ON-CHIP CACHE ALGORITHM DESIGN FOR MULTIMEDIA SOC

Anne Pratoomtong, Yu Hen Hu

University of Wisconsin-Madison, Madison, WI 53706

pratoomt@cae.wisc.edu, hu@engr.wisc.edu

ABSTRACT

In order to optimally implement real time, high throughput, data intensive multimedia applications, it is crucial to optimize the performance of the memory sub-system to minimize excessive off-chip memory bandwidth subject to the constraint of available on-chip memory cache size. This can be accomplished by customizing algorithm transformation and designing customized cache address mapping algorithm for specific class of multimedia applications. In this paper, we propose an algorithm transformation and customized cache mapping to improve the data reusability and reduce address conflict which in turn, reduces the cache miss and memory I/O bandwidth for block-based full-search motion estimation algorithm. Simulation results using test video sequences demonstrate marked performance improvement.

1. INTRODUCTION

In modern SOC multimedia architecture, the memory sub-system often occupies more than 50% of the chip area and consumes even larger portions of power. Multimedia algorithms such as video coding, 3D rendering often require real time computing and high data throughput rate. In order to optimally implement these data-intensive multimedia applications, it is crucial to optimize the performance of the memory sub-system to minimize excessive off-chip memory bandwidth subject to the constraint of available on-chip memory cache size.

Many hardware or software techniques for enhancing the cache performance have been proposed. These include hardware techniques such as pre-fetching, value prediction, victim cache, etc.; and software techniques such as loop transformation and software pipelining. However, these existing hardware solutions are often devised for general purpose computing and do not take advantage of the deterministic data access patterns of multimedia algorithms. Existing software solutions, on the other hand, cannot address the needs to adjust the algorithm execution order so that the data memory access pattern best matches the characteristics of the underlying memory sub-system. Hence, a globally optimal solution to the data memory access problem in designing multimedia SoC must include both the development of cache-aware software and algorithm-specific hardware.

For general purpose computing, cache memory addresses are often derived from logical linear addresses through direct mapping, or various set associative mapping schemes. Comparatively, direct map cache has faster access time, less complexity, lower power consumption, and is more suitable for SOC implementation where power and area are the main constraints. On the other hand, the direct mapping scheme tends to induce more conflict misses. More importantly, these general cache memory address mapping schemes can not take advantage of the predictable memory address access pattern, and hence often result in excessive cache misses and unnecessary off-chip memory I/O. With the availability of re-configurable blocks in modern system on chip (SOC) design, it is now possible to design on-line re-configurable memory address calculation unit that can switch between different cache memory mapping schemes based on programs that are being executed [4]. In this paper, we will present a novel cache address mapping scheme that will reduce conflict cache misses for a full-search block based motion estimation (FBME) algorithm.

From the software aspect, we argue that memory access must be taken into account as a design criterion when a multimedia algorithm is mapped onto a processor array architecture. In particular, the data access pattern should be included as part of the dependence relations to facilitate algorithm mapping. The impact of the transformed algorithm on memory subsystem performance should also be part of the criteria to evaluate the quality of a particular algorithm transformation methodology. In this paper, we present the optimization of FBME algorithm base on the characteristic of the data access pattern which reduces the memory bandwidth substantially.

2. CASE STUDY: FBME ALGORITHM

FBME algorithm produces a motion vector (MV), which yields minimum mean-absolute distortion (MAD) between current block and the $(2p+1)^2$ candidate blocks

