Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Su-Shin Ang · George Constantinides · Wayne Luk · Peter Cheung

Custom Parallel Caching Schemes for Hardware-accelerated Image Compression

Received: date / Revised: date

Abstract In an effort to achieve lower bandwidth requirements, video compression algorithms have become increasingly complex. Consequently, the deployment of these algorithms on Field Programmable Gate Arrays (FPGAs) is becoming increasingly desirable, because of the computational parallelism on these platforms as well as the measure of flexibility afforded to designers. Typically, video data is stored in large and slow external memory arrays, but the impact of the memory access bottleneck may be reduced by buffering frequently used data in fast on-chip memories. The order of the memory accesses, resulting from many compression algorithms are dependent on the input data [18]. These data dependent memory accesses complicate the exploitation of data re-use, and subsequently reduce the extent to which an application may be accelerated. In this paper, we present a hybrid memory sub-system which is able to capture data re-use effectively in spite of data dependent memory accesses. This memory sub-system is made up of a custom parallel cache and a scratchpad memory. Further, the framework is capable of exploiting two dimensional spatial locality, which is frequently exhibited in the access patterns of image processing applications. In a case study involving the Quad-tree Structured Pulse Code Modulation (QSDPCM) application, the impact of data dependence on memory accesses is shown to be significant. In comparison with an implementation which only employs an SPM, performance improvements of up to $1.7\times$ and $1.4\times$ are observed through actual implementation on two modern FPGA platforms. These performance improvements are more pronounced for image sequences exhibiting greater inter-frame movements. In addition, reductions of on-chip memory resources by up to $3.2\times$ are achievable using this framework. These results indicate that, on custom hardware platforms, there is substantial scope for improvement in the capture of data re-use when memory accesses are data dependent.

S-S. Ang · G.A. Constantinides · P.Y.K. Cheung
Department of Electrical and Electronic Engineering, Imperial College London, South Kensington Campus, London SW7 2AZ
E-mail: su-shin.ang04@imperial.ac.uk

W. Luk
Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ

1 Introduction

Contemporary video processing applications involve the exchange of information with increasing quality and content. To achieve the required quality of service, without consuming excessive bandwidth, video compression algorithms have become increasingly important. Several algorithms have been successfully designed to obtain higher compression ratios [18]. However, these algorithms often lead to increased encoding and decoding complexity, providing grounds for deployment on custom hardware platforms. In addition, the flexibility afforded by field programmable hardware is invaluable due to rapidly evolving processing standards. On the other hand, the memory access patterns of these algorithms are often dependent on the input data. Therefore, while significant compression ratios may be achieved, lower required bandwidth does not necessarily translate into better performance because the indeterminate memory accesses of the compression algorithms complicate the exploitation of data re-use in hardware. Consequently, there is a critical need to develop memory sub-systems on FPGA platforms, which are capable of handling data dependence, so that the benefits of compression algorithms may be fully exploited.

Image data is typically stored in large external memory arrays where access times are considerable. To reduce the impact of the external memory bottleneck, frequently re-used data may be identified beforehand and buffered in fast on-chip memories so that repeated accesses to external memory may be avoided. However, identifying frequently re-used data accurately is difficult if the memory accesses are data dependent. In this work, a custom parallel caching methodology is developed with the objective of exploiting data re-use effectively in the presence of data dependent memory accesses. In addition, the cache addressing scheme permits the two dimensional spatial locality of memory accesses, frequently exhibited in video applications, to be exploited. Since the proposed scheme is fully parameterisable, it is

well-suited for FPGAs as the parameters of the memory sub-system may be flexibly adjusted to the application context. Further, the substantial on-chip memory bandwidth of FPGAs may be harnessed in the caching methodology to fully exploit the memory access parallelism inherent in the algorithm.

The caching methodology is applied in conjunction with a scratchpad memory (SPM) in a case study involving the quad-tree structured pulse code modulation (QSDPCM) algorithm. This algorithm is particularly suited for hardware implementation [18] because the computational parallelism in the algorithm may be easily exploited. However, the memory accesses in the algorithm are data dependent, making current approaches to data re-use exploitation sub-optimal in terms of storage requirements and performance. We demonstrate that further improvements in both storage requirements and performance are possible using the proposed hybrid memory sub-system. Specifically, compared with a scheme that only employs an SPM, experimental results indicate that the hybrid scheme is found to produce speed-ups of up to $1.6\times$ and $1.4\times$ respectively on two modern FPGA platforms. In addition, the caching scheme is found to consume up to $3\times$ fewer memory resources. These results indicate that it is indeed important to deal effectively with data dependent memory accesses to realise the full benefits of video compression applications. In summary, the contributions of this work are as follows.

1. A hybrid custom parallel memory sub-system, that is capable of exploiting data re-use effectively in two dimensions, in spite of data dependent memory accesses, is presented.
2. The optimum allocation of memory resources between the custom parallel cache and the SPM has been established using actual implementations on FPGA platforms.
3. The performance benefit of employing the hybrid memory sub-system is found to be greater for video sequences exhibiting greater inter-frame movements.

The organisation of this paper is as follows. First, the background and related work of memory sub-systems and the QSDPCM application are discussed in Section 2. Second, the architecture of the custom parallel cache is presented in Section 3. Third, details of the memory sub-system and datapath of the QSDPCM application are provided in Section 4. Fourth, experimental results involving the proposed memory sub-system and the QSDPCM application are given in Section 5. Finally, the paper is concluded in Section 6.

2 Background

In this section, related work regarding memory sub-systems and the QSDPCM application are discussed in Section 2.1 and Section 2.2 respectively.

2.1 Memory sub-systems

Image data requires considerable storage capacity. Therefore, the data is stored in large external memories with long access times. The external memory bottleneck has a severe impact on application performance. Fortunately, many image processing algorithms have high potential for data re-use, implying that the number of external memory accesses may be significantly reduced by buffering regularly re-used data in fast on-chip memories. Several solutions have been proposed in an effort to exploit data re-use of image processing applications optimally. Caches are commonly used on processor-based platforms. On the other hand, Scratchpad memories (SPM) are used on FPGA platforms to buffer data.

Caches are well studied memory sub-systems [16] that are widely employed in a variety of contexts, such as general processor and digital signal processor software platforms. Caches use a general scheme to decide what data to retain, based on the assumption that memory accesses have high spatial and temporal localities. Consequently, caches are flexible and are guaranteed to operate correctly for any memory access pattern of the current application. On the other hand, using the same scheme, under varying statistical conditions of the memory access pattern, results in the sub-optimal exploitation of data re-use. Several types of caching schemes have been devised to optimise data placement and reduce cache misses [16, 21]. Further, another method of speeding up cache accesses is by increasing data parallelism. Indeed, data access is parallelised through the use of interleaved cache banks [26] or cache duplication [13]. These techniques [13, 26] are targeted at processor architectures so the parameters of the memory sub-system cannot be altered to adjust to the statistical features of the memory access pattern. Conversely, in [8, 12, 19, 25, 29], dynamically reconfigurable caches have been implemented on FPGAs to support processor-based platforms, with the objective of dynamically adapting the cache parameters to different phases of an application.

