# Inter-Tile Reuse Optimization Applied to Bandwidth Constrained Embedded Accelerators

Maurice Peemen, Bart Mesman, and Henk Corporaal
Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven the Netherlands
Email: m.c.j.peemen@tue.nl, b.mesman@tue.nl, h.corporaal@tue.nl

***Abstract*—The adoption of High-Level Synthesis (HLS) tools has significantly reduced accelerator design time. A complex scaling problem that remains is the data transfer bottleneck. To scale-up performance accelerators require huge amounts of data, and are often limited by interconnect resources. In addition, the energy spent by the accelerator is often dominated by the transfer of data, either in the form of memory references or data movement on interconnect. In this paper we drastically reduce accelerator communication by exploration of computation reordering and local buffer usage. Consequently, we present a new analytical methodology to optimize nested loops for *inter-tile* data reuse with loop transformations like interchange and tiling. We focus on embedded accelerators that can be used in a multi-accelerator System on Chip (SoC), so performance, area, and energy are key in this exploration. 1) On three common embedded applications in the image/video processing domain (demosaicing, block matching, object detection), we show that our methodology reduces data movement up to 2.1x compared to the best case of *intra-tile* optimization. 2) We demonstrate that our small accelerators (1-3% FPGA resources) can boost a simple MicroBlaze soft-core to the performance level of a high-end Intel-i7 processor.**

## I. Introduction

For many algorithms the compute efficiency is improved orders of magnitude by using specialized hardware accelerators instead of general purpose processor cores [1]. Designers of embedded compute platforms embrace customization to realize the best possible performance within a low power envelope. This trend has successfully brought complex tasks e.g., HD video playback to Smartphones, and in the extreme to wearable devices such as Smartwatches or Glasses. Recently, the relatively long development time of these accelerators, compared to software, is substantially reduced by High-Level Synthesis (HLS) tools. Development time is critical, since video coding standards frequently change, and new applications are constantly introduced, e.g. Instagram.

Fig. 1 illustrates a schematic overview of a heterogeneous platform. The dominant compute workload is performed by dedicated accelerators connected to the host processor. A very complex scaling problem that is not solved by HLS tools is the data transfer bottleneck of accelerators. There is plenty of hardware for parallel compute units, but providing these with the required high-speed data streams is a major challenge. This especially holds for applications in the domain of computer vision and image processing, since the processed high resolution frames do not fit in fast on-chip memories. Hence, frames are accessed from external memory, which has a limited transfer throughput and consumes a lot of energy per access. However, data elements that are often accessed

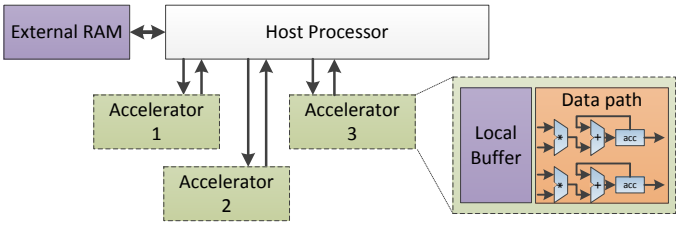


Fig. 1. A host processor with multiple accelerators to achieve high compute efficiency by heterogeneity. Data transfer is reduced by local buffers that exploit data reuse.

can be temporally transferred to small on-chip buffers, so a huge number of transfers can be overcome by efficient reuse of data. To maximize reuse the compute order can be altered, but obtaining the best order is very challenging due to the huge design space.

Our objective is to provide methods that reduce the huge designer effort that is still required to develop efficient hardware accelerators. "Efficient" in this scope is defined as: with minimal data transfer by optimal use of available buffer size. As a result, designers can quickly explore the trade-off space of performance, area, and energy. Our focus is on application kernels described as static nested loops, because these are often responsible for the dominant compute workload and very well suited for acceleration. For data locality optimization combinations of loop interchange and tiling are considered that transform into efficient *tile-strips*. In contrast to others, our method optimizes the *inter-tile* data reuse which opens more reuse opportunities, and is perfectly suited for accelerator controlled local memories. We make the following new contributions:

- Accelerator data transfer model that takes inter-tile reuse into account. (Section IV).
- Pruning of the search space to enable quick design space exploration for the best schedules, given a buffer size (Section V).
- Demonstrator using our technique to equip a simple processor with high performance accelerators (Section VI).
- Evaluation on three common embedded applications, and show that huge speedups can be achieved with a modest amount of buffer size (Section VII and VIII).

First we present related work in Section II, and we continue with a motivational example of iteration reordering (Section III).