within the search area, for each of the N × N block of pixel in the current frame. Figure1 shows the high-level language representation of FBMA. It can be seen that the pixel in the current frame is used without overlapping and to complete ME for one block, each pixel in a current block is used repeatedly $(2p+1)^2$ times. In otherworld, the life span of the pixels in each block is the time it is used to complete the ME for one block. Therefore, as long as the cache is big enough to store one current block, the temporal locality of the current block is fully exploited i.e. each pixel in the current frame can be reused for each iteration. On the other hand, it is difficult to fully exploit the temporal locality of the pixel in search frame since pixel in each search block is overlapped when used. In order to complete ME of one block, $(2p+1)^2 N^2$ pixel need to be accessed but only $(N+2p)^2$ pixel exist per search area. The redundancy accessed get more severe as the block size increase and thus, waste a lot of memory bandwidth. Consider when $p = n/2$, the ratio of the pixel accessed per block and pixel exist per block is $(N+1)^2/4$.

```
do v = 0 to Nv-1
do v = 0 to Nv-1
   MV(h,v) = (0,0);
   Dmin(h,v) = ∞;
   do m = -p to p
   do n = -p to p
      MAD(m,n) = 0;
      do i = 0 to N-1
      do j = 0 to N-1
         currx = hN+j;  curry = vN+i;
         refx = hN+j+m;  refy = vN+i+n;
         MAD(m,n) = MAD(m,n) + |x(currx,curry)
                    -y(refx,refy)|;
      enddo j,i
      if Dmin(h,v) > MAD(m,n)
           Dmin(h,v) = MAD(m,n);
           MV(h,v) = (m,n);
      endif
enddo n,m,h,v
```

Figure1. Six-level nested Do-loop FBME algorithm.

3. ALGORITHM ANALYSIS AND TRANSFORMATION

Many multimedia applications contain nested do loop construct. Therefore the data memory access pattern can be evaluated statically by an equation during design time and represented as a function of loop parameters. An Access Temporal Locality Period (ATLP) can be derived form the access time equation. ATLP = 1 means the data accessed at time t is also access at time $t+1$. This will save the off-chip I/O memory bandwidth since the data is read from the cache only one time but it can be used for many iterations until a new data is requested. This in turn will also reduce the power consumption. Once the ATLP is known, the designer can optimized the algorithm in order to reduce the ATLP. From figure1, the access time equation $t(i,j,m,n,h,v)$ can be written as follows

$$t(i,j,m,n,h,v) = j + iN + (n+p)N^2 + (m+p)N^2(2p+1) + hN^2(2p+1)^2 + v\,N^2(2p+1)^2N_h$$

ATLP of the pixel in the current frame and reference frame can be found using the access time equation as follows

$$\mathrm{ATLP}_{\mathrm{Curr}} = t(i,j,m,n+1,h,v) - t(i,j,m,n,h,v) = N^2$$

$$\mathrm{ATLP1}_{\mathrm{Ref}} = t(i,j,m,n+1,h,v) - \mathrm{t}(i,j,m,n,h,v) = N^2$$

$$\mathrm{ATLP2}_{\mathrm{Ref}} = t(i,j,m+1,n,h,v) - \mathrm{t}(i,j,m,n,h,v) = N^2(2p+1)$$

$$\mathrm{ATLP3}_{\mathrm{Ref}} = t(i,j,-p,-p,h+1,v) - \mathrm{t}(i,j,p,-p,h,v) = N^2(2p+1)$$

$$\mathrm{ATLP4}_{\mathrm{Ref}} = t(i,j,-p,-p,h,v+1) - \mathrm{t}(i,j,-p,p,h,v) = N^2N_h\,(2p+1)^2 - 2pN^2$$

The current frame pixels have a fix ATLP and their temporal locality can be fully exploit when the cache size is big enough to store the entire current block. The reference frame pixels' ATLP, on the other hand, can not be completely determined even the on-chip cache stores the entire reference frame block. This is because many reference frame pixels belong to overlapped reference frame blocks and will be used during different reference frame block ME computation. Given these observations, the goal is to transform the FBME algorithm to reduce ATLP of the reference frame pixels. Unfortunately, the details of the transformed algorithm have to be omitted due to space limitation. We called the reduction of $\mathrm{ATLP1}_{\mathrm{Ref}}$ and $\mathrm{ATLP2}_{\mathrm{Ref}}$ as R1 and R2 algorithm optimization respectively. We only consider the reduction of $\mathrm{ATLP1}_{\mathrm{Ref}}$ and $\mathrm{ATLP2}_{\mathrm{Ref}}$ because $\mathrm{ATLP3}_{\mathrm{Ref}}$ is the same as $\mathrm{ATLP2}_{\mathrm{Ref}}$ and $\mathrm{ATLP4}_{\mathrm{Ref}}$ is too large and depend on N_h which makes it unpractical to optimized as it requires a lot more storage unit for partial MAD and yields more complex program.

