DATA MEMORY OPTIMIZATION IN LTE DOWNLINK

Namita Sharma[1], Tom Vander Aa[2], Prashant Agrawal[2], Praveen Raghavan[2], Preeti Ranjan Panda[1], and Francky Catthoor[2]
[1]Indian Institute of Technology Delhi, New Delhi, India
[2]IMEC, Leuven, Belgium

ABSTRACT

Optimizations related to memory accesses and data storage make a significant difference to the performance and energy of a wide range of data-intensive applications. Such strategies need to evolve with modern SoC and processor architectures, which lead to new optimization opportunities. In this paper, we focus on data memory optimization for LTE downlink receiver as this is a data- and computation-intensive part of the LTE application with tight energy and latency constraints. We study the data dependencies globally and conclude that by providing data samples from the antennas in interleaved form at the FFT input, we can achieve 7-15% reduction in memory access energy over an optimized implementation without any performance overhead.

Index Terms — Data Layout, Memory Optimization, LTE

1. INTRODUCTION

Data and memory optimizations have been the topic of a significant amount of recent research efforts because of the dominating role of memory in a large class of data-intensive applications. LTE is such a data-intensive standard for wireless communication with useful applications in mobile phones and data terminals. LTE defines several channel bandwidths: 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, and 20 MHz [1]. The larger the bandwidth, the higher the channel capacity, and the greater the number of available sub-carriers for transmission. With the use of multiple antennas at both ends of the communication system (MIMO technique) a linear increase in the channel capacity can be obtained for the same bandwidth. Due to the processing requirement for large number of sub-carriers, practical system implementation of this standard are subject to data memory organisation and access bottlenecks. We focus on the LTE MIMO downlink receiver as it is the data-intensive and computation-intensive part with tight energy and latency constraints. Some solutions for energy and performance improvement at different stages of the receiver have been proposed [2], [3].

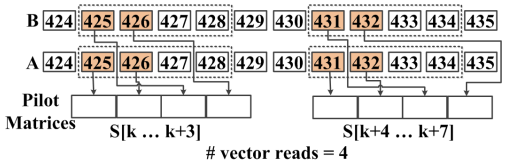
Data layout optimizations aim to arrange data in memory with the objective of improving system performance and energy through means such as reduced memory access count or reduced cache misses. Several more generic data layout techniques have been explored by researchers at various levels in the memory hierarchy. But these are not well matched to our context of LTE optimisations. In particular, Cache partitioning [4], a layout technique for arrays, maps each array to different cache partitions to reduce conflicts. Kulkarni et al. [5] addressed the problem for cache miss reduction by evaluating the tiling size for arrays and merging the arrays appropriately for each loop nest. Memory Hierarchy Layer Assignment (MHLA), an optimization methodology for data memory hierarchy [6], determines for each data set an appropriate layer in the hierarchy and type of memory (single/dual port) taking data re-use into account. The strategy in [7] partitions the variables to Scratch Pad Memory and DRAM in a way that interference among different variables is minimized. To the best of our knowledge, no work has been presented yet for an efficient data layout for LTE. And the cache-oriented general techniques proposed earlier by researchers [8] do not find a straightforward application in our context, where the target system hardware consists of a coarse grained re-configurable array (CGRA) of SIMD functional units and vector registers.

