Adaptive Prefetching for Multimedia Applications in Embedded Systems

Hassan Sbeyti, Smail Niar
LAMIH, University of Valenciennes, France
(Hassan.Sbeyti, Smail.Niar)@univ-valenciennes.fr

Lieven Eeckhout
ELIS, Ghent University, Belgium
Lieven.Eeckhout@elis.UGent.be

Abstract

This paper presents a new and simple prefetching mechanism to improve the memory performance of multimedia applications. This method adapts the memory access mechanism to the access patterns as observed in the application. By doing so, performance is increased, the available resources are better utilized and energy consumption is reduced. Using our prefetch method, we are able to get up to 5.5% IPC improvement, more than 50% cache miss reduction, and up to 4.5% energy reduction. Our mechanism results in better performance for a 2KB data cache than is achievable with an 8KB data cache (without prefetching) for StrongArm SA1110 and Xscale-like processor configurations. This mechanism requires limited hardware resources and generates little additional external bus transfers. This makes this adaptive prefetching well suited for embedded microprocessor systems

1. Memory access behavior for the MPEG4

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 their energy consumption. In this paper, we analyze and show how to exploit the memory behavior of MPEG4 applications on an embedded system. We used the SimpleScalar Tool Set [1] to profile the MPEG4 application at run time. The following video sequences are used: *foreman*, *news*, and *container* with two different frames sizes CIF (352x288) and QCIF (176x144). For each configuration, four VOPs are decoded (IPPP). We define the *inter-miss stride* as the distance in the memory addresses between two consecutive cache misses.

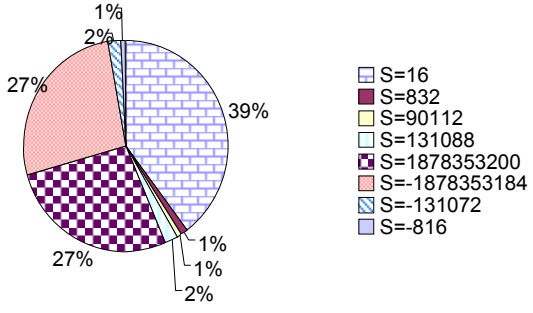

Figure 1: Top 8 inter-miss strides for the container qcif video on SA1110 configuration with an 8KB D-cache.

The top 8 inter-miss strides (in Fig.1) account for 71.26% of the total cache misses. Three inter-miss strides are responsible for 93% of the top 8 or 66.5% of all the cache misses that occur[1]. We also computed the *inter-miss interval* which corresponds to the number of clock cycles between two consecutive cache misses.

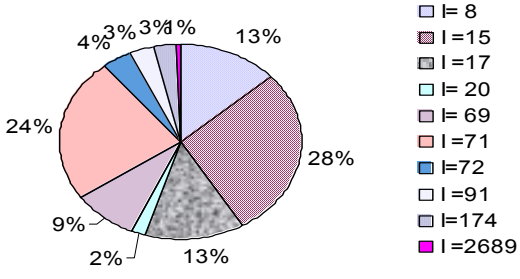

Figure 2: Top 10 inter-miss interval for the container qcif video on SA1110 configuration with an 8KB D-cache.

If a cache miss occurs at T_x and if the next cache miss occurs at T_y, the inter-miss interval is $I = T_y - T_x$.

Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research – Flanders (Belgium) (F.W.O. Vlaanderen).
Hassan Sbeyti is an active member of the Arab Open university.

[1] These large strides (1878353200 -1878353184) are due to cache misses on the heap followed by cache misses in the data segment and visa versa.

1530-1591/04 $20.00 (c) 2004 IEEE

The high occurrences of a small number of miss patterns, both strides and intervals, as depicted in Fig. 1 and Fig. 2, respectively, are due to the memory access patterns of the MPEG4 decoder. We observed similar behavior for the CIF frame size as well as for different cache sizes (from 2KB to 32KB). 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*. 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 $T_{x1},T_{y1},\ T_{x2},T_{y2},\ldots,T_{xn},\ T_{yn}$. A constant miss patterns of length *n* is defined as $(<X_1,T_{x1}>,<Y_1,T_{y1}>,\ldots,<X_n,T_{xn}>,<Y_n,T_{yn}>)$ with $X_1=Y_1=X_2=\ldots=Y_n$ and $T_{x1}=T_{y1}=T_{x2}=\ldots=T_{yn}$. An alternate miss pattern of length *n* is then defined as $(<X_1,T_{x1}>,<Y_1,T_{y1}>,\ldots,<X_n,T_{xn}>,<Y_n,T_{yn}>)$ with $X_1=X_2=\ldots=X_n$, $T_{x1}=T_{x2}=\ldots=T_{xn}$, $Y_1=Y_2=\ldots=Y_n$ and $T_{y1}=T_{y2}=\ldots=T_{yn}$.