Well established techniques [15, 22] have been developed to optimise the memory accesses to SPMs within loop nests, which are typical in image processing applications. The objective of this approach is to exploit data re-use between successive executions of a given loop nest, by buffering the required data using on-chip memory before the first execution of the loop nest. Given that a significant amount of overlap occurs between required data in different iterations, the number of external memory accesses is minimised using this approach. Naturally, the data that needs to be buffered, and subsequently the required buffer size, is dependent on how the image data is accessed within the loop nest. The process of obtaining the minimum buffer size, and the corresponding access functions for the image data within a loop nest, is demonstrated to be non-trivial in [22] where accesses are assumed to be affine functions of the loop iterators. Significant performance benefits and buffer size reductions are achieved as a result of this technique. The ROCCC com-

piler [15] is another tool which optimises memory accesses within loop nests. The compiler addresses the problem of data mapping to different memory types on the FPGA. Specifically, data at the point of use is stored in registers while other data is stored in block RAMs. Such an arrangement is beneficial to performance because all registered data may be accessed in parallel while only two data items may be retrieved from each block RAM.

Our work differs from those described above in the following ways. Firstly, whereas the adaptive caches and data parallel caches described above are designed in a processor-based context with a fixed amount of parallelism, the objective of our work is to develop a memory sub-system for FPGA-based applications with flexible computational parallelism. Secondly, unlike SPMs, where the memory access pattern needs to be known beforehand, the memory sub-system proposed in this paper is able to handle data dependent memory accesses, where the exact memory access pattern is unknown at design time. Thirdly, the optimal allocation of the memory resources between the SPM and the cache is established for memory access patterns with different statistical properties; these results are obtained through actual implementation on an FPGA platform.

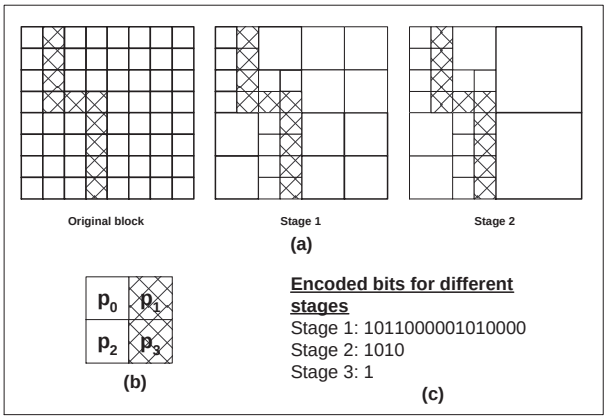


Fig. 1 Quad-tree structured pulse code modulation algorithm.

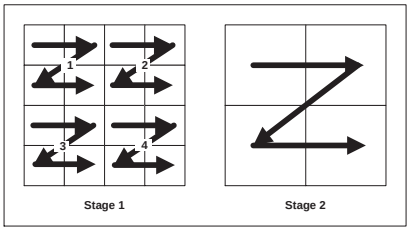


Fig. 2 QSDPCM memory access pattern.

2.2 The QSDPCM application

The QSDPCM algorithm is a direct video encoding technique, which achieves greater compression of motion compensated prediction (MCP) errors compared to the traditional Discrete Cosine tranform (DCT) technique [27]. Specifically, MCP error frames typically contain many regions of high contrast that tend to occur at the object boundaries. In a significant number of cases, applying the DCT transform to the MCP error blocks results in the spread of energy in the MCP errors [27]. Alternatively, better compression can be achieved by the QSDPCM algorithm, which represents sub-block regions, of varying sizes, with local sample means. The following steps [27] describe the process of building quad-trees for a block of MCP errors, where the block of error pixels is obtained by computing the absolute differences between the reference block and a block of pixels, displaced by motion vector (*mx*,*my*) from the original block, in the adjacent frame.

1. Figure 1(a) shows the original block of *n* by *n* pixels and the results of various stages involved in merging the block of pixels, where $n = 8$ in this particular example. A sub-block of two by two pixels, shown in Figure 1(b), is first retrieved from the top left hand corner of the original block, where p_0, p_1, p_2, p_3 and p_a indicate the intensities of the four pixels and the average intensity of the four pixels respectively. The binary value of the quad-tree leaf is 0, and the sub-block is merged when $|p_i - p_a| < \epsilon$ for all i, where ϵ is a pre-determined threshold value. Otherwise, the quad-tree leaf has value 1 and the sub-block is not merged. In this example, the sub-block is not merged because p_1 and p_3 have substantially different absolute values from p_a.
2. Further sub-blocks are retrieved in a hierarchichal zig-zag manner shown in stage one of Figure 2. Each sub-block is processed in the way described in step one. The result and bit sequence, of the merging process are shown in stage one of Figure 1(a) and Figure 1(c) respectively.
3. To compute the bit sequence for the next level in the quad-tree, the merging process is repeated in a manner similar to step one, for a sub-block size of four by four pixels. However, instead of a single value from the original block, p_i represents the accumulated sum of the intensity values of four immediately neighbouring pixels. The sub-block access pattern is illustrated in stage two of Figure 2. The results are shown in stage two of Figure 1(a) and (c).
4. The previous step is repeated with sub-block areas that increase four fold per iteration and terminates when the size of the sub-block and the original block size are equal. After the merging process is completed, all the quad-tree nodes are obtained, as seen in Figure 1(c).
5. Eight other quad-trees are then computed for surrounding image blocks at a distance of ± 1 pixel from the original block using the above steps.

Techniques have been developed to optimise the memory accesses of the QSDPCM application in a variety of processor-based platforms [11, 20, 23]. Many techniques make use of high-level code transformations to improve the locality of memory accesses and subsequently reduce data trans-

fers between main memory and the cache or SPM. For instance, power and performance gains have been realised by applying loop transformations and scalar replacement techniques on a custom hardware processor platform with a two-level memory hierarchy in [23]. Further task and data-level parallelism are investigated in [11] where the application is mapped to a platform with 13 processors and memory blocks of different sizes. In the context of a tall, multi-layered memory hierarchy, the impact of different combinations of these loop transformation techniques have been studied [20] where the trade-offs between performance, power consumption and area for different combinations are presented.

For a system involving both hardware and software components, these loop transformations may be carried out before hardware and software partitioning [10] to reduce the amount of data transfer between components. Hardware and software partitioning is also carried out for the QSDPCM application [14], where performance gains are achieved by assigning kernels with dense instruction counts to FPGA hardware. However, these gains are reduced by the presence of communication overheads between hardware and software components. Further, no explicit techniques are employed to optimise data storage and transfers in the application.

In the custom hardware context, SPMs may be used to exploit data re-use in the application. In [17], the memory accesses of the application are simplified by ignoring inter-frame movements. On the other hand, ignoring inter-frame movement will lead to error blocks with greater magnitude, and therefore lower achievable compression ratios. Since the memory accesses and computations of the QSDPCM algorithm are embedded in a multi-level loop nest, data re-use is exploited in [6] by loading the SPM at loop levels where loop iterators of the sub-loop nest do not impact the predictability of the memory accesses. However, this approach imposes limits on the exploitation of data re-use. On the other hand, by combining a SPM and the proposed custom parallel cache, this work allows data re-use opportunities that exist at higher loop levels, in the QSDPCM application, to be captured.

3 Hybrid memory sub-system

The principal contribution of this work is a hybrid memory sub-system that effectively captures data re-use despite data dependent memory accesses, particularly for custom hardware applications. When the memory accesses of the target application are data dependent, it becomes difficult to accurately identify data with high re-use frequencies. Examples are shown in [6], where references to a large memory array within a loop nest is a function of other array references that are unknown at design time. In these circumstances, sub-optimal usage of memory resources may result if scratchpad memories (SPM) are used, indicating the need for a versatile memory sub-system. The general architecture of the hybrid memory sub-system and its arbitration circuitry will be described in Section 3.1 and Section 3.2 respectively.

