Adaptive Prefetching for Multimedia Applications in Embedded Systems

Hassan Sbeyti*, Smail Niar*, Lieven Eeckhout**
***LAMIH, University of Valenciennes France**
****ELIS, University of Ghent, Belgium**

Abstract

Video sequence compression/decompression algorithms like MPEG4 are used in many applications because of their efficiency in supporting different compression bit rates (5Kbits/s-5Mbits/s), their enhanced error resilience/robustness, their content based interactivity and scalability which make them applicable in mobile systems. On the other hand these algorithms require a high computational processing power and memory bandwidth. The high memory bandwidth requirements do not only affect the real time behavior of such applications but also its power consumption. In this paper, we analyze and show how to exploit the memory behavior of MPEG4 applications. To improve the memory access performance, we present a new and simple prefetching mechanism, which adapts the memory access mechanism to the memory access patterns of the different application parts. By doing so, we are able to increase performance, to better utilize the available resources and to reduce power consumption. Using our prefetch method, we are able to get up to 6% IPC improvement, more than 50 % cache miss reduction, and up to 4.7 % power reduction. Our mechanism results in better performance for a 2KB data cache than is achievable with 8KB data cache (without prefetching) for StrongArm SA1110 and Xscale-like processor configurations. This mechanism requires limited hardware resources and generates little additional overheads (external bus transfers). This makes this adaptive prefetching well suited for embedded processor micro-architectures.

1 Memory Access Behaviors by MPEG4

In order to study the memory access behavior (concerning the data cache only) of the MPEG4 application running in an embedded system, we modified the sim-wattch SimpleScalar simulator [7] to profile the MPEG4 application at run time. Intel StrongArm1110 and Xscale micro-architecture models were used. The following video sequences are used: *foreman*, *news*, and *container* with two different frames size CIF (352X288) and QCIF (176X144). For each configuration, four VOPs are decoded as I-VOP, P-VOP, P-VOP and P-VOP.

We first compute the "**Inter-miss stride**" which is defined as the distance in the memory addresses between two consecutive cache misses. If a memory access to address X causes a cache miss and if the next cache miss is caused by an access to memory address Y, then the inter-miss stride is computed as S = Y–X.

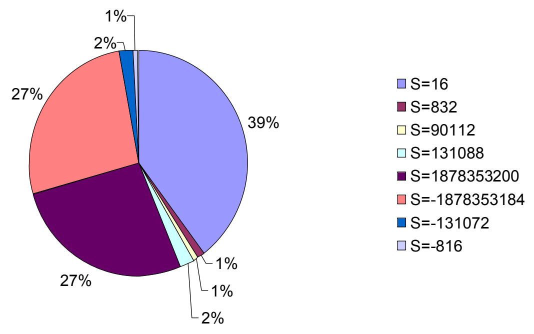


Figure 1. The top 8 "Inter-miss strides" for the container qcif video sequence with an 8KB data cache.

Figure 1 summarizes the top 8 inter-miss strides that were observed for the container qcif video sequence for a SA1110 configuration with an 8KB data cache. We also computed the **"inter-miss interval"** which corresponds to the number of clock (cycles) between two consecutive cache misses. If a cache miss occurs at T_x and if the next cache miss occurs at T_y, the inter-miss interval is $I = T_y – T_x$. In hardware, this can be calculated using a hardware counter: it is incremented each clock cycle and reset to zero on a cache misses.

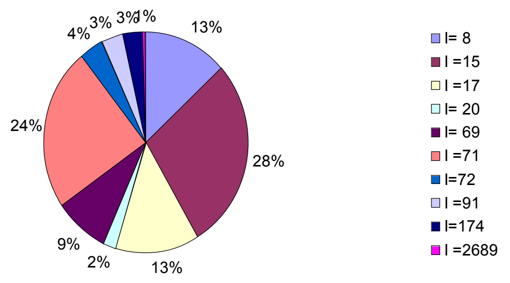


Figure 2: The top 10 "Inter-miss intervals" for the container qcif video sequence with an 8KB data cache.