2. MOTIVATING EXAMPLE

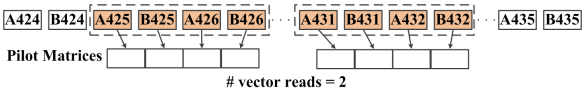
We illustrate the motivation behind the data layout transformation with the following example. Figure 1(a) shows a code snippet from the *Channel Estimation* stage of the LTE application that involves pilot extraction followed by product with constant conjugate pilots. Here, arrays *A* and *B* correspond to the sample space of the two antennas. Only a subset of these arrays is used in the data processing block. Figure 1(b) shows the access pattern, with the accessed array elements highlighted.

```
int k=0;
for(i=424; i<1624; i+=12) {
/* Extracting Pilot Matrix 1 */
   S[k]=A[i+1]*C0;
   S[k+1]=A[i+2]*C1;
   S[k+2]=B[i+1]*C2;
   S[k+3]=B[i+2]*C3;
/* Extracting Pilot Matrix 2 */
   S[k+4]=A[i+7]*C4;
   S[k+5]=A[i+8]*C5;
   S[k+6]=B[i+7]*C6;
   S[k+7]=B[i+8]*C7;
   k=k+8;
}
```

(a) Example loop



(b) Array access pattern (Layout 1)



(c) Array access pattern with interleaving (Layout 2)

Fig. 1: Motivational example from LTE

For these data sets, Figure 1 shows two possible layouts. Assuming that a vector register READ operation loads 4 consecutive words from the data memory into the vector register, in Layout 1 (Figure 1(b) which is the normal array storage strategy), two vector reads are required to load the elements for each pilot matrix. Here, the loads bring along the irrelevant data also, including A[427], A[428], etc. Layout 1 is conventionally chosen as the inputs and outputs of the OFDM demodulation block are all in sequential order. Consider Layout 2 shown in Figure 1(c) where elements of A and B are stored in an *interleaved* manner. Here, only one memory access per pilot matrix is required. Thus, a suitable layout in the memory reduces the number of memory accesses. This creates an important exploration problem, since the complexity increases if we have multiple arrays that can be interleaved. The choices of the subset

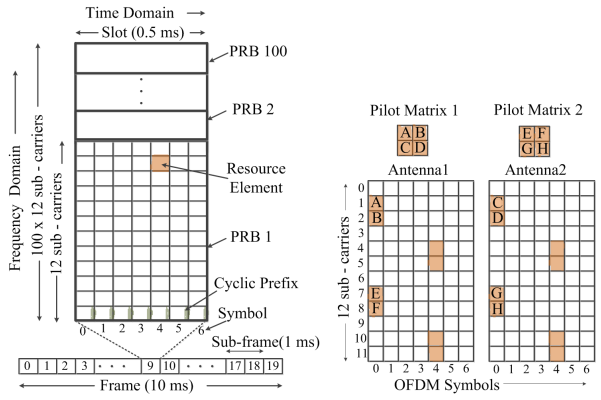
of arrays to be interleaved and the granularity of interleaving, have to be comprehensively evaluated so as to produce energy-efficient implementations without compromising on performance.

3. APPLICATION OVERVIEW

In this section, we explain the LTE application code structure with a brief introduction to the channel configuration.

3.1. Basics and Terminology

A specific number of *sub-carriers* (Figure 2(a)) in the frequency domain are allocated to possibly multiple users for a pre-determined amount of time. The communication channel is thus configured along both the time and frequency domains. Transmission is based on *frames* of 10 ms duration. Each frame consists of 10 *sub-frames* that are divided into 2 *slots* each as shown in Figure 2(a). Each slot is made up of 6 to 7 OFDM *symbols* [1] depending on the *cyclic prefix* length associated with them. A sub-frame is the smallest unit of time allocated to a user. The channel bandwidth is divided into *Physical Resource Blocks (PRBs)*. Each PRB consists of 12 sub-carriers of 15 kHz bandwidth and is constant irrespective of the bandwidth under consideration. Since their frequencies are mutually orthogonal, there is no need for a guard frequency to avoid inter-channel interference. *Resource element* is the smallest constituent of the resource grid and represents one single sub-carrier for one symbol period. Number of resource elements in a PRB is $12 \times \#OFDM\ symbols\ in\ a\ slot$. In order to facilitate carrier offset estimation, channel estimation and timing synchronization, some of the resource elements are reserved for the transmission of special reference symbols (*pilots*) according to 3GPP standards and are not available for user data. Figure 2(b) shows the pilot distribution for LTE 2x2 [9].