## II. Related work

The scheduling of loop iterations for data locality studied for decades [2]. A milestone in this field is the Data Transfer and Storage Exploration (DTSE) methodology [3]. For embedded processors DTSE uses loop fusion and interchange to improve access regularity and locality. More recent works rely on a Polyhedral description [4] of the loop iterations on which automatic transformations are applied that enhance performance. State-of-the-art approaches for x86 CPU code that are used in production compilers are, Pluto [5], and POCC [6]. These works select transformations for communication-minimized parallelization and locality, in some sense this looks very similar to *inter-tile* data reuse optimization. However, their approach is opposite to ours we maximize communication between tiles, and convert that into reuse. Their approach is similar to *intra-tile* reuse optimization i.e., maximize reuse within a tile and minimize communication over tiles, which is good for parallelization and locality. In section VIII (intra-tile opt.), we compare with such scheduling approaches and show significant data transfer reductions.

Exploiting data overlap of successive tiles is introduced only very recently [7], here it is used after optimization to remove redundant transfers. In section VIII we compare to this strategy (inter-tile reuse) and show that it is important to include inter-tile reuse into the tile size selection process. Their followup work [8] parameterizes the tile size selection, so time consuming empirical search methods with the system in the loop can be used to find good parameter configurations.

The tile size selection problem is not solved, with analytical bounds and empirical search methods it is possible to find good performing tile parameters for CPU caches [9]. This work is extended with multi-level caches, conflict misses, and vectorization [10]. We focus on accelerator workloads that rely on more area and power efficient scratchpad memories. For compile-time known access patterns [11] shows that efficient data placement for reuse is more efficient than cache, but in contrast to us they consider the iteration order fixed.

The Halide compiler [12] also focuses on static image processing and computer vision applications, but for x86 and GPUs. In their work Interval Analysis is used for optimization with a stochastic auto tuner that takes up to 2 days to converge to good solutions. In [13] the data reuse optimization problem for FPGA hardware is solved by efficient geometric programming. To use geometric programming a simplified data reuse model is used. In contrast to our method, important properties such as overlap between successive tiles is neglected.

Recently a new tool is developed that optimizes HLS input descriptions for parallelism and locality [14]. This method uses the polyhedral framework for transformations and uses a HLS tool such as Vivado HLS (former AutoESL) to estimate the quality of result. The downside of this approach is its long iteration time; e.g. testing 100 design points can take up to five hours. Secondly, the polyhedral framework generates x86 optimized code with complicated loop bounds resulting in many extra divisions, and min/max operations. In [15] the authors remove some of the x86 artifacts in the generated output code with a HLS friendly code generator, but the fundamental problem of complex bounds remains.

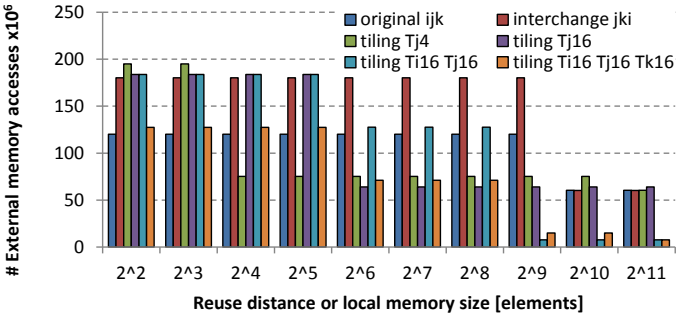


Fig. 2. Data transfer histogram for matrix multiplication, as given in listing 1.

## III. Motivation: Scheduling for data locality

For hardware controlled local memories, the reuse distance [16] is a good metric to predict if a value can be reused given a certain memory size. Reuse distance is defined as: *the number of distinctive data elements accessed between two consecutive uses of the same element.* A transformation of the iteration order of a loop nest can change the reuse distance of the enclosed array accesses. Important transformations for locality optimization are *interchange* and *tiling*.

The effect of transformations is demonstrated on matrix multiplication, the corresponding loop nest is given in Listing 1. For simplicity the result matrix **C** is initialized to 0, and the loop bounds are set to: **Bi**=500, **Bj**=400, **Bk**=300. We used *Suggestions for Locality Optimizations* (SLO) [17] to simulate the remaining data transfers. Remaining transfers are defined as the total number of memory accesses minus the reuses of data elements, as depicted for different buffer sizes in fig. 2. With a very small buffer ($2^2$ elements) only inner loop accesses to **C** are reused. When buffer size increases to $2^{10}$ elements, also rows of **A** are reused.

```
for(i=0; i<Bi; i++){
  for(j=0; j<Bj; j++){
    for(k=0; k<Bk; k++){
      C[i][j] += A[i][k] * B[k][j];
```

Listing 1. Nested loop code for matrix multiplication as a running example.