3.1 Architecture

The architecture of the hybrid memory sub-system is shown in Figure 3. Data re-use and memory access parallelism are exploited using N seperate sub-caches and SPMs. The control logic regulates the flow of data between the external memory and the memory sub-system. Specifically, there are two states in the operation of the memory sub-system. In the first state, the SPMs are filled with data that have high re-use frequency. Therefore, communication exists solely between the external memory and SPMs. In the second state, the datapath retrieves data from the memory sub-system for computation. If the required data items are not in the SPMs, they are retrieved from the sub-caches. If the required data item is absent from the sub-cache, a sub-cache miss occurs and the data is retrieved from external memory. Because multiple sub-cache misses can occur in parallel and misses can occur for any sub-cache, in any order, a versatile arbitration mechanism is required.

Data re-use and memory access parallelism are exploited by storing data in the sub-caches and SPMs. Each sub-cache is a direct-mapped cache that contains memory resources for data and tag storage, as well as circuitry to carry out tag comparisons. When multiple sub-cache misses occur, an arbitration mechanism is used to sequentialise accesses to external memory. The arbitration mechanism will be detailed in Section 3.2. Two-dimensional spatial locality of image data is exploited by organising the external memory address in the manner shown in Figure 4. The address is split into the row and column parts of the required location in two-dimensional image data. Each part is further sub-divided into three components *i.e. Port0* (*Port1*) , *Line0* (*Line1*) and *Tag0* (*Tag1*), which identify lower half (upper half) of the sub-cache, line and tag of the required data. *Port0* and *Port1* may then be concatenated to obtain the target sub-cache. Similarly, the two components, *Port0* and *Port1*, are used to determine which SPM the data item is stored and the components, *SLine0* and *SLine1* are used to determine where the data maps to within the SPM. The memory access parallelism can be increased by using a larger number of sub-caches, which implies allocating a larger number of bits to the components, *Port0* and *Port1*. Further, the total number of bits allocated to *Port0*, *Port1*, *Line0* and *Line1* can be varied to adjust the cache size, with reference to the acceptable cache hit rate and the available on-chip memory resources. Similarly, in the SPM address scheme, the number of bits allocated to *Port0*, *Port1*, *SLine0*, and *SLine1*, can be varied to adjust the memory access parallelism and the SPM size.

Since data accesses are frequently block-based in video applications, there will be no contentious accesses to a given sub-cache within the same block under this address mapping scheme, assuming that the number of sub-caches is the same as the number of items within the target block. Consequently, there is no need for an arbitration mechanism between the datapath and the sub-caches. On the other hand, arbitration is required between the sub-caches when multiple sub-cache misses occur in parallel. The architecture of

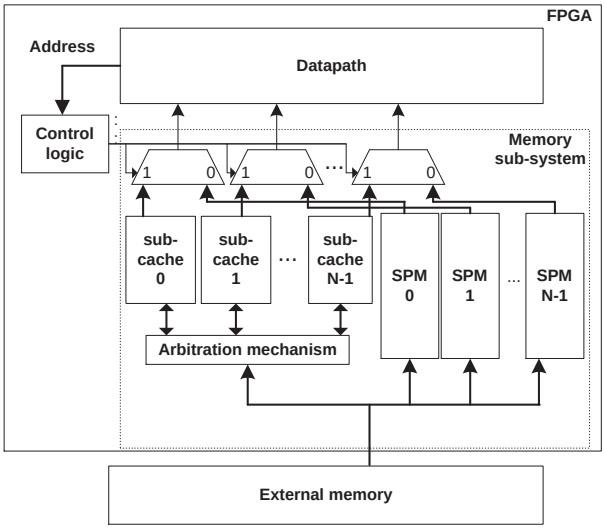


Fig. 3 Cache architecture.

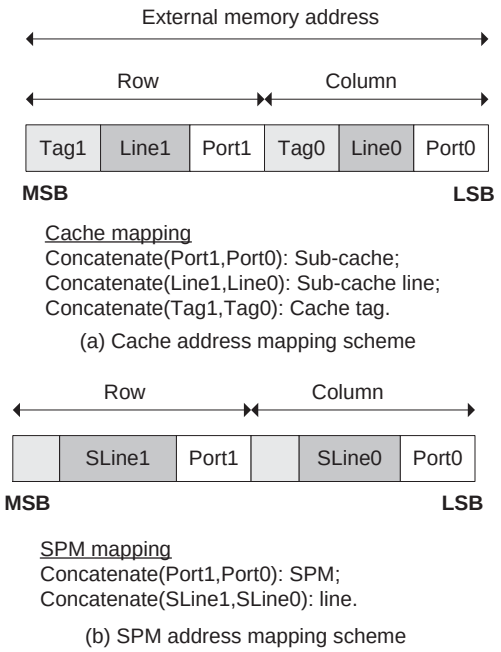


Fig. 4 Memory sub-system addressing schemes.

the arbitration mechanism is explained in greater detail in Section 3.2.

3.2 Arbitration scheme

The arbitration mechanism of the cache is designed to mediate between multiple requests for data from the external memory by the sub-caches. In the arbitration scheme, each sub-cache is assigned a unique cycle to access external memory in the event of a sub-cache miss, in a round robin manner.

The control path of the arbitration circuitry is constructed using a token based system, that is first proposed in [24]. For each group of *N* accesses, the arbitration process is initiated only when a token is received. When all of the *N* accesses have been completed, the token is passed on to the operations that are downstream of the arbitration process. The circuitry involved in the arbitration is shown in Figure 5. Specifically, a control block, C_i, is assigned to each sub-cache i where $0 \leq i \leq N-1$, as shown in Figure 5(a). When the arbitration process is initiated, the $START$ signal is asserted by a pulse, that lasts for one clock cycle. If sub-cache i experiences a hit or when $MISS_i = 0$, there is no need for external memory access because the data already exists in the sub-cache. The circuit in the shaded area of Figure 5(a) will ensure that the $FINISH_i$ signal is asserted in the same clock cycle as the $START_i$ signal. On the other hand, if sub-cache i experiences a miss, $i + 3$ delays are inserted to ensure that the start of the external memory access, associated with the sub-cache, is different from other sub-caches. Three cycles are required for the external memory access to complete and for the sub-cache data banks to be updated. These delays are injected using a cascaded chain of $i + 4$ registers as seen in Figure 5(a). External memory access begins when signal S_i is asserted and the signal $FINISH_i$ is asserted after $i+3$ cycles. The timing diagram of the arbitration process is shown in Figure 5(b) in a particular instance when $i = 4$.

The control units for individual sub-caches, C_i, are combined as shown in Figure 5(c). All the control units are initiated in parallel by the $START$ signal. Subsequently, the external memory access operations for the individual sub-caches are triggered in the appropriate cycle by the tokens S_0, ... and S_{N-1} . The circuitry in the grey area is responsible for ensuring that all the external memory access and cache updating operations complete before operations downstream of the arbitration process begin. Specifically, the signal, $FINISH$, is asserted only when all $FINISH_i$ signals are asserted, using the shaded *AND* gate with *N* inputs. However, the $FINISH_i$ signal remains asserted for only one cycle. Therefore, the assertion of $FINISH_i$ is retained using a register shown in the dotted box in Figure 5(c). Specifically, the output of the register, for sub-cache i, is asserted in the next cycle if the access operations required for sub-cache i have completed but the entire arbitration has not completed in the current cycle. The delay and area growth of the circuit are $O(\log N)$ and $O(N^2)$ respectively. The round robin arbitration scheme is used for the following reasons.