2. Mechanism of adaptive prefetching

Our prefetch mechanism is based on two hardware units: the *miss pattern detector* and the *block loader*. The miss pattern detector detects the beginning of a constant or alternate miss pattern, whereas the block loader prefetches ahead. The miss pattern detector proceeds as follows. It needs to compare consecutive inter-miss strides and their corresponding inter-miss intervals that occur timely as follows: $(<X,T_x>,<Y,T_y>,<Z,T_z>,<W,T_w>,\ldots)$. If for example, $X=Z$, $T_x=T_z$, $Y=W$ and $T_y=T_w$, the miss pattern detector has detected an alternate miss pattern. The block loader then 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 prefetch address is obtained from the memory address that caused the last miss plus the inter-miss stride. The prefetching sequence continues as long as no new cache misses occur.

3. Simulation results

We used the Wattch SimpleScalar simulator [1][2] with the Intel Strong ARM SA-1110 [3] and the Xscale [4] microprocessor configurations. The MPEG4 decoder (MoMuSys) was taken from the reference software [5].
Figures 3, 4 and 5 depict the data cache miss reduction, the IPC improvement and the energy reduction that are obtained using our adaptive prefetching method.The cache miss rate reduction (Fig. 3) is more than 50% for 32KB D-cache sizes and 35% for 2KB to 8KB caches. The IPC improvement (Fig. 4) can be up to 5.5%. The energy consumption (Fig. 5) for a 2KB D-cache with prefetching is 4.5% smaller than the energy consumption of an 8KB cache without prefetching.

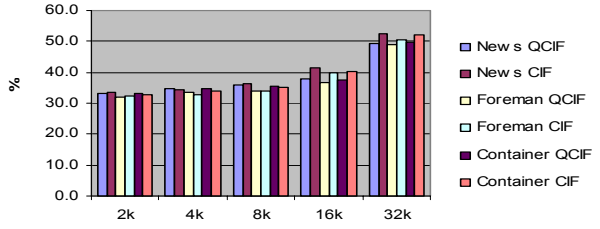

Figure 3: D-cache miss reduction through adaptive prefetching for different cache sizes.

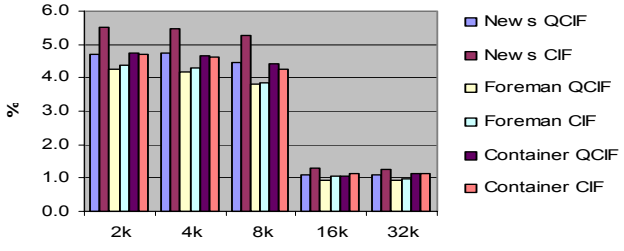

Figure 4: IPC improvement through adaptive prefetching for different cache sizes.

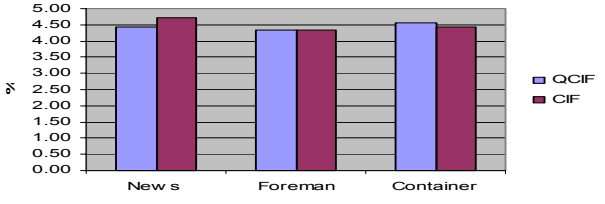

Figure 5: Energy reduction for a 2KB D-cache with prefetching versus an 8KB D-cache without prefetching.

4.Conclusion

In this paper we have proposed a simple prefetching mechanism that can be used in embedded processors. We have demonstrated that this mechanism improves performance and reduces the total energy consumption for the MPEG4 multimedia application. This mechanism is particularly useful for small data caches.

5. References

[1] D. Burger and T. M Austin. The SimpleScalar Tool Set, Version 2.0. Computer Architecture News, June 1997.
[2] D. Brooks, V. Tiwari and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, ISCA-27, 2000.
[3] Intel Corporation. Intel Strong ARM SA-1110. Microprocessor, Developers manual, 2000.
[4] Intel corporation. The Intel XScale Microarchitecture technical summary.
[5] International Standard ISO/IEC 14496-5, reference software, second edition 2001-12-15.