4. DATA AND RESULTS

Table1 give the comparison of the number of reads per block and number of increasing partial MAD storage unit with respect to the number of partial MAD storage unit of the original FBME algorithm for each optimization level. Table2 shows the resolution of each video format and Table3 shows the bandwidth for each optimization level.

Optimization level	*No optimization*	*R1 optimization*	*R2 optimization*
#read/block	$(2p+1)^2N^2$	$2N^2(2p+1)$	$(2p+N)^2$
#increasing register	-	N	$(2p+1)^2$-1

Table1. Amount of I/O VS optimized algorithms

We calculate the worst case bandwidth for standard video format by assume that bus width = 1 byte. We also assume that all the data are available in memory. The bandwidth is equal to xN_vN_hf where x is the number of read per block shown in table 1 and f is frame rate.

Reducing the ATLP to 1 not only reduces the memory bandwidth required, it also reduces the address and data bus transition which in turn reduce the power consumption.

Format	Resolution			
	Pels	lines	Block size	Frame/sec
QCIF	176	144	8	29.97
CIF	352	288	16	29.97
NTSC	480	480	16	29.97
HDTV	1920	1080	32	30
HDTV	1280	720	32	60

Table2. Resolution of different video format.

Format	Bandwidth (pixel/sec)		
	No optimization	R1	R2
QCIF	6.15E+07	1.37E+07	3.04E+06
CIF	8.78E+08	1.03E+08	1.22E+07
NTSC	2.00E+09	2.35E+08	2.76E+07
HDTV	6.77E+10	4.11E+09	2.49E+08
HDTV	6.02E+10	3.65E+09	2.21E+08

Table3. Off-chip bandwidth requires.

In SOC, the main bandwidth bottleneck lies on the chip boundary. It is costly to bring the data from off chip. Therefore, we would like to minimize the off chip bandwidth by algorithm transformation as we did earlier. Figure2 shows the effect of the algorithm optimization to the off chip memory bandwidth with various on chip cache size. In order to show the effectiveness of the algorithm transformation, we again assume worst case scenario when the bus with is 1 byte.

The minimum off chip bandwidth required to perform FBME on a QCIF data format shown by the horizontal dash line in figure2 is the rate to bring one frame into the on chip cache which is equal to 176×144×29.97 byte/s. One can see that for current frame access, when the cache size is greater or equal to $ATLP_{Curr}$ which is 258 or 8 in log2 base, the minimum bandwidth is achieved. For reference frame access, fortunately, the cache size does not have to be as big as $ATLP4_{Ref}$ to achieve the minimum off chip bandwidth. From figure2, cache size equal to $2ATLP2_{Ref}$ (cache size = $ATLP2_{Ref}$ is vertical line at 12.08) can achieve the minimum off chip bandwidth.

Figure3 show the relationship between the cache size and the off chip memory bandwidth require to perform FBME on a HDTV data format with the different optimization level of the algorithm transformation. One can see, for current pixel access, that the minimum bandwidth (show by the horizontal line) can not be achieved even though the cache size is greater or equal to $ATLP_{Curr}$ which is 64 or 6 in log2 base (show by the vertical line) for this case. This indicates that the conflict miss occurred. For reference pixel access, the R2 optimization out perform R1 optimization and no optimization. However, even though the cache size that require to achieve the minimum off chip bandwidth is much smaller than $ATLP4_{Ref}$, it is much larger than the case for QCIF data format (≈ 57 $ATLP2_{Ref}$).