1. Using this arbitration mechanism allows each miss event to be processed independently by the individual control block, C_i. Therefore, it is likely to have a shorter critical path than the arbitration scheme proposed in [28], where additional layers of logic are required for the implementation of a complicated handshake multiplexer.
2. Different arbitration schemes have different impacts on cache miss penalties when cache hit and miss events oc-

cur concurrently within a group access. However, our experimental results indicate that the hit events in the address traces are highly correlated due to data re-use and compulsory misses respectively, indicating that the concurrent occurence of hit and miss events is rare. This observation suggests that the application performance will not suffer significantly due to a potentially larger cache penalties that are incurred, but can benefit from a shorter critical path of the proposed arbitration scheme.

4 QSDPCM case study

To investigate the effectiveness of the proposed hybrid memory sub-system, a case study is carried out with the QSDPCM application. Firstly, the impact of data dependence on the memory accesses of the application is demonstrated in Section 4.1. Secondly, we show how the cache may be combined with a SPM to exploit data re-use in Section 4.2. Finally, an description of the data path is provided in Section 4.3.

4.1 Analysis of memory access pattern

The algorithm description in Section 2.1 indicates that the memory accesses of the QSDPCM application are data dependent. Specifically, the memory accesses between different groups of quad-tree computations cannot be statically determined because these accesses are dependent on motion vectors, that are unique for each image sequence. Data dependence is shown in Figure 6(a), which illustrates the kernel used in quad-tree computation. In the example, it is assumed that the array *Image*, which contains image data, is stored in external memory. Aside from the loop indices *i, j, k, l, m* and *n*, the indices of array *Image* are also dependent on motion vectors *mx[i]* and *my[j]*.

To analyse the impact of input data dependence on data re-use, an investigation is carried out on two real image sequences. The first image sequence, 'hkmovement', depicts people walking on a crowded and busy street. Between adjacent frames, there are movements throughout the frame but the absolute range of the motion vectors are limited in range. The motion vector distribution, in the vertical dimension, is shown in Figure 7(a). As expected, the distribution of absolute motion vector lengths (vertical dimension) is found to be substantially biased towards small values. Conversely, the second image sequence, 'parachute', shows a sky diving event involving a man with his parachute. Again, there are several movements due to rapid changes in the landscape below the man, and the absolute range of the motion vectors (vertical dimension) are substantial. In this case, the distribution is similarly biased towards smaller motion vector lengths. However, the distribution is spread out to a larger extent, as seen in Figure 7(b). These observations indicate that there are significant activities in both image sequences but the movements are more pronounced in 'parachute' relative to 'hkmovement'.

```
FOR i= ... ...
 FOR j= ... ...
  FOR k= ... ...
   FOR l= ... ...
    FOR m= ... ...
     FOR n= ... ...
      block[m][n] =  Image[i+k+m+mx[i]][j+l+n+my[j]];
     END  *n loop
    END  *m loop
    build_quadtree(block);
  END  *l loop
 END *k loop
END *j loop
END *i loop
```

(a) Original code

```
FOR i= ... ...
 spm[0..2*MX+2][0..framew] = Image[i-MX-1..i+MX+1][0..framew-1];
 FOR j= ... ...
  FOR k= ... ...
   FOR l= ... ...
    FOR m= ... ...
     FOR n= ... ...
      block[m][n] = spm[k+m+mx[i]+MX][j+l+n+my[j]];
     END *n loop
    END *m loop
    build_quadtree(block);
  END *l loop
 END *k loop
END *j loop
END *i loop
```

(b) SPM optimised code

```
FOR i= ... ...
 spm[0..2*SMX+2][0..framew] = Image[i-SMX-1..i+SMX+1][0..framew-1];
 FOR j= ... ...
  FOR k= ... ...
   FOR l= ... ...
    FOR m= ... ...
     FOR n= ... ...
      IF withinrange(k+m+mx[i]+mx[i]))
       block[m][n] = spm[k+m+mx[i]+MX][j+l+n+my[j]];
      ELSE
       block[m][n] = cache(i+k+m+mx[i],j+l+n+my[j]);
      END
     END *n loop
    END *m loop
    build_quadtree(block);
  END *l loop
 END *k loop
END *i loop
```

(c) SPM and cache optimised code.

Terminologies:

frameh: Number of rows in the image frame.
framew: Number of columns in image frame.
subblockh: Number of rows in sub-block.
subblockw: Number of columns in sub-block.
mx[]: Array storing vertical dimension of motion vector.
my[]: Array storing horizontal dimension of motion vector.
MX: Maximum element in mx[].
SMX: SMX < MX or SMX = MX.

Fig. 6 Code optimisation.

Further, a simulation of the QSDPCM with sequentialised memory access is carried out to analyse the data re-use of the algorithm, in a manner that is independent of the platform and particularly the memory sub-system. From the simulation, a trace of the memory addresses in all the image frames is generated. Subsequently, the average frequency of access of each address in a frame is computed, as shown in Figure 8. The horizontal axis shows the two dimensional address in the frame, which has been linearised to one dimensional, and the vertical axis indicates the average access fre-

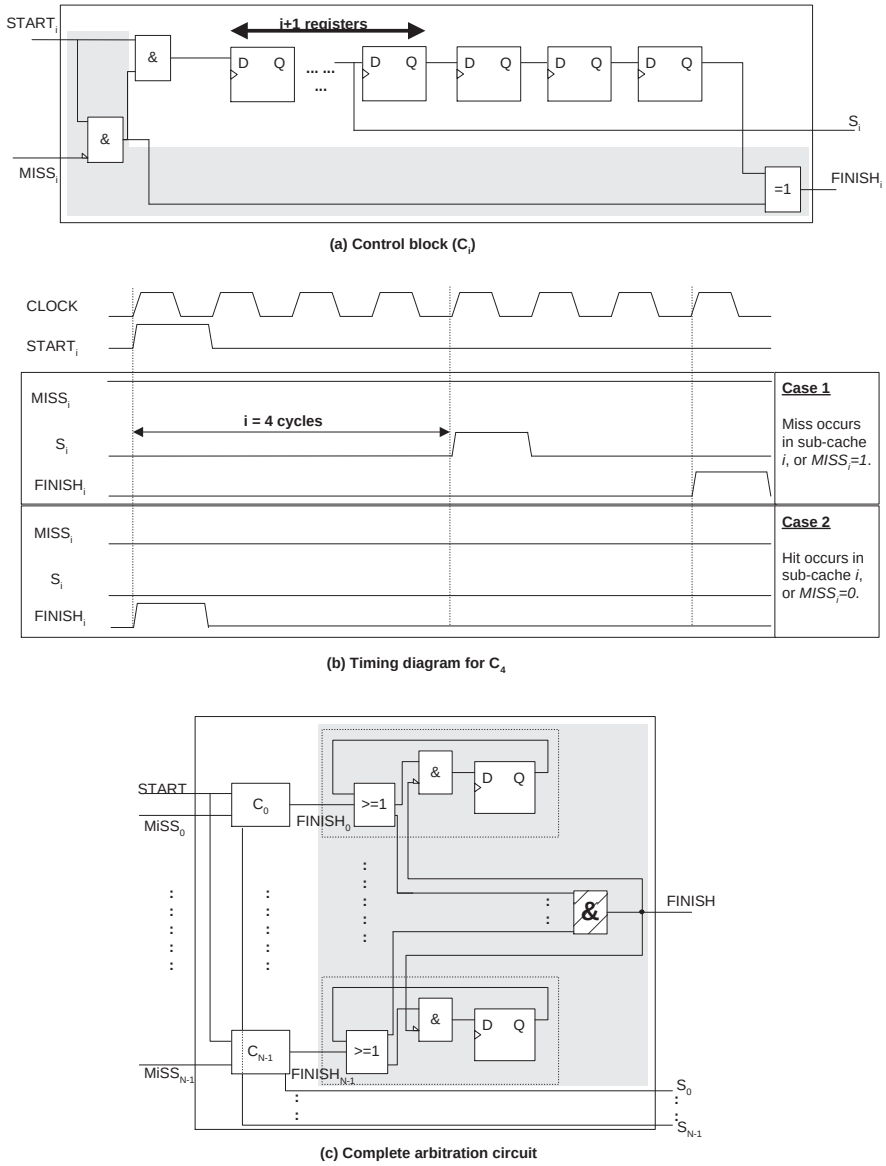
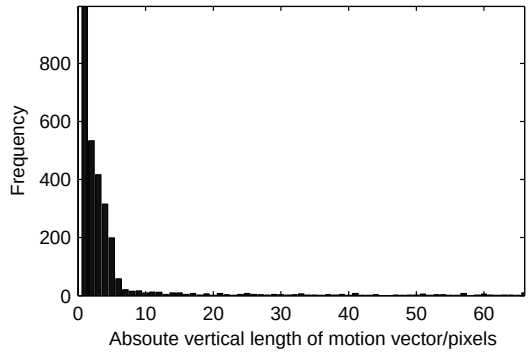