Figure 2 summarizes the top 10 inter-miss intervals for the container qcif video sequence for a SA1110 configuration with an 8KB data cache

Based on the "inter-miss stride" and the "inter-miss interval" we now define two new concepts, namely the "**constant miss pattern**" and the "**alternate miss pattern**". We first define the constant miss pattern. Consider a sequence of inter-miss strides $X_1,X_2,\ldots,X_n$ and a corresponding sequence of inter-miss intervals $T_1,T_2,\ldots,T_n$. A constant miss pattern of length *n* is then defined as $(<X_0,T_0>,<X_1,T_1>,\ldots,<X_n,T_n>)$ with $X_1=X_2=\ldots=X_n$ and $T1=T_2=\ldots=T_n$.

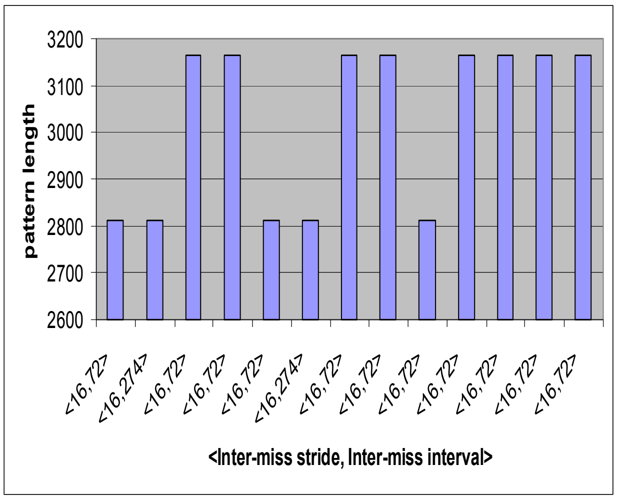


Figure 3: The top 13 "constant miss patterns" for the container qcif video sequence with an 8KB data cache.

Figure 3 summarizes the top 13 constant miss patterns as observed for the container qcif video sequence for a SA1110 configuration with an 8KB data cache. From figure 4 we observe that the constant miss pattern (<16, 72>,<16,72>) has a pattern length of about 3150.

We now define the alternate miss pattern. Consider a sequence of inter-miss strides $X_1,Y_1,X_2,Y_2,\ldots,X_n,Y_n$ and a corresponding sequence of inter-miss intervals $T_{x1},T_{y1},$ $Tx_2,Ty_2,\ldots,Tx_n, Ty_n$. An alternate miss pattern of length *n* is then defined as $(<X_0,T_{x0}>,<Y_0,T_{y0}>,<X_1,T_{x1}>,<Y_1,T_{y1}>,\ldots,<X_n,T_{xn}>,<Y_n, T_{yn}>)$ with $X_1=X_2=\ldots=X_n$, $T_{x1}=T_{x2}=\ldots=T_{xn}$, $Y_1=Y_2=\ldots=Y_n$ and $T_{y1}=T_{y2}=\ldots=T_{yn}$.

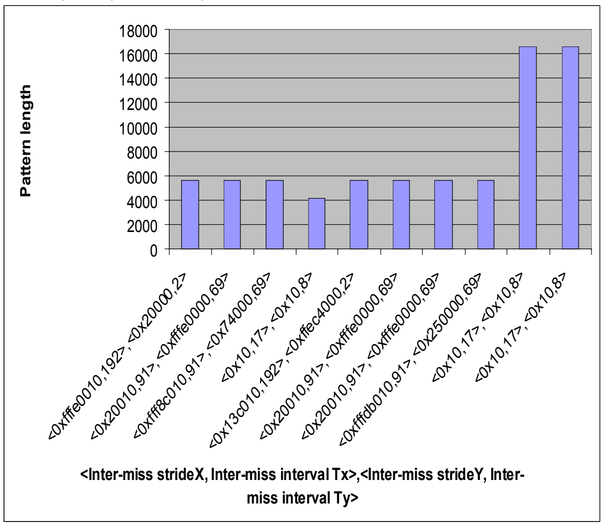


Figure 4: The top 10 "alternate miss patterns" for the container qcif video sequence with an 8KB data cache.