By loop interchange, loop **i** can be positioned as inner loop, so reuse of **B** is exploited. However, it removed the reuse of **A** and **C**, which increases communication as illustrated in fig. 2. One could better perform tiling of loop **j** with factor **Tj**=4, hence the reuse distance of **A** is reduced to $2^4$ entries. Listing 2 demonstrates tiling in all three dimensions and possibly with different factors. Further experiments, reveal that obtaining the best configuration for a buffer size is a very challenging problem. Even worse is the huge difference in data transfers per configurations, i.e. the design space is very chaotic. E.g., loop tiling with **Ti**=16 and **Tj**=16 gives excellent results for a buffer size of $2^9$, but for $2^8$ it is one of the worst schedules. If the designer could find the best schedules a huge reduction in the number of communications can be achieved, at the cost of a modest amount of buffer area.

```
for(ii=0; ii<Bi; ii+=Ti){
  for(jj=0; jj<Bj; jj+=Tj){
    for(kk=0; kk<Bk; kk+=Tk){
      for(i=ii; i<ii+Ti; i++){
        for(j=jj; j<jj+Tj; j++){
          for(k=kk; k<kk+Tk; k++){
            C[i][j] += A[i][k] * B[k][j];
```

Listing 2. Loop tiling to transfer parts of loop **i**,**j**,**k**, to the inner loops.

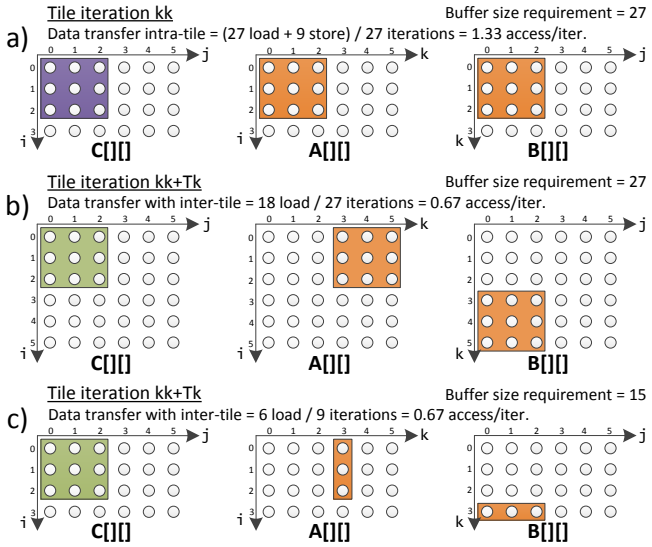


Fig. 3. Data access pattern for for matrix multiplication. a) Only intra-tile reuse, requires transfers for all accessed elements. b) Inter-tile reuse of **C** over successive tiles gives a 2x communication reduction. c) Reducing tile size in the dimension of **k** gives equal reuse with less buffer space.

## IV. Modeling The Scheduling Space

To obtain the best tiling and interchange transformation we define an optimization problem with a cost function that models external transfers, and a bounding function to limit the buffer size.

### A. Modeling intra-tile reuse

For the cost function a loop nest is split into two parts; an inner part for accelerator execution, and an outer part that runs on a host processor. The outer part facilitates the data transfer between external memory and accelerator, while the inner part computes. For the loop nest in Listing 2 the cost function is given below:

$$N\text{tiles} * (\text{datatransfer/tile}) = \left\lceil \frac{B_i}{T_i} \right\rceil \left\lceil \frac{B_j}{T_j} \right\rceil \left\lceil \frac{B_k}{T_k} \right\rceil (2T_iT_j + T_iT_k + T_kT_j) \quad (1)$$

The first part models the *number of tiles* by dividing the domain of each loop by the corresponding tile factor. By ceiling the number of tiles the required padding values for tile factor that are no divisor of the bound are taken into account. As a result our cost function favours minimal padding values, and we can use simple loop bounds instead of complex MIN/MAX operations. The second part of eq. (1) is the *data transfer per tile*, input are the array references, and the size of the tile. The term $2T_iT_j$ models reads and writes to array **C**, array **A** is only read and therefore modeled by $T_iT_k$. Buffer size requirement are modeled as the number of distinct array elements accessed in an inner tile. For Listing 2 this results in eq. (2). The selected tile factors and the array indices model the data volume of an inner tile.

$$T_iT_j + T_iT_k + T_kT_j \leq [\text{Buffer size}] \quad (2)$$

For simple accelerators that overwrite all buffer content after each tile the intra-tile model is correct. However, we could increase reuse by exploiting the data overlap of tiles.