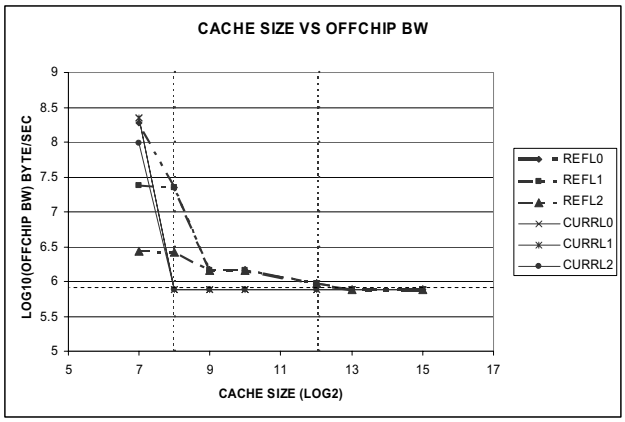

Figure2. Data format is QCIF(176×144) with block size = 16 and frame rate = 29.97 frame/s

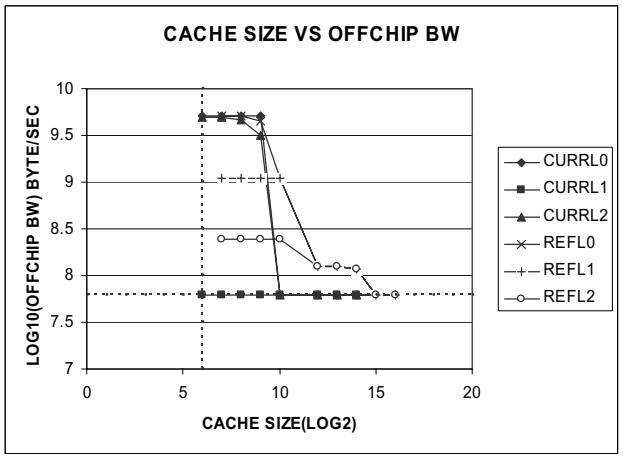

Figure3. Data format is HDTV(1920×1080) with block size = 8 and frame rate = 30 frame/s

5. MEMORY SIZE AND MEMORY MAPPING SYNTHESIS

There are 3 cache misses type, cold miss occur when the data is reference for the first time, capacity miss occur when the cache is not big enough to hold all the data, and conflict miss occur when the data is map into the same cache location and interfere with each other. There is no capacity miss if the cache is big enough to hold the whole working set. Unfortunately, it is almost impossible to have such a big cache since for multimedia application; the working set is quite large. However, most of the