Fig. 5 Cache arbitration mechanism.

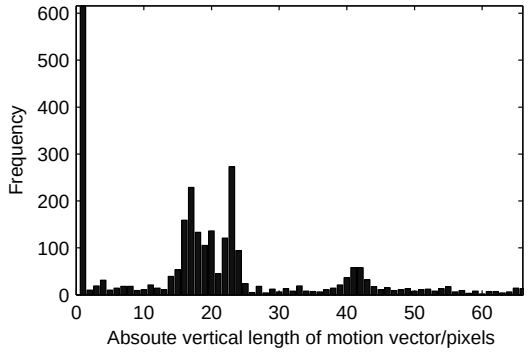
quency for each address. For both image sequences, wide spreads of data re-use frequencies are observed and the re-use profiles for the two different image sequences are significantly different. This indicates that whether a given datum is worth buffering is dependent on image features. Consequently, there is a need to consider the variability introduced by data dependent memory accesses to exploit data re-use effectively.

4.2 Memory sub-system

The need for a memory sub-system to exploit data re-use in a versatile manner is established in Section 4.1. Referring back to Figure 6, the number of pixel accesses, from external memory, may be reduced by storing data in a smaller local memory [6], for use by operations in the innermost loop-nest; this is illustrated in Figure 6(b). The computations of nine quadtrees take place within the loop nests with loop iterators *k, l, m, n*. By initiating SPM transfers before the loop nests described by *k, l, m, n*, data re-use for the computation of these quadtrees may be exploited. On the other hand, initiating SPM transfers before loop nests *j, k, l, m, n* will allow data re-use across multiple groups of quad-tree computations to be exploited as well. Specifically, entire rows of pixels are buffered using on-chip memory before computations in the inner-most loop take place. Since the indices of *Image* are a function of both the loop iterators as well as *mx* and *my*, the SPM size is dependent on the maximum absolute value of *mx[i]*.

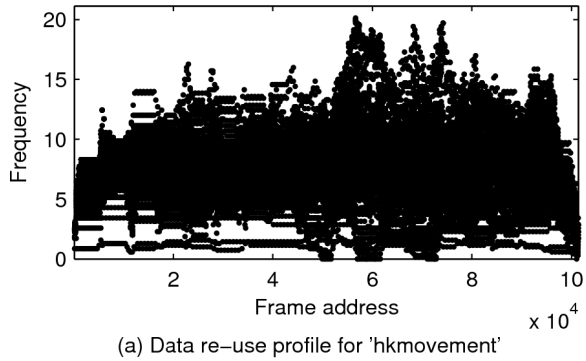


(a) Motion vector distribution for 'hkmovement'

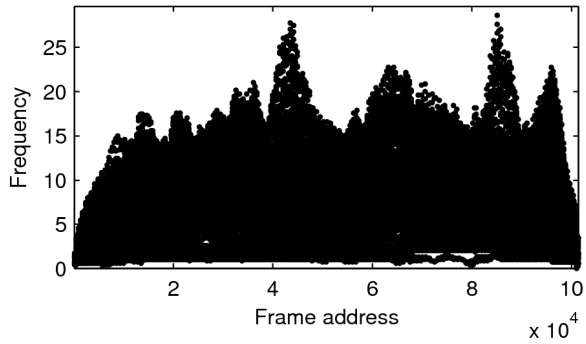


(b) Motion vector distribution for 'parachute'

Fig. 7 Distrbution of absolute vertical motion vector length for image sequences (a) 'hkmovement' and (b) 'parachute'.



(a) Data re–use profile for 'hkmovement'



(b) Data re–use profile for 'parachute'

Fig. 8 QSDPCM data re-use profile.

Figure 7 indicates that the distributions of motion vector lengths are biased towards smaller values, suggesting that it is worthwhile to buffer the rows of pixels close to the reference block. Consequently, an implementation that permits direct external memory access when the required data does not lie within the SPM could result in lower run time compared to an implementation with fully sequential memory accesses and an implementation where inner-loop memory accesses are strictly limited to SPM. However, due to the wide spread of re-use frequencies shown in Figure 8, further data re-use could be exploited by using the custom parallel cache.

According to step one of the algorithm description in Section 2.2, the image frame is accessed in blocks of four pixels. These four pixels may be accessed in parallel from four individual sub-caches. In this case, sub-cache mapping is determined by the parities, or the even and odd combinations, of the row and column addresses respectively, according to the cache addresing scheme in Figure 4. This observation may also be applied to the SPM. Indeed, instead of organising the pixels contiguously in a single group of on-chip memories, a given pixel is mapped to one of four groups of on-chip memories based on the parities on the two dimensional address of that pixel. In effect, an entire block of four pixels may be retrieved from the SPM in a single cycle.

4.3 Datapath

In this section, we describe how the function *build_quadtree*, in Figure 6, is parallelised to exploit the increased memory bandwidth through the use of the memory sub-system described in Section 4.2. A loop nest, LN_i, is required in the computation of each loop nest level, where $0 \leq i \leq 3$. In loop nest LN_0, the local mean of each group of four pixels is obtained to determine if they can be merged. Further, the means are stored in buffer X_0, for computations in subsequent loop nests. In loop nest, LN_i where $1 \leq i \leq 3$, similar computations are used to determine the merging of larger blocks, and the local means of the partial sums are stored in buffer X_i. The loop nest computations may be executed in parallel to accelerate the application. Critically, dependencies between different loop nests must be correctly handled. Specifically, each iteration of the loop nest requires four items of data in the buffer before computation commences. Therefore, four iterations of loop nest LN_i need to be completed before computations take place in loop nest LN_{i+1}.

Figure 9 illustrates how parallel loop nest computations are achieved using multiple Quad-tree processing element, labelled *QEs*. Specifically, Figure 9(a) shows the circuitry involved in the computation of a quad-tree leaf, at a given quad tree level. An on-chip buffer stores four partial sums from the *QE* to the left. When the buffer is full, a fully

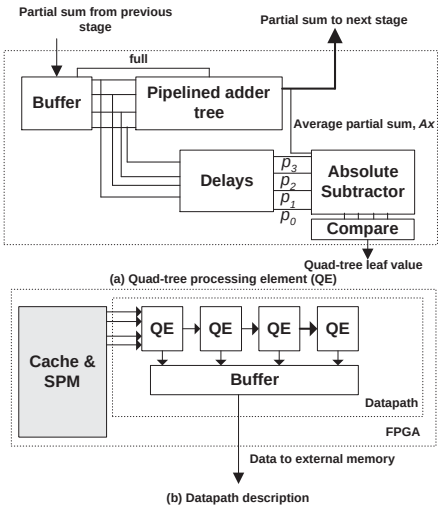


Fig. 9 Quad-tree structure generation circuit; *Absolute Subtractor* sub-block implements $|p_i - Ax|, 0 \leq i \leq 3$.

pipelined, tree-based adder is used to compute the current partial sum.