(a) Downlink Resource Grid (b) Pilot distribution for LTE 2x2

Fig. 2: Ref. elements distribution for channel BW=20 MHz

3.2. LTE Details

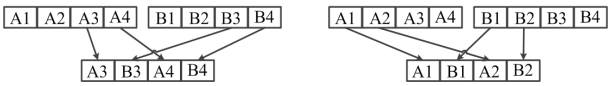
LTE 2x2 MIMO application has two antennas at the transmitter as well as the receiver end. Pilots are positioned at symbols 0 and 4 in each slot (Figure 2(b)). Considering Symbol 0, sub-carriers at positions 1 and 2 (A/B and C/D) in the frequency domain of PRBs corresponding to both the antennas constitute one pilot matrix, while those at positions 7 and 8 (E/F and G/H) constitute the second matrix. Similarly, the pilot matrix elements for symbol 4 are positioned at 4, 5 and 10, 11 in the PRBs for the two antennas. Thus, we have two pilot matrices associated with each of the pilot symbols 0 and 4. Figure 4 illustrates the complete application flow. The OFDM demodulation step consists of the *IQ CFO Compensation* kernel followed by *FFT* to move from time domain to the frequency domain. Output of *FFT* goes to the *Shuffle Carrier Pilot0/4* kernel for the pilot symbols 0/4 and to *Shuffle Carrier* for non-pilot symbols. These kernels separate out the data and pilot carriers (for pilot symbols) from the dummy sub-carriers both at the start and end of the band (424 on each side) and the DC carrier. The *Channel Estimation* kernel computes the channel estimate matrices (H matrices) and the rotational angles based on pilot symbols. It does the preparation for interpolation along the frequency domain in the *Frequency Interpolation* block. Here, the H matrices for reference carriers are linearly interpolated along the frequency domain to obtain H matrices for all the sub-carriers corresponding to each PRB. The *Pre-Scaling* kernel does a linear scaling of matrices by a computed pre-coding constant matrix. Following this, QR decomposition method is used for matrix inversion to compute the equalizing matrix in the *Matrix Inversion* kernel. From the point of pilot extraction to the matrix inversion, the flow is the same for both pilot symbols. After this, the *LLR* block is executed where Log-Likelihood Ratios (LLR) are calculated based on the inverse channel matrices of Symbol 0. The Δ *Inverse* computation block for Symbol 4 uses the inverse matrices of both pilot symbols to prepare for interpolation along the time domain. Channel estimates are obtained for the intermediate non-pilot symbols in kernel *Payload*. These are then forwarded to *Payload* together with the received data signals for retrieval of the transmitted information bits. The *Payload* processing for symbols 1-4 can be done only after the *Channel Estimation* for symbol 4. This leads to a large storage requirement for data buffering. The *Payload* processing for symbols 0, 5, and 6 can be done without any extra storage overhead.

4. EXPLORATION FOR INTERLEAVING DECISION

Interleaving is a data layout transformation for combining the storage of multiple arrays, so that blocks of data from different arrays are stored contiguously, in order to achieve spatial locality in memory accesses. For the LTE application under consideration, the sample space of each of the antennas is represented by an array of size $2K$, the number of samples per antenna. By interleaving, the functionality of the *Shuffle Carrier Pilot* kernel (Figure 4) to group the pilot elements from the different arrays can be achieved. Reading out elements for the pilot matrices requires loading of pilot elements from each of the arrays separately (Figure 1(b)). From the application knowledge, we know that the position of elements to be fetched is the same for all the antennas. So, by interleaving we are able to group the data to be extracted and thus reduce the number of memory accesses for loading the same pilot matrices as observed in Figure 1(c).

