Pattern-driven prefetching for multimedia applications on embedded processors

Hassan Sbeyti [a], Smaïl Niar [a], Lieven Eeckhout [b],*

[a] *LAMIH-ROI, University of Valenciennes, Le Mont Houy, 59313 Valenciennes Cedex 9, France*
[b] *ELIS, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium*

Abstract

Multimedia applications in general and video processing, such as the MPEG4 Visual stream decoders, in particular are increasingly popular and important workloads for future embedded systems. Due to the high computational requirements, the need for low power, high performance embedded processors for multimedia applications is growing very fast. This paper proposes a new data prefetch mechanism called pattern-driven prefetching. PDP inspects the sequence of data cache misses and detects recurring patterns within that sequence. The patterns that are observed are based on the notions of the inter-miss stride (memory address stride between two misses) and the inter-miss interval (number of cycles between two misses). According to the patterns being detected, PDP initiates prefetch actions to anticipate future accesses and hide memory access latencies. PDP includes a simple yet effective stop criterion to avoid cache pollution and to reduce the number of additional memory accesses. The additional hardware needed for PDP is very limited making it an effective prefetch mechanism for embedded systems. In our experimental setup, we use cycle-level power/performance simulations of the MPEG4 Visual stream decoders from the MoMuSys reference software with various video streams. Our results show that PDP increases performance by as much as 45%, 24% and 10% for 2KB, 4KB and 8KB data caches, respectively, while the increase in external memory accesses remains under 0.6%. In conjunction with these performance increases, system-level (on-chip plus off-chip) energy reductions of 20%, 11.5% and 8% are obtained for 2KB, 4KB and 8KB data caches, respectively. In addition, we report significant speedups (up to 160%) for various other multimedia applications. Finally, we also show that PDP outperforms stream buffers.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Embedded processors; MPEG; Multimedia applications; Prefetching

---

* Corresponding author. Tel.: +32 92643405; fax: +32 92643594.
  E-mail addresses: hassan.sbeyti@univ-valenciennes.fr (H. Sbeyti), smail.niar@univ-valenciennes.fr (S. Niar), leeckhou@elis.ugent.be (L. Eeckhout).

1. Introduction

While there is a lot of work done on prefetching for general-purpose and scientific applications, the amount of work done on prefetching for multimedia applications, and more in particular MPEG4 Visual stream decoders is rather limited. However, due to the regular data memory access patterns of multimedia applications, prefetching techniques can be employed to significantly improve memory performance as well as overall performance.

This paper proposes a new data prefetch mechanism called *pattern-driven prefetching* (*PDP*) specifically targeted for embedded processors running MPEG4 Visual stream decoders—the design of PDP was initially motivated by the memory access pattern behavior as observed for the MPEG4 Visual stream decoder from the MoMuSys reference software; this paper however shows that other multimedia applications can also benefit from PDP. The basic idea of PDP is to inspect the sequence of data cache misses and to detect patterns in this sequence. The patterns that we envision contain information concerning the inter-miss stride (the memory address stride between two misses) and the inter-miss interval (the number of cycles between two misses). Based on these miss patterns, prefetch actions are initiated. An important design issue for prefetchers is to know when to stop prefetching ahead in order to reduce cache pollution as well as the number of additional memory accesses. In PDP, prefetching stops as soon as (i) a prefetched block was not read before the next prefetch action is initiated or (ii) another cache miss occurs.