### B. Adding inter-tile reuse to the model

Using the data overlap between tiles increases the complexity of the accelerator, because tiles are not independent anymore. More specific, the first tile (prolog) of a series must be initialized, next successive tiles with data overlap are processed (steady state), finally the last tile (epilog) must be handled correctly to obtain the results. Furthermore, dependencies are created between successive tiles that reduce inter-tile parallelism. Nevertheless, it significantly reduces data transfer, which is a key for accelerators.

Fig. 3a depicts an example, which visualizes data transfer for matrix multiplication. Optimizing for intra-tile reuse with a buffer size constraint of 32 elements results in the tile factors **Ti**=3, **Tj**=3, **Tk**=3. Without inter-tile reuse the host sends 27 values (3x3 patch of **A**, **B**, **C**), and receives 9 values (3x3 patch of **C**) for every tile. If data overlap of successive tiles is exploited, only 18 values (3x3 patch of **A** and **B**) are transferred in the steady state tiles. This schedule is illustrated in fig. 3b, all steady state require 2x less communication.

The cost function in Section IV-A does not take inter-tile reuse into account, so tile factors and interchanges are suboptimal regarding data transfer. The key observation that opens opportunities to find even better schedules is that tiling of the inner control loop has no influence on inter-tile reuse. E.g., if **Tk**=1 as in fig. 3c the number of accesses per compute iteration does not change, but the memory footprint reduces. As a result, not tiling the inner control loop creates opportunities in other dimensions to increase reuse. The data transfer effects with inter-tile reuse are modeled by an expression similar to (1), but considering the full range of the inner control loop a *tile strip*. Hence, the transfers of the prolog, steady state, and epilog are included in the model. Equation (3) shows the updated data transfer model with **kk** as inner control loop.

$$\left\lceil \frac{B_i}{T_i} \right\rceil \left\lceil \frac{B_j}{T_j} \right\rceil (T_iT_j + T_iB_k + B_kT_j) \quad (3)$$

Equation (3) is derived from the intra-tile data transfer expression (1) by subtracting the inter-tile reuses. These reuses of **C** occur between the first load and final store in a sequence of iterations of loop **kk**. From (3) is concluded that $T_k$ does not influence the communication volume, but for buffer size we should minimize $T_k$ as shown in (2). In other words, one dimension gets all reuse for free.

A different inner control loop for the *tile strip* influences the amount of data overlap between successive tiles. By loop interchange **jj** can be the inner control loop, and instead transfers for array **A** are minimized. The corresponding model is given in eq. (4). For complete evaluation each possible inner control direction is separately modeled.

$$\left\lceil \frac{B_i}{T_i} \right\rceil \left\lceil \frac{B_k}{T_k} \right\rceil (2T_iB_j + T_iT_k + T_kB_j) \quad (4)$$

As depicted in fig. 4 the inter-tile models can find schedules that require less accelerator communication. As expected intra-tile reuse is worse for these models but the overall communication is reduced by 1.48x over the best schedule of fig. 3c. This demonstrates that inter-tile reuse must be taken into account during optimization to prevent suboptimal results.

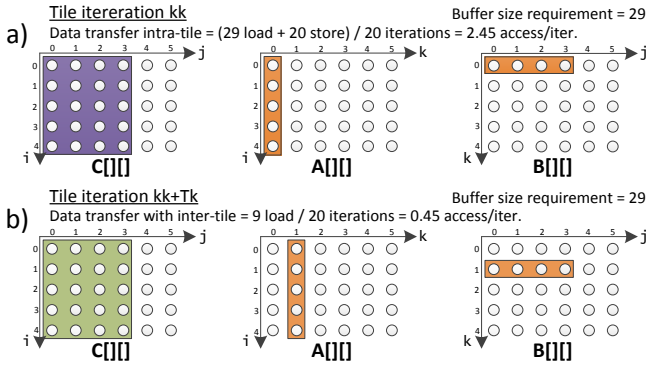


Fig. 4. Inter-tile optimized data access patterns for matrix multiply. a) Intra-tile reuse is less compared to fig. 3a. b) Inter-tile reuse is more than fig. 3b; with tile factor **Tk**=1 there is more reuse of **C** with almost similar buffer size.