4.1. Impact of Interleaving

Propagating the dependencies backwards from the point of extraction of pilot elements (*Shuffle Carrier Pilot* kernel), where we need an interleaved data set, the output of FFT is required to be in interleaved form. This poses a requirement of interleaved data samples at the input of FFT, that can be made possible by writing out the output of *IQ CFO Compensation* kernel in this form. To achieve this data re-arrangement we need two interleave operations for each pair of vector loads – one each for least significant (LS) words and most significant (MS) words as shown in Figure 3. Both these operations are single-cycled in the processor we consider.



(a) For LS words (b) For MS words

Fig. 3: Interleave operation

The complete sample space (*#elements*) to be re-arranged is $N \times ArraySize$, where N is the number of arrays to be interleaved and

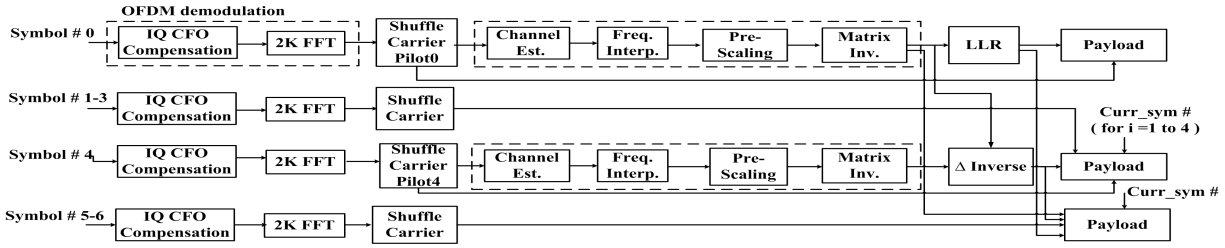


Fig. 4: High level overview of the LTE 2x2 application

ArraySize is the number of samples associated with each of the antennas (=FFT size). Thus, the number of vector loads (*#Vector loads*) for shuffling the data becomes $\#elements/W$, where W is the SIMD width. Since interleaving at each stage is done between a pair of arrays, interleaving N arrays takes $k = log_2N$ stages. As two interleave operations are required for each pair of vector loads, the number of additional operations for interleaving at the output of *IQ CFO compensation* kernel becomes

$$\#Extra\ Ops,\ A = 2 \times \frac{\#Vector\ loads}{2} \times log_2N \tag{1}$$

After *IQ CFO compensation*, *FFT* is to be performed on each array. With interleaved samples at the input, now instead of performing FFTs for each antenna data set (*Array*) separately, we compute one FFT with 11 stages (for *2K* size) on *2K*×*N* samples. As a result we obtain interleaved FFT outputs. Performing FFT on all arrays together reduces the number of coefficient (twiddle factors) accesses. With the re-arranged data set, the samples to be operated upon with the same twiddle factors are accessed together, thus preventing the repeated coefficient loads by a factor equal to the number of antennas, *N*. Following *FFT* we have the *Shuffle Carrier*/*Shuffle Carrier Pilot* kernel which, after interleaving, has been removed as its functionality of grouping the pilot elements has been achieved by interleaving. This reduces the memory accesses required to fetch all samples, vector-pack operations (involving extraction and insertion of pilot elements), and finally writing back to memory the pilot and data sub-carriers. In *Channel Estimation* kernel, the variation in number of memory accesses, if any, is taken into account by the array access computations. Proceeding from *Channel Estimation*, all kernels until *Payload* have no impact due to interleaving. In the *Payload* kernel, we need to de-interleave the data sub-carriers before processing. De-interleaving at each stage brings together elements of an array separated by $W/2$ as shown in Figure 5. Thus, to obtain contiguous arrangement interleaving is done in log_2W stages, leading to additional operations given by