Figure 4 summarizes the top 10 alternate miss patterns as observed for the container qcif video sequence for a SA1110 configuration with an 8KB data cache. From figure 5 we can see for example that the alternate miss pattern (<0x10,17>,<0x10,8>,<0x10,17>,<0x10,8>) has a pattern length of about 16500. The constant miss patterns and the alternate miss patterns represent about 75% of all the cache misses. The remaining miss patterns (causing 25% of all the cache misses) are more complex and are therefore not taken into consideration in this paper.

2 Mechanism of the Method Adaptive Data Prefetching

In this paragraph we introduce a new prefetching method that is able to exploit constant and alternate miss patterns as observed in the memory access behavior in MPEG4. We will call it "adaptive prefetching" because it adapts itself at run time to the different inter-miss strides and to the different inter-miss intervals. Our prefetch mechanism is based on two hardware units: the miss pattern detector and the block loader. The miss pattern detector should detect the beginning of a constant miss pattern as well the alternate miss pattern, while block loader should prefetch ahead.

To detect the beginning of the constant miss pattern the miss pattern detector proceeds as follows: It needs to compare two recent inter-miss strides (X, Y) and their corresponding inter-miss intervals (T_x,T_y). If they are equal to each other($X=Y$ and $T_{x1=}T_{y1}$), this means that the beginning of a new constant miss pattern is detected, namely (<X, T>, <X, T>, …). We observed that 90% of all the constant miss patterns have a pattern length bigger then 3. To detect the beginning of the alternate miss pattern the miss pattern detector proceeds as follows:

It needs to compare two inter-miss strides (X, Z) and their corresponding inter-miss intervals (T_x, T_z) that occur timely as follows: $(<X,T_x>,<Y,T_y>,<Z,T_z>, <W,T_w>,\ldots)$. If $X=Z$, $T_x=T_z$, $Y=W$, and $T_y=T_w$ this means that the beginning of a new alternate miss pattern is being detected.

The block loader of the adaptive prefetching mechanism initiates a prefetch operation within C clock cycles after the occurrence of a miss. Where C is the inter-miss interval minus the main memory latency.

The address to prefetch is obtained from the memory address that caused the last miss plus the inter-miss stride and the prefetching sequence continues as long as no new cache miss has occurred.

3 Simulation Results

We used the Wattch SimpleScalar simulator [7][8] with the Intel Strong ARM SA-1110 [1] and the Xscale[2] microprocessor configurations.

As for MPEG4 decoder application (MoMuSys) we use the reference software (version. 2, ISO/IEC 14496-5:2001) [6]. For the video sequences, we used three different test sequences: *foreman*, *news*, and *container*. We worked also with different frames size CIF (352X288) and QCIF (176X144) which are suitable for embedded systems.

As we are interested with the effect of the data cache on the architecture performances, five configurations of this cache has been tested: 2KB, 4KB, 8K, 16KB, and 32KB. we demonstrated that for multimedia application like MPEG4, the access to the data cache can be optimized by the use of new prefetching methods. The question that arose is whether to choose between more data cache size and the optimization of the data cache access, by implementing our adaptive prefetching mechanism.

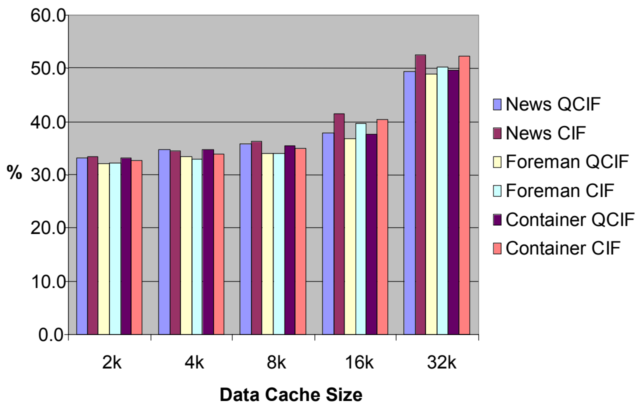


Figure 5: Data Cache miss reduction for decoding four VOP's IPPP using the adaptive prefetching mechanism.