## V. Scheduling Space Exploration

When considering transformations as interchange and tiling on nested loops the scheduling space can be huge, as shown in Section III. We use the models proposed in Section IV to obtain the best schedules, and this will take only seconds instead of hours or days as reported by other search methods [14], [12]. This target is achieved by using analytical models that can be evaluated quickly, and using inter-tile reuse optimization, which prunes the search space. Our approach is outlined with a simple convolution kernel, shown in Listing 3. The loop bounds of the example nesting are **Bi**=50 and **Bj**=100.

```
for(i=0; i<Bi; i++){
  for(j=0; j<Bj; j++){
    Out[i] += X[i+j] * H[j];
```

Listing 3. Nested loop description for convolution.

For *intra-tile* optimization different tiling factors should be explored e.g., combinations of **Ti** for loop **i**, and **Tj** for loop **j** that fit the constraints. The search space for this problem is depicted in Fig. 5. However, for *inter-tile* optimization the search space is much smaller, so one loop e.g. **ii** is selected as inner control loop and for the other loop **j**, tile sizes are evaluated. This is also done with the other option **jj** as control loop. Essentially one dimension of the search space is removed. The corresponding cost functions are given in eq. (5), and the buffer size constraint is eq. (6).

$$\text{Cost} = \begin{cases} \left\lceil \frac{B_i}{T_i} \right\rceil (T_i + (T_i + B_j - 1) + B_j) & \textbf{Tj}{=}1 \\ \left\lceil \frac{B_j}{T_j} \right\rceil (2B_i + (B_i + T_j - 1) + T_j) & \textbf{Ti}{=}1 \end{cases} \quad (5)$$

$$T_i + (T_i + T_j - 1) + T_j \leq [\text{Buffer size}] \quad (6)$$

The search for the best configuration is performed by a bounded search through the valid solution space. Since the buffer size requirement function is monotonic, the bounds on the valid solution space can be efficiently set with the guarantee that optimal solutions are obtained. This search method is visualized in Fig. 5. Important to note is the short search time that is required to obtain the best schedules for very deep nested loops. On a standard laptop the search space for our 8-level deep motion estimation kernel is explored in 4.2 seconds, with a buffer size constraint of 1024 elements.

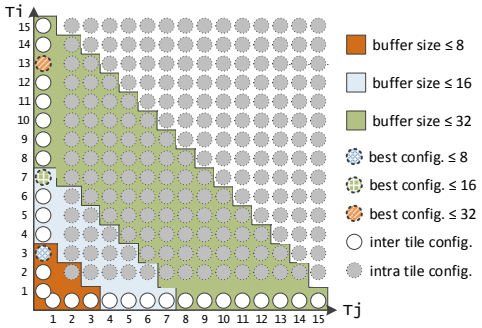


Fig. 5. Design space for tiling configurations on the convolution code in listing 3. Each axis represent a possible tile factor as parameter. For inter-tile reuse optimization one of the tile factors must equal one, which prunes the space. Two options remain **Ti**=1 or **Tj**=1 the other parameter must be optimized. The best configuration given a buffer size, is not always located on the border as shown for a buffer constraint of 32.

## VI. Implementation Demonstrator

Optimized schedules are converted to host processor code and HLS accelerator descriptions according to the template in fig. 1. The required conversions are outlined by the matrix multiplication example with a buffer size constraint of 32 entries. The optimal schedule, derived from Listing 2 and inter-tile reuse optimization, has tiling parameters **Ti**=5, **Tj**=4, and the inner control loop is **kk**.

The host processor code executes the outer control loops, and performs data transfers between host and accelerator. Basically, the outer control loops of listing 2 are used, and prolog and epilog parts are inserted. Finally, the inner control loop is inserted, which transfers the steady state data chunks. In Listing 4, a description of the host code is given. Note that the Send and Receive functions facilitate FIFO based communication. For the connection between the host and accelerator we use the Fast Simplex Links (FSLs) from Xilinx.

```
for(ii=0; ii<Bi; ii+=Ti){
  for(jj=0; jj<Bj; jj+=Tj){
    //prolog part nothing to send
    for(k=0; k<Bk; k++){ //steady state
      Send(A[ii:ii+Ti-1][k]);
      Send(B[k][jj:jj+Tj-1]);
    }
    //epilog part receive results
    Receive(C[ii:ii+Ti-1][jj:jj+Tj-1]); } }
```

Listing 4. Host processor code corresponding to the matrix multiply example.