$$\#Extra\ Ops,\ B = 2 \times \frac{\#Vector\ loads}{2} \times log_2W \tag{2}$$

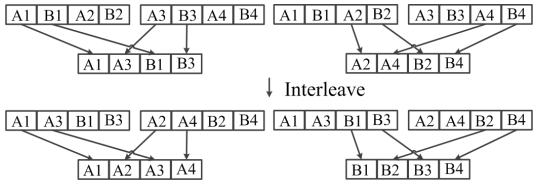
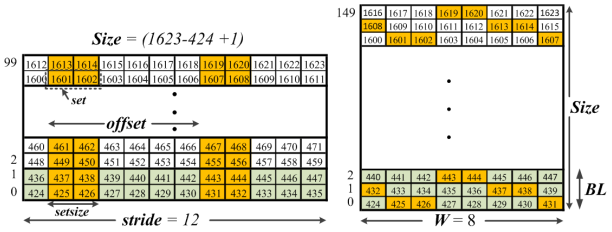


Fig. 5: De-interleave operation

4.2. Global Trade-off Analysis and Overall Strategy

Interleaving may not always result in improved performance or energy. The layout decision needs to be made taking into account the overall objectives and constraints. For the SDR standard application under consideration, our focus is to achieve energy optimization for wireless devices without compromising on device performance. Implementing interleaving requires additional operations that may lead to overall performance degradation. Thus, there is a need for a quantitative estimate of its impact. The complexity of the decision increases when we have several candidate arrays for interleaving (at least $\theta(2^{NM})$ when there are N arrays in M loops). The subset of arrays to be interleaved, and the granularity of interleaving or interleaving width, have to be chosen after a proper estimation of the expected savings and associated overheads.

We begin with computing the total memory accesses for the kernel mappings for both the cases – without and with interleaving – by summing up the accesses associated with each array of the kernel. The memory access count is estimated from the following parameters: *sets* of *setsize* consecutive elements at intervals of *offset* accessed in each loop iteration, the loop stride (*stride*), SIMD width (W) and the array size (*Size*). Figure 6(a) illustrates the accesses for an array (A



(a) Logical access pattern (1 row = 1 loop iteration) (b) Physical access pattern (1 row = 1 memory access)

Fig. 6: Access patterns for loop in Figure 1(a)

or B) corresponding to the loop shown in Figure 1(a) with an iteration count of 100 and a *stride* of 12. Figure 6(b) shows the accesses made with vector registers of size W for the same array. Comparing these figures, we conclude that the access pattern in the block of length $BL = L.C.M.(stride, W)$ (= $L.C.M.(12, 8)$ = 24), is repeated over the entire array space. An array of size $Size(= UB - LB + 1$; UB and LB denote the upper and lower loop bounds respectively), is sub-divided into $\left\lceil \frac{Size}{BL} \right\rceil$ blocks. Since the *sets* are positioned at fixed offset *offset*, the number of *sets* per block $= BL/offset$. There are 2 possible cases.

- $offset \geq W$. In this case only one *set* can be loaded per set of memory accesses of W width. Number of accesses per *set* is given by $\left\lceil \frac{setsize}{W} \right\rceil$ and the memory accesses per block

$$acc\_per\_blk = \frac{BL}{offset} \times \left\lceil \frac{setsize}{W} \right\rceil \tag{3}$$

- $offset < W$. Here multiple *sets* may be accessed per vector load. Assuming f *sets* are accessed in k vector loads, the desired values of (k, f) will be the minimum of the positive integral solutions for Equation 4 with k bounded by the maximum number of accesses per block (= BL/W)

$$setsize + (f-1)offset = k \times W \tag{4}$$

With f *sets* accessed in k vector loads, the number of accesses corresponding to *sets* in a block becomes

$$acc\_per\_blk = \left\lceil \frac{BL}{offset} \times \frac{k}{f} \right\rceil \quad (5)$$

If there does not exist an integral solution to Equation 4 then, memory accesses per block is bounded by BL/W which leads to loading all the elements of a block.