multimedia algorithm accesses a data in a regular stride pattern and thus the cache only need to be as big as ATLP to reduce the capacity misses. We call this cache size as a target cache size. The conflict miss, however, does not depend on cache size. Even though the cache size is equal to the target cache size, the conflict miss can occur and mostly will be severe since it will repeat regularly as the access pattern. If the cache is at target cache size and there is a conflict miss occur, the problem lies in the mapping i.e. the mapping does not result in a uniform address distribution. Figure2 show an example for ideal case where there is no conflict miss when the cache size is at the target cache size. However, the conflict miss is sensitive to the block size and frame size and thus can occur when these parameters change. For a 2 dimension array data such as image, the data is mostly store in the off chip memory in a raster scan order and should remain that way for compatibility. However, we can modify the address mapping of the on chip cache. Let's consider a simple example of performing FBME on an image with N=4, Nv=3, Nh=3. Since the access pattern is the same for all current pixels, we can consider only one block. Since $ATLP_{Curr} = N^2$, the target cache size in this example is 16. Follow the linear mapping convention, we can see that pixel at the memory location 0-3, 12-15, 24-27, 36-39 are accessed and thus will map to the cache location 0-3, 12-15, 8-11, 4-7. There is no interference and the address is uniformly distribute in the cache and thus, with cache size = $ATLP_{Curr}$, the minimum cache miss is achieve which in turn save the off chip bandwidth. However, if the Nh is changed to 2, the address is mapped to the cache location 0-3, 8-11, 0-3, 8-11 which cause the worst case conflict miss! As a result, we purpose an adjustable address mapping scheme base on the access pattern. Again, we can analyst only one block since the access pattern is the same for every block. The access pattern for the first block (v=h=0) is iNNh+j = 8i+j, $0 \le i \le N-1$, $0 \le j \le N-1$. Therefore, in order to avoid conflict miss, $iNNh\%N^2 = iN$. As a result, Nh should set to 5. Note that this does not affect the real frame size since the conventional mapping is still used to bring the data from off chip memory. This modification of mapping work as a "virtual array padding". And as a result of this, the conflict miss is reduced. Figure 4 illustrates the efficiency of the modified mapping scheme by apply it to the data inquiries by the R2 FBME algorithm running on HDTV data format. One can see from the plot that the conflict miss of the current pixel data that occur in figure3 is now resolved. Furthermore, the bandwidth required for reference pixel data is also reduced.

6. CONCLUSION

In this paper, we shorten the ATLP of the data required by the FBME algorithm by customized algorithm transformation and modify the on chip cache mapping using the virtual array padding. By shorten the ATLP, the number of pixel read per block reduced substantially and thus reduce the memory bandwidth. Taken into account of the memory access characteristic of the FBME algorithm, the memory bandwidth can be reduce further with respect to the target cache size by the modify the address mapping to make it more uniform which reduces the conflict miss of the on chip direct map cache.

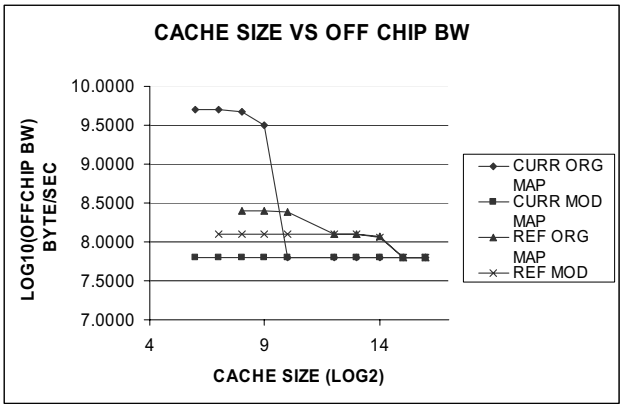

Figure4. Bandwidth require by the L2 FBME algorithm with and without modified mapping VS cache size.

We would like to extend these techniques to use with various data intensive multimedia algorithm and generate a more automate and general framework of the algorithm transformation to reduce the ATLP and cache address mapping to reduce the conflict miss.

7. REFERENCES

[1] K. Beyls, Y. Yu, E. D'Hollander, "Characterization and Optimizaation of Cache Behavior", http://winpar.elis.rug.ac.be/ppt.

[2] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, "Custom Memory Management Methodology", Kluwer Academic Publishers, 1998.

[3] S. pratoomtong, S. Kittitornkun, and Y. H. Hu, "Data flow scheduling and control for motion estimation array processor", ICEP 2004, Songkhla, Thailand.

[4] M. Miranda, F. Catthoor, M. Janssen, H. De Man, "ADOPT: Efficient Hardware Address Generation in Distributed Memory Architectures", IEEE 1996.

[5] P. R. Panda, N. Dutt, A. Nicolau, "Memory issue in embedded Systems-On-Chip", Kluwer Academic Publishers, 1999.

[6] J. Tuan, T. Chang, C. Jen, "On the Data Reuse and Memory Bandwidth Analysis for Full-Search Block-Matching VLSI Architecture", IEEE 2002.