The accelerator code performs the content of the tiles and has no notion of the position in the program. Furthermore, it describes the read/store policy in the local buffers. The prolog and epilog parts are specified, and in addition the steady state inner control loop is inserted. This last part contains a data transfer and a compute part. If required, the compute part can be parallelized by adding HLS specific pragmas for pipelining or unrolling [18]. Listing 5 shows the accelerator code:

```
Init(C[0:Ti-1][0:Tj-1]); //prolog
for (k=0; k<Bk; k++){ //steady state
  Receive(A[0:Ti-1]);
  Receive(B[0:Tj-1]);
  for(i=0; i<Ti; i++){
    for(j=0; j<Tj; j++){
      C[i][j]+=A[i]*B[j];
} } }
Send(C[0:Ti-1][0:Tj-1]); //epilog return results
```

Listing 5. Accelerator code, which computes on incoming data by executing inner loops of the matrix multiply example.

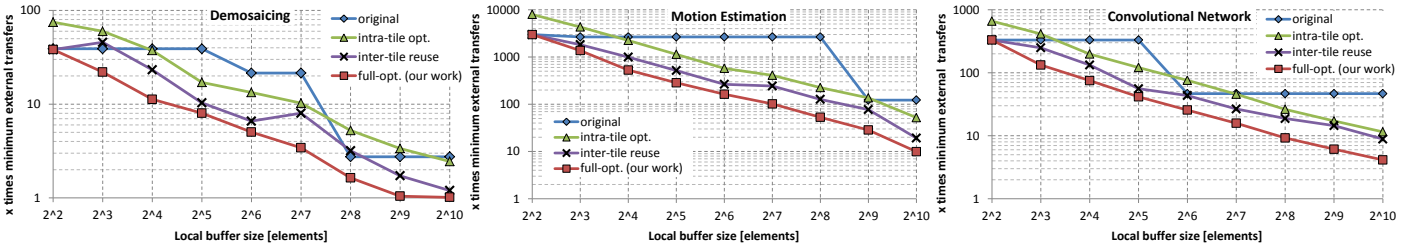


Fig. 6. External data transfers versus accelerator buffer size, for multiple schedules.

## VII. Evaluation Methodology

For evaluation we map three real-world embedded applications from the image and video processing domain, with extensive data transfer requirements. As outlined these applications should contain static affine nested loops that represent the major compute workload. A short overview of the applications is given below, and in addition their sources are made available on the web: http://parse.ele.tue.nl/research/locality/

*1) Demosaicing:* Typically used in camera processing pipelines, since the red, green, and blue (RGB) channels of the sensor are laid out in a Bayer [19] pattern. We use a 5x5 position adaptive interpolation kernel based upon the Malvar-He-Cutler [20] method on an 8 Mpixel input image.

*2) Motion Estimation:* Important for video coding; since it significantly improves compression, though substantially increasing complexity. In modern coding standards, such as H.264, the Integer Motion Estimation (IME) step represents 78% of the compute workload, and 78% of the memory accesses [21]. We use a full-search block matching kernel with a window of 32x32 that searches in a previous and future reference frame for the best matching block, using the Sum of Absolute Differences (SAD) cost function.

*3) Convolutional Network:* For state-of-the-art object detection and recognition, e.g. face detection in photo cameras, and speed sign recognition in navigation devices [22]. In our evaluation we use an optimized speed sign recognition application on a 720p video stream [23], [24].

### A. Platform and tools

As a reference for evaluation fixed-point versions of the applications are mapped to three different platforms:

1) Intel Core-i7 960 CPU at 3.2GHz
2) Arm-A9 CPU at 667MHz, Xilinx Zynq SoC
3) MicroBlaze configured for performance, at 200MHz

In addition, our methodology is used to develop hardware accelerators that increase the performance of a MicroBlaze host processor. We synthesize our designs for the Xilinx ML605 board, which has one Virtex-6 FPGA (xc6vlx240t-1ffg1156). For development we use the Xilinx Vivado 2012.3 tools, including Vivado HLS (AutoESL), which is used to create accelerators. The clock frequency of our FPGA designs is set to 200 MHz.

## VIII. Results

To quantify the effectiveness of our inter-tile reuse optimization strategy we analyze the data transfer effects for the three benchmark applications. For each application the cost functions and buffer size requirements are derived, and the scheduling space exploration is as outlined in section V.

### A. Inter-tile reuse optimization

Fig. 6 shows the data transfer requirements for the original iteration ordering, and three other optimization strategies. The amount of data transfer is specified as a factor with respect to the theoretical minimum, i.e. communicating each input and final output only once. The communication volume is plotted against the required buffer size, which should be as small as possible.