In both cases, the total number of memory accesses is given by

$$tot\_mem\_acc = \left\lceil \frac{Size}{BL} \right\rceil \times acc\_per\_blk \quad (6)$$

For each symbol s, we compute the memory accesses saved over the non-interleaved mapping. We then compute the performance *Overhead* for the mappings with interleaving over the one with contiguous mapping using the statically obtained operation details per kernel. If the overhead is less than the estimated savings, then the decision for interleaving is justified.

5. EXPERIMENTAL RESULTS

5.1. Framework

We have used an extension of the DRESC compiler framework [10] for mapping the reference application to a CGRA architecture (Figure 7). The CGRA part comprises four SIMD enabled functional units (FUs) and a central vector register file which is also coupled to a VLIW processor with 3 FUs. For efficient utilization of the vector FUs, the register file has a wide interface (256 bit wide) with the two scratch pad memories (SPM), while the one with VLIW FUs is 32 bit wide. An XML based language is used to describe the architecture. The processor simulator provides run time statistics to compare the memory accesses and performance for the kernel mappings. We synthesized the HDL models of the processor using Cadence RTL compiler for TSMC 40nm standard library and carried out Power simulations on the synthesized design using Synopsys Primepower. Delay and energy numbers for SPM are derived from a commercial memory compiler.

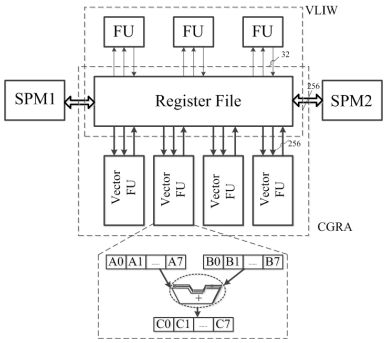
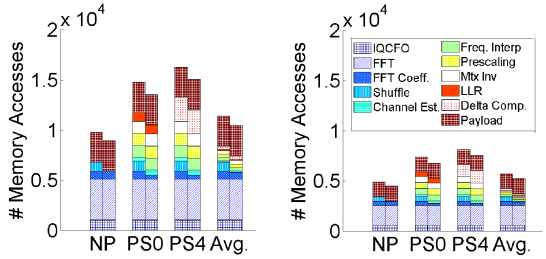


Fig. 7: CGRA processor

5.2. Exploration Experiments for LTE

Based on our interleaving decision strategy discussed in Section 4.2, we conclude that interleaving should be done only for the pilot symbols in a slot. For non-pilot symbols, though there is a gain with respect to memory access energy, the performance overhead is very high. Interleaving implicitly implements one of the functions of *Shuffle* kernel to group the pilots. The other one for extracting out the data carriers can be postponed until *Payload* processing. Thus, *Shuffle* kernel can be completely removed for all the symbols. The kernels are re-written for studying the effect of interleaving on the memory access variations due to different architectural parameters for the LTE 2x2 application mapping on to the CGRA processor.

Variation with SIMD width: Memory access count is inversely proportional to the SIMD width. With varying SIMD width, the memory access counts associated with each of the kernels changes by the same factor. Thus, the overall gain remains the same with SIMD width variations as shown in Figure 8. For each non-pilot (NP) symbol, the gain is 8.68%, while the gains are 8.37% and 7.72% for pilot symbols 0 and 4 respectively. Averaging out over the time slot (Avg. in figure) 8.51% reduction in memory accesses is achieved.



(a) SIMD width = 8 (b) SIMD width = 16

Fig. 8: Non-interleaved vs. interleaved storage (BW = 20MHz)

Variation with number of occupied PRBs: The maximum number of PRBs associated with 20 MHz channel bandwidth is 100. However, since 100% PRB occupation occurs rarely in practice, we study the effect of interleaving with the number of occupied PRBs as a parameter. A substantial improvement in gain is achieved with reducing number of occupied PRBs. All the data samples have to be processed until the *Channel Estimation* stage, while from MIMO detection stage, the lower the number of occupied PRBs, the lower the computations and corresponding memory accesses. Thus, the percentage reduction in memory accesses with unchanged gain from *Shuffle* kernel shows an approximately linear increase for each of the symbols. The average gain over a time slot varies from 8.57% for 100 PRBs to 14.33% when the number of occupied PRBs is one.

