with different pairs of ports or sets of registers). In this way coarse grained data-level parallelization can be applied. The same designs as in the previous subsection have been implemented. However, in this case the two input arrays are partitioned into 18 sub-arrays, each stored in a different block of on-chip RAMs with two ports. Introduction of the reuse level in this case requires introduction of one reuse array per partition. The reuse arrays are stored in registers. The results of the exploration are presented in Table 5. The original and the data reuse optimized designs are described in the right hand side of the Fig. 5.

The execution time of the data reuse optimized design is reduced by 80% due to the large decrease of the number of cycles despite the reduction of the clock frequency by

Table 5. Data reuse exploration results assuming on-chip storage of the input frames with data partitioning: (a) original without data reuse exploitation; (b) one reuse level for each input frame: (CB, RW) stored in registers .

Version	Block RAMs	Slices	Cycles /Frame	Freq. (MHz)	Time /Frame (s)	Power on-chip (mW)	Smallest device
(a)	54	7280	165706	101.5	0.002	1966.16	XC2V 2000
(b)	54	19791	34502	92.1	0.0004	2431.40	XC2V 4000

about 9%. The amount of resources required for the data optimized design is doubled compared to the original design, and a larger device is required in this case. Power consumption is increased by about 24% as well. Overheads are larger in this case.

5. DATA REUSE EXPLORATION ASSUMING OFF-CHIP STORAGE OF THE INPUT FRAMES

In this case it is assumed that the full search motion estimation algorithm operates on CIF frames of 288×352 pixels. These frames are assumed to be stored in off-chip SRAMs. In our experiments, the Celoxica RC300 platform with a Xilinx Virtex II XC2V6000-ff1152-4 and 4 banks of SRAMs is used as the implementation platform. The same design flow as in the case of the on-chip frame storage is followed.

5.1. Arrays treated as single data entities

In this case all the data reuse arrays are treated as single entities and stored in on-chip RAMs with a single pair of ports or in registers. In the case of registers storage all elements of the arrays can be accessed at the same time corresponding to unrolling of the innermost two loops. The results of this exploration are presented in Table 6. Four different designs exploiting data reuse have been developed (rows 2 - 5 of Table 6). All these designs introduce one or two levels of data reuse. The implemented data reuse options in this case are presented in the left hand side of Fig. 6.

The designs with one reuse level lead to large reductions of execution time up to 86%, due to large reduction of the number of execution cycles and 41% improvement of the clock frequency. The design with two reuse levels provides 86% reduction of execution time with large reduction of execution cycles and 22% improvement of the clock frequency. When input frames are stored off-chip, another important benefit is the large reduction of the number of accesses to off-chip memories, corresponding to system power consumption reduction. A reduction of up to 95% is achieved in two cases (a one reuse level design and the two reuse levels design). Another important observation is that although

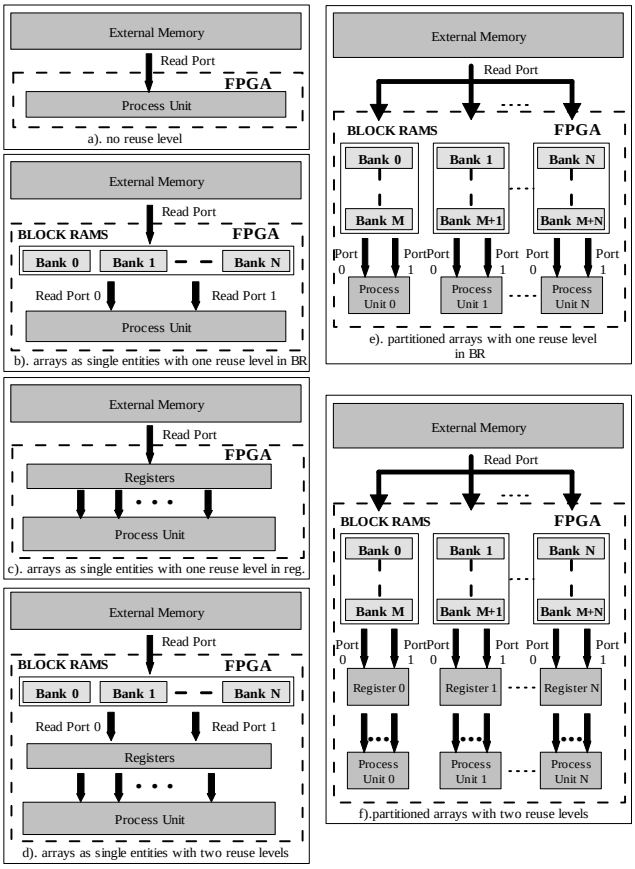

Fig. 6. Implemented data reuse options assuming off-chip storage of the input frames.

overheads in the amount of slices used are introduced by all designs exploiting data reuse these overheads are not large enough to lead to larger devices for the implementation of the optimized designs compared to the original one. As expected the on-chip power is increased by about 12% for the copying of data to the data reuse levels.

5.2. Arrays partitioned into sub-arrays

In this case, the arrays corresponding to data reuse levels are partitioned to form a number of sub-arrays that are distributed to different memories (Block RAMs or sets of registers). In our designs, the arrays in the first reuse level with a line of current blocks (LCB) and a line of reference windows (LRW) are vertically divided into equal parts. The results of this exploration are shown in Table 7.