The absolute difference between the partial sums, p_i, and the average pixel value, Ax, are carried out by the *Absolute Substractor* sub-block before comparisons take place with the pre-determined threshold, T. The averaging operation is simplified by dropping the two least significant bits of the current partial sum to obtain Ax. Finally, the sub-block, *Compare* generates the binary value of the quad-tree leaf. The *Delay* block is essential to ensure that the buffer contents are properly aligned with the current partial sum generated from the pipelined adder. Using these *QEs*, quad-tree computations may be parallelised. Further, the capacity of intermediate buffers, for partial sum storage, may be reduced to four pixels. The *QE*, that is nearest to the memory sub-system, requires a bandwidth of four pixels per cycle.

The datapath is shown in the white coloured blocks of Figure 9(b). Four *QEs* are required in the quad-tree leaf computation for each level of the quad-tree structure. The *QE*, nearest to the memory sub-system, requires a bandwidth of four pixels every cycle. Therefore, four sub-caches and SPMs are required. Equivalently, one bit is allocated to each component, *Port0*, and *Port1*, in the address schemes described in Figure 4. Subsequently, the partial sums are progressively computed at each stage and passed on to the *QE* at the next stage towards the right of the figure. The quad-tree leaf values, which are computed by each *QE*, are then stored in the buffer before being transmitted to the external memory. Buffer storage is employed so that the storage process can be pipelined with computations.

5 Results

The benefits of the hybrid memory sub-system are quantified empirically, using results obtained from actual implementations, and presented in this section. The observations are summarised as follows.

1. Performance gains and memory savings of up to 1.7 times and 3.2 times are achieved when the QSDPCM application is implemented with the hybrid memory sub-system on modern FPGA platforms.
2. The hybrid memory sub-system is more effective in exploiting data re-use for image sequences with greater inter-frame movements. It is therefore important to consider image features when determining the type and size of the memory sub-system.

Consequently, employing the proposed hybrid memory sub-system leads to significant increases in the achievable frame rate, and can therefore be of critical importance to the application domain. More generally, the proposed technique and the subsequent experimental results help to provide insight, regarding the technique and the extent, of the the trade off between FPGA resource usage and performance. In Section 5.1, the experimental setup is discussed following which we present application run time and resource usage results in Sections 5.2 and 5.3 respectively.

5.1 Experimental setup

In the experiment, four image sequences with different degrees of inter-frame movements are used. They are called ‘hkmovement’, ‘matrix’, ‘starwars’ and ‘parachute’, in the order of increasing inter-frame movements. To obtain motion vectors of comparable quality for QSDPCM compression, the motion vectors for the image sequences are computed by exhaustively searching the entire frame for the best match. Subsequently, the search window size required to capture 75% of the accurate motion vectors are determined. Motion estimation is then repeated using these window sizes to obtain the required motion vectors.

Two modern FPGA platforms are chosen for the implementation of the QSDPCM application. They are the Celoxica RC250 [2] and the RC300 [3] boards, that contain a Stratix 2 [1] FPGA and a Xilinx Virtex 2 [5] FPGA respectively. The Stratix FPGA contains embedded block RAMs with a variety of memory capacities while the embedded block RAMs of the Virtex FPGA are of uniform size. Specifically, the Stratix FPGA contains 488 M512 RAMs, 408 M4K RAMs and 4 MRAMs. The number after prefix ‘M’ indicates the storage capacity of the block RAMs in terms of the number of bits; the MRAM contains about 0.5 MB of memory. The Virtex FPGA contains 144 block RAMs of homogeneous size, of 16K bits each.

In addition, the video frames are compliant with the standard CIF video format [7], which are 288 by 352 pixels in size. These frames are stored in large external SSRAM memories. A single access to the external SRAM requires two cycles, that involves setting up the address in the first cycle and the actual access in the second cycle; these two operations may be easily pipelined in the presence of multiple

Table 1 Table of abbreviations for different memory sub-systems.

Abbrieviations	Explanations
Spm	SPM size accomodates largest absolute motion vector.
Spm_d	SPM with direct external memory access when data is absent.
Hybrid_1	Hybrid SPM and cache (512 pixels)
Hybrid_2	Hybrid SPM and cache (1024 pixels)

memory accesses, such that the effective memory bandwidth is one pixel per cycle. On the other hand, each access to the embedded block memory requires only one cycle and multiple data items may be accessed in parallel by storing them in different block RAMs. The language used in both the description of the application and the memory sub-system is Handel-C [4].

Performance and resource usage comparisons are made between different memory sub-systems, by adjusting the parameters of the memory sub-system statically, at compile time. The abbrieviations and explanations for them are shown in Table 1. In the *Spm* scheme, an SPM which accomodates the largest possible motion vector is used, such that data accesses are strictly limited to the SPM. Two hybrid schemes, with caches which can store up to 512 and 1024 pixels, are chosen for this experiment. Since the number of sub-caches is fixed at four for both schemes, the cache capacity is changed by varying the number of cache lines, or the number of bits allocated to the components, *Line0* and *Line1*, in the address scheme descibed in Figure 4(a). Further, in both schemes, the same number of bits are allocated to the components, *Line0* and *Line1*. In this work, the run-time adaptation of the memory sub-system to the statistical properties of the input data stream is not carried out.

5.2 Performance

The relative performance of different memory sub-systems is shown in this section. The application run-time of the *SPM_d* scheme is expected to fall initially, as the SPM size increases, because pixels with high re-use frequencies are stored in the SPM. On the other hand, as SPM size continues to grow, an increasing number of pixels which are not accessed are stored in the SPM, resulting in an increase of application run-time. Table 2 verifies that the cycle count variation is consistent with the reasoning above. Indeed, the minimum cycle count of 852×10^4 cycles occurs at a SPM height of 60 pixels.

However, the results in Table 2 also indicate that variation in clock period is not convex. This is due to the random nature of the synthesis and place/route tools. On the other hand, a convex curve may be fitted into the resultant variation of application run-time with SPM size in Figure 10 for the *SPM_d* scheme. Incorporating the system clock period has an effect of shifting the minimum point of the curve towards a smaller SPM size. Predictably, the optimum SPM size is smaller for the image sequence 'hkmovement' compared with 'parachute', because of more abrupt movements in image sequence 'parachute'. Specifically, the minimum points occur where the SPM heights are 52 pixels and 24 pixels in Figure 10(a) and Figure 10(b) respectively. In general, the system clock period is observed to degrade with increasing SPM size. This is likely to be the result of having to make use of a larger multiplexer to combine more block RAMs together to form a larger buffer.