In the above mappings, the percentage reduction in energy is directly proportional to the corresponding reduced number of memory accesses. However, interleaving leads to additional vector register accesses, with a corresponding energy increase. We observe that in the worst case, when all PRBs are occupied at maximum bandwidth of 20 MHz, the additional register accesses lead to a 1.07% reduction in gain due to reduced memory accesses over a time slot. This effect becomes negligible with reducing number of PRBs because of the reduced interleave operations in *Payload* and an unchanged gain from *Shuffle* kernel. Similar studies performed on other configurations of the LTE 2x2 application are not shown due to space limitations. Our approach is also suitable for signal processing applications involving data compression for audio and video signals. Here, in the coding layer, frames of samples are created by grouping samples from each of the sub-bands, which is similar to the case of pilot matrices extraction from the output of FFT in LTE.

6. CONCLUSION

LTE is a memory dominant application with SDR because the number of samples for FFT computation is very large, with significant data buffering in the local memory in payload processing for non-pilot symbols. This causes system implementations to suffer from data memory organisation bottlenecks. We presented an approach to improve memory energy by reducing memory accesses using an efficient data layout targeting a CGRA with SIMD structure. We conclude from global analysis of kernels that significant reduction (7-15%) in data memory energy consumption can be achieved if array data are interleaved, with no performance overhead.

7. REFERENCES

[1] JDSU, *LTE PHY Layer Measurement Guide*. Application Note, 2011.

[2] S. Schwarz, M. Wrulich, and M. Rupp, "Mutual information based calculation of the Precoding Matrix Indicator for 3GPP UMTS/LTE," in *International ITG Workshop on Smart Antennas (WSA)*, Feb. 2010, pp. 52 –58.

[3] L. Ruiz de Temino, C. Navarro I Manchon, C. Rom, T. Sorensen, and P. Mogensen, "Iterative Channel Estimation with Robust Wiener Filtering in LTE Downlink," in *IEEE 68th Vehicular Technology Conference*, Sept. 2008, pp. 1 –5.

[4] N. Manjikian and T. S. Abdelrahman, "Array Data Layout for the Reduction of Cache Conflicts," in *In Proceedings of the 8th International Conference on Parallel and Distributed Computing Systems*, 1995.

[5] C. Kulkarni, C. Ghez, M. Miranda, F. Catthoor, and H. De Man, "Cache conscious data layout organization for conflict miss reduction in embedded multimedia applications," *IEEE Transactions on Computers*, vol. 54, no. 1, pp. 76 – 81, Jan 2005.

[6] E. Brockmeyer, M. Miranda, and F. Catthoor, "Layer assignment techniques for low energy in multi-layered memory organisations," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pp. 1070 – 1075.

[7] P. R. Panda, A. Nicolau, and N. Dutt, *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[8] P. R. Panda, N. D. Dutt, A. Nicolau, F. Catthoor, A. Vandecappelle, E. Brockmeyer, C. Kulkarni, and E. de Greef, "Data Memory Organization and Optimizations in Application-Specific Systems," *IEEE Design & Test of Computers*, vol. 18, no. 3, pp. 56–68, 2001.

[9] Technical Specification Group Radio Access Network, "3GPP TS 36.211 V8.9.0 (2009-12)," 3rd Generation Partnership Project (3GPP), Tech. Rep., Release 8, 2009.

[10] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "DRESC: a retargetable compiler for coarse-grained reconfigurable architectures," in *IEEE International Conference on Field-Programmable Technology (FPT) Proceedings*, Dec. 2002, pp. 166 – 173.