Our experimental setup uses cycle-level simulations of the MPEG4 Visual stream decoder from the MoMuSys reference software and various MPEG4 video streams. Our results show that PDP reduces the data cache miss rate significantly over a range of cache sizes: from 13% for a 2KB cache up to 50% for a 32KB cache. In case PDP is unable to hide memory latency completely, PDP reduces the average load latency (up to 2.5× for the 2KB cache). This translates itself in significant performance increases: 45% for a 2KB D-cache, 24% for a 4KB D-cache and 10% for an 8KB D-cache. This leads to the interesting obser-

vation that a 2KB D-cache enhanced with PDP outperforms a 4KB D-cache without PDP by as much as 15%. We show that the increase in external memory accesses due to PDP is limited to less than 0.6%. The reduction in energy consumption (on-chip plus off-chip) due to PDP is significant as well: up to 20% for a 2KB D-cache, 12% for a 4KB D-cache and 6% for an 8KB D-cache. In addition, various experiments show that PDP also improves performance for other multimedia applications. We report significant speedups of up to 160%. Finally, we also compare PDP versus stream buffers and conclude that PDP yields higher speedup and requires orders of magnitude less additional external memory accesses.

This paper extends [11] in several ways: (i) we study the miss patterns in more details, (ii) we present a new stop criterion for our prefetch mechanism which significantly improves performance due to reduced cache pollution with newly prefetched blocks and which in addition minimizes the number of additional memory accesses, (iii) we evaluate PDP on a number of multimedia applications other than the MPEG4 Visual stream decoder, (iv) we now consider on-chip as well as off-chip energy consumption when presenting energy results—this gives a more faithful view on the overall system energy savings due to PDP; [11] only considered on-chip energy consumption—and (v) we compare PDP versus the stream buffer.

The remainder of this paper is organized as follows. We first discuss related work on data prefetching for multimedia applications. In Section 3, we study the memory access patterns of the MPEG4 Visual stream decoder and show that the miss patterns are regular and can thus be used to drive our prefetch mechanism called PDP which is presented in Section 4. Section 5 details our experimental setup and Section 6 presents our results. Finally, we conclude in Section 7.

2. Related work

A large body of literature is devoted to data prefetching for scientific and general-purpose applications. Various authors have proposed soft-

ware prefetching as well as hardware prefetching techniques. Software prefetching [5] relies on statically inserting prefetch instructions into the binary of an application. These prefetch instructions load the requested data from memory well in advance of the corresponding load instruction that will access the same memory location. There are a number of issues related to software prefetching, namely the selection of loads for which to generate prefetches and the scheduling of these prefetch instructions sufficiently early to hide the memory latency.

Hardware prefetching on the other hand, observes the run-time memory behavior of the application and initiates prefetch actions according to anticipated future memory accesses. Unlike software prefetching, hardware prefetching does not incur additional instruction overhead, but lacks information about future memory references that is potentially available at compile time.

In this paper, we consider hardware prefetching for embedded systems running MPEG4 Visual stream decoders. The amount of work done on prefetching for multimedia applications, and more in particular MPEG-like applications is rather limited.

Zucker et al. [14] studied hardware and software prefetching for MPEG applications. They compared three hardware prefetch approaches (the stream buffer [7,9], the stride prediction table [6] and the stream cache [6]) versus a newly proposed software prefetching approach. In their evaluation and in contrast to this work, they only used the fraction of eliminated misses as a performance metric; we also consider overall performance measured in the number of instructions executed per cycle (IPC). Another important difference between this paper and the work done by Zucker et al. [14] is the fact that they assumed an infinite memory bandwidth and did not consider energy consumption; we consider the amount of additional memory accesses as well as the overall energy consumption as important design issues for our prefetch mechanism.

McKee et al. [8] investigated the performance bottlenecks of MPEG4 applications on general-purpose microprocessors without single-instruction-multiple-data (SIMD) support. Based on experiments done on real MIPS hardware, they conclude that MPEG4 is mainly computation bound (not memory bound) and thus will not benefit from memory system optimizations. Their study was done on general-purpose microprocessors having fairly large caches, namely 32KB. In this paper however, we show that on resource-limited embedded systems, MPEG4 Visual stream decoders can benefit significantly from data prefetching, more in particular PDP.