Table 2 Cycle count and maximum clock frequency for *SPM_d* and *Hybrid_1* (image sequence 'parachute')

SPM height/ pixels	*SPM_d*		*Hybrid_1*	
	Cycle count/ $\times 10^4$	Clock frequency/ Mhz	Cycle count/ $\times 10^4$	Clock frequency/ Mhz
16	1057	56.19	610	56.72
18	1045	51.93	611	55.39
20	1042	50.57	617	54.26
24	1031	55.81	627	54.54
26	1025	54.13	632	50.90
28	1016	46.90	637	42.37
30	1005	46.64	641	55.73
32	992	46.36	644	47.14
40	936	49.71	658	49.11
48	882	47.67	671	49.53
56	856	47.44	688	46.69
60	852	48.37	698	50.85
64	857	44.36	711	45.43
68	864	48.65	724	44.97

The cycle count and system clock frequency data for the *Hybrid_1* scheme are also shown in Table 2. In this scheme, the point where minimum cycle count occurs occurs at a SPM height of 16 pixels. The minimum execution time occurs at the same SPM height of 16 pixels in Figure 10(b). In effect, the minimum execution time of *Hybrid_1* is lower than *SPM_d* by 1.64 times. Using the *Hybrid_2* scheme, which has a larger cache, results in an execution time which is 1.7 times lower. For the *Hybrid_1* scheme, the cache hit rate is already saturated, at about 85%. Therefore, the performance improvement is modest even though the cache capacity has been doubled. For the image sequence 'hkmovement', the performance improvements are about 1.07 times and 1.04 times for the *Hybrid_1* and *Hybrid_2* schemes are observed. Given that the motion vectors in 'hkmovement' are shorter on average compared to 'parachute', a larger proportion of the accesses are directed at the SPM rather than the cache, resulting in marginal improvements. These results show that the optimum SPM height is substantially smaller than the length of the longest motion vector of the image sequence, implying that the *SPM* scheme is sub-optimal compared to *SPM_d* and the hybrid schemes.

Table 3 summarises the results which are obtained for different image sequences and FPGAs. The image sequences

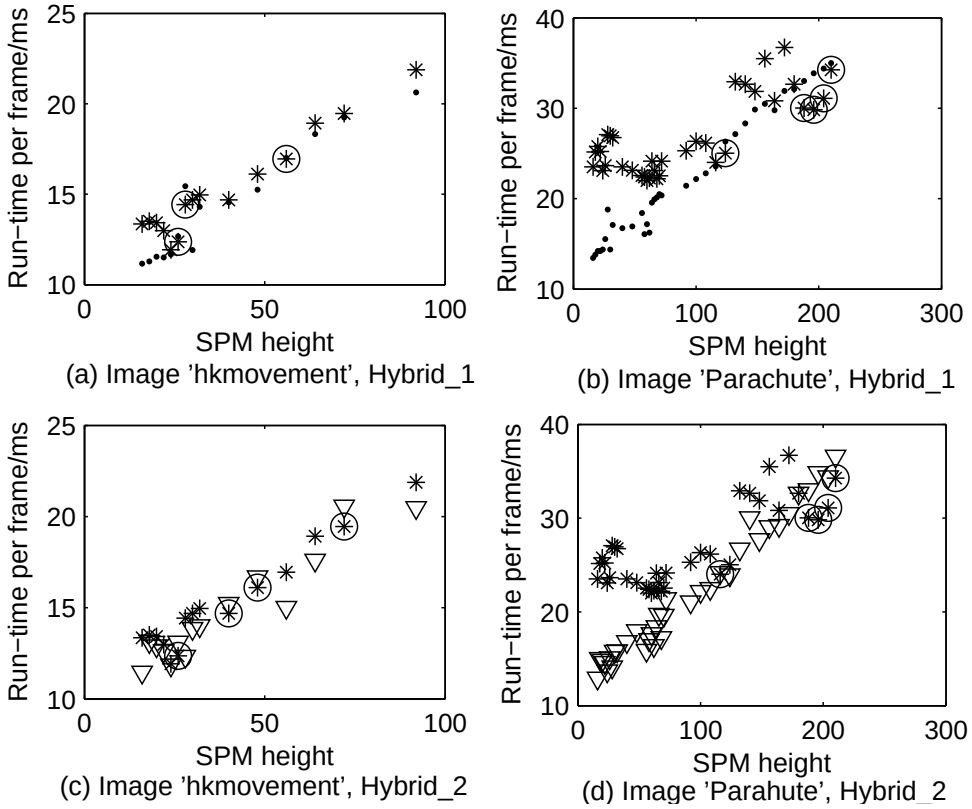


Fig. 10 Application run time results; *Spm_d*:'$*$', *Hybrid_1*:'.', *Hybrid_2*:'∇'. Data points where *Spm_d* outperforms the hybrid schemes are circled.

are arranged in the order of increasing inter-frame movements; this is reflected in the column *Window size*. The application acceleration from using the hybrid schemes is listed under the columns *Hybrid_1* and *Hybrid_2*, with respect to the lowest run time achievable by *SPM_d*. The hybrid schemes always perform better than *SPM_d* in terms of cycle count. However, the effective speed-up for 'hkmovement', using the Xilinx Virtex 2 FPGA, is less than one. This is because of more severe clock period degradation when the cache is instantiated, for that image sequence. Significantly, the gains from using the hybrid schemes increase with the amount of inter-frame movements within the image sequence, suggesting that the proposed memory sub-system is best suited for abrupt video sequences.

5.3 Resource usage

In this section, resource usage comparisons between different memory sub-systems, for image sequences 'hkmovement' and 'parachute', are made for the Altera Stratix device. The variation of the register count with SPM size does not vary significantly for both hybrid schemes and *SPM_d*. On average, there is less than 10% difference in the register counts between these two schemes. On the other hand, it is seen that the number of M4Ks used by *Spm* is much greater than all the other schemes as seen in Figure 11(a), due to the large absolute value of the longest motion vector, implying that this scheme is sub-optimal in terms of resource usage. The *Spm_d* scheme requires the least amount of resources, due to the larger number of M512 RAMs employed by both hybrid schemes; this occurs because the memory banks in the cache are implemented using M512 RAMs. Specifically, in terms of embedded memory bits, *Hybrid_1* and *Hybrid_2* have storage requirements which are higher than *Spm_d* by 1.06 and 1.11 times respectively. This observation implies that the designer faces a trade-off between performance and resource usage for the hybrid and *Spm_d* schemes when the input video features are similar to 'hkmovement'.

Figure 11(b) illustrates the resource usage profile for the video 'parachute'. The profile is similar to the profile of 'hkmovement' except that the hybrid schemes require less M4K RAMs than the *Spm_d* scheme. This difference is due to the fact that the point of minimum run time for 'parachute' is much larger, in terms of SPM height, than the point of minimum run time for 'hkmovement'. The required number of M512s for the hybrid schemes is larger than the *Spm_d* scheme, but the total storage requirements for *Hybrid_1* and *Hybrid_2*, are respectively lower than the *Spm_d* scheme by 3.18 and 3.04 times in terms of embedded memory bits. Consequently, the hybrid combinations are more efficient

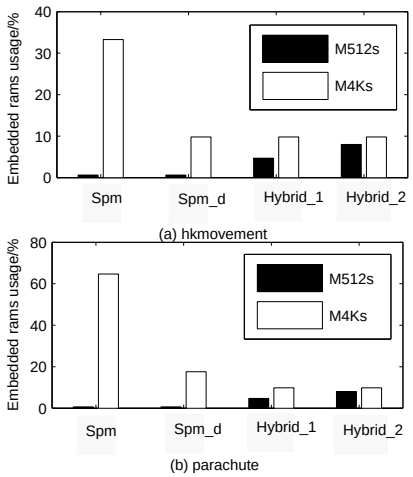


Fig. 11 Application resource usage results.

Table 3 Application acceleration for different image sequences and architectures.