Intra-tile optimization shows the result for an accelerator that resets the buffer contents after each tile, as discussed in section IV-A. This is a naïve optimization target; as a result it sometimes gives worse results compared to the original iteration order. If we exploit the available inter-tile reuse for schedules first optimized for intra-tile reuse the communication or the required buffer size is significantly reduced. Finally, the schedule is optimized for inter-tile reuse, which for all benchmarks results in the least amount of communication at the smallest buffer size.

Important to note is the huge data transfer reduction. E.g., for motion estimation the best schedule reduces data transfer up to 50x with respect to the original. Furthermore, we demonstrate that a small local buffer of 1024 elements can reduce data transfer substantially. For motion estimation and object recognition the remaining number of transfers is within one order of magnitude of the minimum. However, for demosaicing the minimum is already reached with a 512 entry buffer. At this point a designer should stop increasing buffer size, the communication volume stays constant and it only increases area usage.

### B. Quality of results

The quality of our implemented schedules is evaluated by comparing execution time with resource usage. For FPGAs, resource usage is defined as: *MAX(%DSPs, %BRAMs, %LUTs, %flip-flops)*. In fig. 7 the execution time versus resource usage is depicted by plotting different design points. In addition, the area-delay product is used to define the efficiency of different implementations. For each benchmark the best area-delay product is extrapolated and plotted as a dotted line in the comparison.

The results of demosaicing and object recognition behave intuitive. Designs that are balanced score close to the dotted line, while designs severely limited by either compute or buffer resources occur further from this line. As already predicted for demosaicing in fig. 6, increasing the buffer size beyond 512 entries does not improve the design any further. Important to note is that Vivado HLS is a production tool, therefore a small change in the input description can trigger different optimizations. Due to unrolling with a factor two the number of required LUTs in the 32 and 64 entry designs

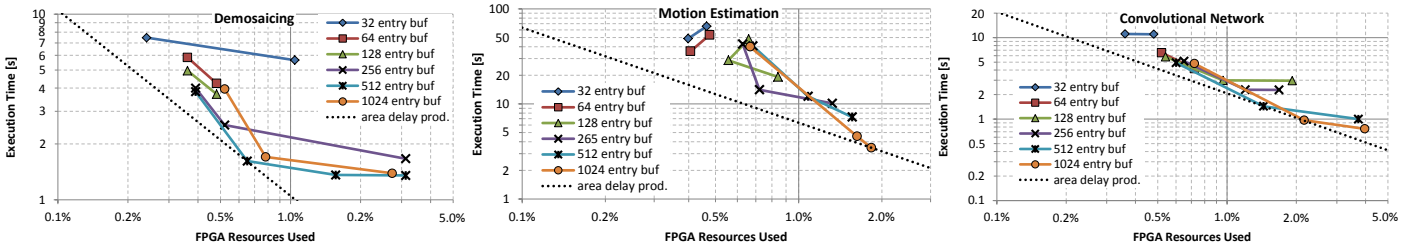


Fig. 7. FPGA resource utilization for accelerator mappings with an optimal iteration order. The best area delay product is extrapolated by the dotted line.

TABLE I. EXECUTION TIME COMPARISON FOR MULTIPLE PLATFORMS

| Platform | Demosaic [s] | Block Match [s] | CNN [s] |
|---|---|---|---|
| Intel-i7 | 0.54 | 8.12 | 0.63 |
| Arm-A9 | 5.75 | 72.32 | 5.92 |
| MicroBlaze | 22.10 | 283.96 | 19.05 |
| Accelerator | 1.36 | 3.45 | 0.75 |

are reduced. This reduces overall resource usage and the area delay product. Finally, we compared the best accelerators with other platforms, as shown in table I. The accelerators do not have a dedicated DMA controller and are therefore constraint by communication bandwidth. However, the accelerators can increase the original MicroBlaze performance by 16 to 82 times, at the cost of a very small increase of overall resource usage. Hence simple embedded processor can perform on par with a high-end general purpose processor. Dedicated DMA can be added, but it will only shift the result of fig. 7.