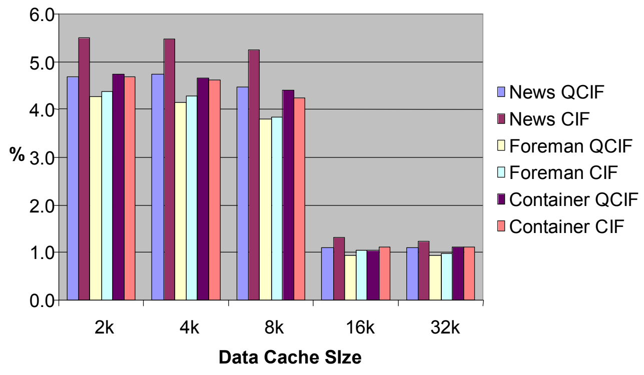
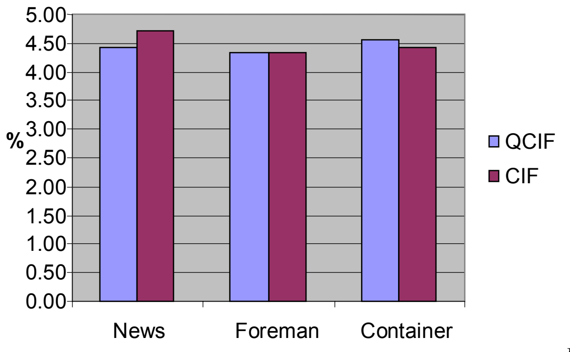


Figure 6 : IPC improvement for decoding four VOP's IPPP using the adaptive prefetching mechanism**.**



. *Figure 7: Energy reduction for 2KB Data cache with prefetching versus 8K KB Data cache without for decoding fou VOP's IPPP.*

Figure 7 shows that the energy reduction for 2KB with prefetching versus 8KB without prefetching. The energy reduction is about 4.5 % for the different video sequence and frame size. In this figure the additional external data bus accesses are not taken into consideration because of the following reason:

4 Conclusion

In this paper we have proposed a simple and new prefetching mechanism that can be used in embedded processors. We have demonstrated that the usage of such mechanism can improve the IPC and reduce the energy consumption of the cache memory significantly for multimedia applications (MEPG4) especially for small size data caches. We tested this technique among different video test sequences (news, foreman, container) with different frame sizes (QCIF and CIF), and we modeled different micro architecture, the Intel Strong ARM 1110 and the xscale. Our mechanism presents a constant improvement overall.

We showed that by the usage of our adaptive prefetching mechanism with 2KB cache size, we were able to get an IPC improvement better then using 8k without prefetching. This leads to energy reduction about 5%, by the cost of minimal additional hardware. While for embedded processors the energy consumption and the cost are dominant factors, the use of the adaptive prefetching method is then a good alternative.

References

[1] Intel Corporation. Intel Strong ARM SA-1110. Microprocessor, Developers manual, 2000.

[2] Intel corporation, "The Intel XScale Microarchitecture technical summary", ftp://download.intl.com/design/intelxscale/XSacleDatasheet4.pdf.

[3] Peter Kuhn, Algorithms, complexity analysis and VLSI architecture for MPEG-4 motion estimation, Kluwer Academic Publishers, 1999.

[4] Advanced RISC Machines Ltd (ARM) 1995, ARM 7TDMI Data Sheethttp://www.arm.com/arm/documentation?OpenDocument.

[5] International Standard ISO/IEC 14496-2, Information technology-Coding of audio-visual objects—Part2: Visual, second edition 2001-12-01.

[6] The Wattch SimpleScalar simulator http://www.ee.princeton.edu/~dbrooks/sim-wattch-1.0.tar.gz.

[7]D. Burger and T. M Austin. The SimpleScalar Tool Set, Version 2.0 Computer architecture News, Pages 13-25, June 1997.