Image sequence	Window size/ pixels	*Altera Stratix 2*	
		Hybrid_1	*Hybrid_2*
'hkmovement'	38	1.068x	1.041x
'matrix'	55	1.358x	1.355x
'starwars'	67	1.427x	1.437x
'parachute'	97	1.637x	1.702x
Image sequence	Window size	*Xilinx Virtex 2*	
		Hybrid_1	*Hybrid_2*
'hkmovement'	38	0.974x	0.977x
'matrix'	55	1.224x	1.275x
'starwars'	67	1.282x	1.344x
'parachute'	97	1.298x	1.387x

than the *Spm_d* scheme, in terms of both performance and resource usage, for this image sequence.

6 Conclusion

In this work, data dependence is demonstrated to have a significant impact on memory accesses, and subsequently on the data items which are worth buffering. The proposed hybrid memory sub-system will be effective in the application contexts which have significant data re-use opportunities, where the opportunities are difficult to exploit using conventional means due to input data dependence. In addition, this memory sub-system is designed for FPGAs because it is fully parameterisable and allows parallel accesses to data by making use of the distributed memory resources on FPGAs. Further, the memory sub-system exploits spatial locality of memory accesses in two dimensions, which is typically exhibited in the memory access trace of image processing applications.

The QSDPCM application is a relevant application because of significant overlaps between block accesses to image data, and because these overlaps are dependent on inter-frame movements. Indeed, experiments involving the QSDPCM application indicate that in comparison to a scheme that only makes use of a SPM, performance improvements of up to $1.7\times$ and $1.4\times$ are achieved on two modern FPGA platforms using the hybrid memory sub-system. Also, on-chip memory resource usage reductions of up to $3.2\times$ are found. Significantly, these improvements are found to be more pronounced for image sequences exhibiting greater inter-frame movements.

The results from this work indicate that there is substantial scope in the exploitation of data re-use in the presence of data dependent memory accesses. Further, the proposed memory sub-system is by no means limited to the QSDPCM application. Indeed, the memory sub-system can be used in other data dependent computation tasks like sparse matrix multiplication, which is a critical process in applications like pattern recognitions [9]. Therefore, in our future work, we intend to identify the aspects of image data accesses, which are critical to the determination of optimum memory sub-system parameters. Further, techniques will be developed to determine the design space of the memory sub-system, based on the energy consumption, resource usage and application performance.

References

1. (Undated) Altera, stratix 2 datasheet. URL `http://www.altera.com/literature/hb/stx2/stx2_sii51002.pdf`, accessed 10 July 2007
2. (Undated) Celoxica, RC250 board specifications. URL `http://www.celoxica.com/products/rc250/default.asp`, accessed 10 July 2007
3. (Undated) Celoxica, RC300 board specifications. URL `http://www.celoxica.com/techlib/files/CEL-W040216143F-257.pdf`, accessed 10 July 2007
4. (Undated) Celoxica website. URL `http://www.celoxica.com`, accessed 11 Jan 2007
5. (Undated) Xilinx, virtex 2 datasheet. URL `http://www.xilinx.com/partinfo/ds031.pdf`, accessed 10 July 2007
6. Absar M, Catthoor F (2005) Compiler-based approach for exploiting scratch-pad in presence of irregular array access. In: Proceedings of the Design, Automation and Test in Europe, pp 1162–1167
7. Aho J (Undated) A quick guide to digital video resolution. URL `http://lipas.uwasa.fi/~f76998/video/conversion/`, accessed 10 July 2007
8. Balasubramonian R, Albonesi D, Buyuktosunoglu A, Dwarkadas S (2003) A dynamically tunable memory hierarchy. IEEE Transactions Computers 52:1243–1257

9. Cohen E, Lewis D (1999) Approximating matrix multiplication for pattern recognition tasks. Journal of Algorithms 30:211–252
10. Danckaert K, Catthoor F, Man HD (1997) System level memory optimization for hardware-software co-design. In: Proceedings of the Fifth International Workshop on Hardware/Software Codesign, pp 55–59
11. Danckaert K, Masselos K, Catthoor F, Man HD (1999) Strategy for power efficient combined task and data parallelism exploration illustrated on a QSDPCM video codec. The EUROMICRO Journal of Systems Architecture 45(10):791–808
12. Dhodapkar A, Smith J (2002) Managing multi-configuration hardware via dynamic working set analysis. In: Proceedings of the International Symposium Computer Architecture, pp 233–244
13. Edmondson J, Rubinfield P, Bannon P, Benschneider B, Berstein D, Castelino R, Cooper E, Dever D, Donchin D, Fischer T, Jain A, Mehta S, Meyer J, Preston R, Rajagopalan V, Somanathan C, Taylor S, Wolrich G (1995) Internal organization of the Alpha 21164 a 300MHz 64-bit quad-issue CMOS RISC microprocessor. Digital Technical Journal 7(1):119–135
14. Galanis M, Dimitroulakos G, Kakarountas A, Goutis C (2005) Speedups from partitioning software kernels to FPGA hardware in embedded SoCs. In: Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation, pp 485–490
15. Guo Z, Buyukkurt B, Najjar W, Vissers K (2005) Optimized generation of data-paths from C codes for FPGAs. In: Proceedings of the conference on Design, Automation and Test in Europe, pp 112–118
16. Henessy JL, Patterson DA (1996) Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishing Co., Menlo Park, California
17. Jackson DJ, Ren H, Wu X, Ricks KG (2007) A hardware architecture for real-time image compression using a searchless fractal image coding method. Journal of Real-Time Image Processing 1(3):225–237
18. Jain A (1981) Image data compression: A review. In: Proceedings of the IEEE, pp 349–389
19. Kim C, Burger D, Keckler S (2002) An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In: Proceedings of the International Conference Architectural Support for Program, Languages and Operating System, pp 211–222
20. Kulkarni C, Catthoor F, Man HD (1998) Hardware cache optimization for parallel multimedia applications. In: Proceedings of the Euro-Par98 Parallel Processing, pp 923–932
21. Kulkarni C, Catthoor F, Man H (2000) Data and memory optimization techniques for embedded systems. In: Proceedings of the IPDPS Workshops on Parallel and Distributed Processing, pp 186–193
22. Liu Q, Constantinides GA, Masselos K, Cheung PYK (2007) Automatic on-chip memory minimization for data reuse. In: Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines
23. Masselos K, Catthoor F, Goutis C, DeMan H (1999) Low power mapping of video processing applications on VLIW multimedia processors. In: Proceedings of the IEEE Alessandro Volta Memorial Intnl. Wsh. on Low Power Design, pp 52–60
24. Page I, Luk W (1991) Compiling Occam into FPGAs. In: Proceedings of the Field-Programmable Logic and Applications, pp 271–283
25. Ranganathan P, Adve S, Jouppi N (2000) Reconfigurable caches and their application to media processing. In: Proceedings of the International Symposium Computer Architecture, pp 214–224
26. Sohi G, Franklin M (1991) High-bandwidth data memory systems for superscalar processors. In: Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp 53–62
27. Strobach P (1990) Tree-structured scene adaptive coder. IEEE Transactions on Communications 38(4):477–486
28. Venkataramani G, Chelcea T, Goldstein SC, Bjerregaard T (2005) Soma: a tool for synthesizing and optimizing memory accesses in asics. In: Proceedings of the 3rd IEEE international conference on Hardware/software codesign and system synthesis, pp 231–236
29. Zhang C, Vahid F, Najjar W (2003) A highly configurable cache architecture for embedded systems. In: Proceedings of the International Symposium Computer Architecture, pp 136–146