## IX. CONCLUSION

In this paper we presented our new inter-tile reuse optimization strategy for nested loop accelerators. This optimization strategy searches the best combination of loop interchange and tiling with optimal tile sizes. We demonstrated that maximizing efficiency for local buffers in FPGA based accelerators gives a substantial performance improvement. These improvements are enabled by efficiently exploiting local buffering with data access optimizations for nested loops. Our optimizations manage to limit buffering requirements while achieving a significant reduction of external data transfer. The main improvement is achieved by optimizing for inter-tile reuse. Although the design space can be huge and chaotic for deeply nested loops, we show that the best configuration of transformations can be found with a model-based approach. Our focus on inter-tile reuse effectively prunes the search space of configurations, and limits the effort of exploration to mere seconds. The fruits of our optimizations are verified with the Xilinx Vivado HLS tools, and we observe a significant reduction in the effort of designing efficient dedicated accelerators. With our approach the number of required design iterations is minimized, by directly computing the best candidates before time consuming synthesis. As a result, the mapping process of static image or video processing applications to dedicated hardware accelerators is much better manageable.

## REFERENCES


[1] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," ser. ISCA '10, 2010.

[2] M. Wolf and M. Lam, "A data locality optimizing algorithm," in *Proc. PLDI '91*, 1991, pp. 30–44.

[3] F. Catthoor, K. Danckaert, C. Kulkarni, E. Brockmeyer, P. G. Kjeldsberg, T. Van Achteren, and T. Omnes, *Data access and storage management for embedded programmable processors*, 2002.

[4] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model," in *ETAPS CC*, 2008.

[5] U. Bondhugula, J. Ramanujam, and P. Sadayappan, "Pluto: A practical and fully automatic polyhedral program optimization system," in *ACM SIGPLAN (PLDI)*, 2008.

[6] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan, and N. Vasilache, "Loop transformations: convexity, pruning and optimization," in *POPL 38*, 2011.

[7] C. Alias, A. Darte, and A. Plesco, "Optimizing remote accesses for offloaded kernels: Application to high-level synthesis for fpga," in *DATE*, 2013.

[8] A. Darte and A. Isoard, "Parametric Tiling with Inter-Tile Data Reuse," in *IMPACT 2014*, Vienna, Austria, 2014.

[9] J. Shirako, K. Sharma, N. Fauzia, L.-N. Pouchet, J. Ramanujam, P. Sadayappan, and V. Sarkar, "Analytical bounds for optimal tile size selection," in *Proceedings of the 21st International Conference on Compiler Construction*, 2012, pp. 101–121.

[10] S. Mehta, G. Beeraka, and P.-C. Yew, "Tile size selection revisited," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, pp. 35:1–35:27, 2013.

[11] I. Issenin, E. Brockmeyer, M. Miranda, and N. Dutt, "Drdu: A data reuse analysis technique for efficient scratch-pad memory management," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 2, 2007.

[12] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *PLDI*, 2013.

[13] Q. Liu, G. Constantinides, K. Masselos, and P. Cheung, "Combining data reuse with data-level parallelization for fpga-targeted hardware compilation: A geometric programming framework," *Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, 2009.

[14] L.-N. Pouchet, P. Zhang, P. Sadayappan, and J. Cong, "Polyhedral-based data reuse optimization for configurable computing," in *Proceedings of the ACM/SIGDA international symposium on FPGA*, 2013, pp. 29–38.

[15] W. Zuo, P. Li, D. Chen, L.-N. Pouchet, S. Zhong, and J. Cong, "Improving polyhedral code generation for high-level synthesis," in *CODES+ISSS*, 2013.

[16] C. Ding and Y. Zhong, "Reuse distance analysis," University of Rochester, Tech. Rep., 2001.

[17] K. Beyls and E. D'Hollander, "Refactoring for data locality," *Computer*, vol. 42, no. 2, pp. 62–71, feb. 2009.

[18] Xilinx, *Vivado Design Suite User Guide, High-Level Synthesis, UG902*.

[19] B. Bayer, "Color imaging array," US Patent Patent 3,971,065, 1976.

[20] H. Malvar, L.-W. He, and R. Cutler, "High-quality linear interpolation for demosaicing of bayer-patterned color images," in *IEEE International Conference on Acoustics, Speech, and Signal Processing,*, 2004.

[21] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, "Analysis and architecture design of an hdtv720p 30 frames/s h.264/avc encoder," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, pp. 673–688, 2006.

[22] M. Peemen, B. Mesman, and H. Corporaal, "Speed sign detection and recognition by convolutional neural networks," in *8th International Automotive Congress*, 2011, pp. 162–170.

[23] ——, "Efficiency optimization of trainable feature extractors for a consumer platform," in *ACIVS*, 2011, pp. 293–304.

[24] M. Peemen, A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *31st International Conference on Computer Design (ICCD)*, 2013.