Three partition factors, *i.e.* 2, 4 and 8, have been explored. The designs with one reuse level lead to reduction of execution time up to about 89%, while the designs with two reuse levels lead to larger reduction of execution time up to about 96%. Execution time is reduced more as the number of partitions is increased. Moreover, the number

Table 6. Data reuse exploration results assuming off-chip storage of the input frames and storage of the data reuse levels as single entities: (a) original without data reuse exploitation; (b) one data reuse level for each input frame: (LCB, LRW) stored in Block RAMs; (c) one data reuse level for each input frame: (CB, RW) stored in Block RAMs; (d) one data reuse level for each input frame: (CB, RW) stored in registers; (e) two reuse levels for the previous frame: (LRW) stored in Block RAMs and (RW) stored in registers, and one data reuse level for the current frame: (CB) stored in registers.

Version	Block RAMs	Slices	Cycles /Frame	Freq. (MHz)	Time /Frame (s)	Power on-chip (mW)	Accesses off-chip memory	Smallest device
(a)	0	383	2014228882		0.2	757.18	8312832	XC2V 250
(b)	5	475	12133726	101.5	0.119	822.49	412992	XC2V 250
(c)	2	504	12349008	111.1	0.11	785.73	1051776	XC2V 250
(d)	0	852	3022418	115.9	0.026	775.03	1051776	XC2V 250
(e)	4	1030	2788418	102.2	0.027	834.91	412992	XC2V 250

Table 7. Data reuse exploration results assuming off-chip storage of the input frames and data partitioning of the data reuse arrays: (a) original without data reuse exploitation; (b) one reuse level for each input frame: (LCB, LRW) stored in Block RAMs; (c)two reuse levels for each input frame: (LCB, LRW) stored in Block RAMs and (CB, RW) stored in registers.

Version	Num of data partitions	Block RAMs	Slices	Cycles /Frame	Freq. (MHz)	Time /Frame (s)	Power on-chip (mW)	Accesses off-chip memory	Smallest device
(a)	1	0	383	2014228882		0.2	757.18	8312832	XC2V 250
(b)	2	6	1020	6762886	99.7	0.086	895.09	412992	XC2V 250
(c)	2	6	2356	2036234	98.2	0.021	1016.5	412992	XC2V 500
(b)	4	8	1783	3565942	100.2	0.036	971.58	412992	XC2V 500
(c)	4	8	4562	1202618	92.3	0.013	1151.30	412992	XC2V 1000
(b)	8	16	3551	1988206	90.6	0.022	1180.64	412992	XC2V 1000
(c)	8	16	9279	806546	92.5	0.009	1540.97	412992	XC2V 2000

of accesses to off-chip memories in all cases is reduced by 95%. A difference from the case where arrays are treated as single entities is that most designs exploiting data reuse require larger devices for their implementation compared to the original one because of the increasing amount of slices used. As expected the on-chip power is increased up to double. Power and area overheads increase as the number of partitions and reuse levels increase.

6. CONCLUSION

Data reuse exploration results for a popular video processing kernel have been presented. A systematic four steps approach has been adopted for the exploration of the data reuse options. The combination of data reuse exploitation with coarse grained data-level parallelization has been also explored. Execution time is largely improved when input frames are stored either on chip or off chip, in some cases even by more than 90%. Exploitation of data reuse also leads to significant reduction of the number of off-chip memory accesses, in some cases up to 95%. On-chip power and area (number of on-chip resources) overheads are introduced, especially when data reuse exploitation is combined with data-level parallelization. In this case area overheads may lead to requirement for larger devices compared to the one required for the non-optimized design. Even for an algorithm of small code size such as the full search motion estimation the number of data reuse options is large. Design automation support is required for the efficient evaluation of the trade-offs related to data reuse exploration for realistic algorithms.

7. REFERENCES

[1] http://www.accelchip.com/files/datasheets/AccelChip_DS_6_3_05.pdf.

[2] http://www.mentor.com/products/c based_design/catapult_c_synthesis/index.cfm.

[3] http://www.impulsec.com/C_to_fpga_overview.htm.

[4] http://www.celoxica.com/techlib/default.asp.

[5] M. B. Gokhale, J. M. Stone, , J. Arnold, and M. Kalinowski, "Stream-oriented fpga computing in the Streams-C high level language," in *Proc. Symp. on Field-Programmable Custom Computing Machines (IEEE Computer Society Press 2000)*.

[6] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau, "Spark: a high-level synthesis framework for applying parallelizing compiler transformations," in *Proc. Int. Conf. on VLSI Design, 2003*.

[7] F. Catthoor, E. de Greef, and S. Suytack, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[8] P. Diniz and J. Park, "Automatic synthesis of data storage and control structures for FPGA-based computing engines," in *Proc. of FCCM 2000, IEEE Computer Society Press, 2000*.

[9] M. Weinhardt and W. Luk, "Memory access optimization for reconfigurable systems," in *IEE Proceedings on Computers and Digital Techniques, 1999*.

[10] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.