A well known problem with prefetching is cache pollution, i.e., prefetch data replacing other data in the cache that will be used in the future by the processor. Cache pollution is especially a problem for small caches since it seriously degrades overall performance. This problem can be addressed by adding a hardware structure to hold prefetch data as is the case for stream buffers. In some circumstances, adding additional hardware is not always a viable solution and therefore Reungsang et al. [10] propose the *fixed prefetch block* approach which limits the number of prefetch blocks in the cache to one single block per set in the cache. In this paper, we address the cache pollution problem using our stop criterion which stops prefetching when the previously prefetched block remains untouched when the next prefetch is to be initiated.

In [13], Tang et al. compare fetch size adaptation versus stream buffers for media applications. In fetch size adaptation, the cache line size is adjusted according to the spatial behavior of the application execution. Large line sizes are used when good spatial locality is detected and small line sizes are used when poor spatial locality is detected. Different line sizes can co-exist at the same time. Their comparison shows that fetch size adaptation performs equally well as stream buffers.

3. Memory access patterns in MPEG applications

As stated in Section 1, our prefetch mechanism uses data cache miss patterns to drive the prefetching. In order to gain a better understanding on those cache miss patterns, we first introduce a number of concepts. We define the *inter-miss stride* 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 $S = Y - X$. We also define the *inter-miss interval* as the number of clock cycles between two consecutive cache misses. If a cache miss occurs at time T_X and if the next cache miss occurs at time T_Y, the inter-miss interval is $I = T_Y - T_X$. This can be implemented using a hardware counter: it is incremented each clock cycle and is reset to zero on a cache miss. Note that the inter-miss interval does not include the latency for servicing the cache miss. For example, if a cache miss occurs at t_{10} and the next cache miss occurs at t_{65} with a memory access latency of 30 cycles, then the inter-miss interval equals 25 cycles.

Based on the inter-miss stride and the inter-miss interval we now define the concept of the *m-order miss pattern*. Consider a sequence of inter-miss strides $S_0, S_1, \ldots, S_{n-1}$ and a corresponding sequence of inter-miss intervals $I_0, I_1, \ldots, I_{n-1}$. An *m-order miss pattern* of length n is then defined as

$$(\langle S_0, I_0\rangle \ \langle S_1, I_1\rangle \cdots \langle S_{m-1}, I_{m-1}\rangle)$$
$$(\langle S_m, I_m\rangle \ \langle S_{m+1}, I_{m+1}\rangle \cdots \langle S_{2\cdot m-1}, I_{2\cdot m-1}\rangle) \cdots$$
$$(\langle S_{n\cdot m}, I_{n\cdot m}\rangle$$
$$\langle S_{n\cdot m+1}, I_{n\cdot m+1}\rangle \cdots \langle S_{(n+1)m-1}, I_{(n+1)m-1}\rangle)$$

with the following properties:

- $S_0 = S_m = \cdots = S_{n\cdot m}$ and $I_0 = I_m = \cdots = I_{n\cdot m}$
- $S_1 = S_{m+1} = \cdots = S_{n\cdot m+1}$ and $I_1 = I_{m+1} = \cdots = I_{n\cdot m+1}$
- $\cdots$
- $S_{m-1} = S_{2\cdot m-1} = \cdots = S_{(n+1)m-1}$ and $I_{m-1} = I_{2\cdot m+1} = \cdots = I_{(n+1)m-1}$.

For instance, a sequence of second-order miss patterns of length n is defined as

$$(\langle S_0, I_0\rangle \ \langle S_1, I_1\rangle)$$
$$(\langle S_2, I_2\rangle \ \langle S_3, I_3\rangle) \cdots$$
$$(\langle S_{2\cdot n}, I_{2\cdot n}\rangle \ \langle S_{2\cdot n+1}, I_{2\cdot n+1}\rangle)$$

with $S_0 = S_2 = \cdots = S_{2\cdot n}$, $S_1 = S_3 = \cdots = S_{2\cdot n+1}$, $I_0 = I_2 = \cdots = I_{2\cdot n}$, and $I_1 = I_3 = \cdots = I_{2\cdot n+1}$. A concrete example of a second-order pattern of length 3 could be the following inter-miss stride
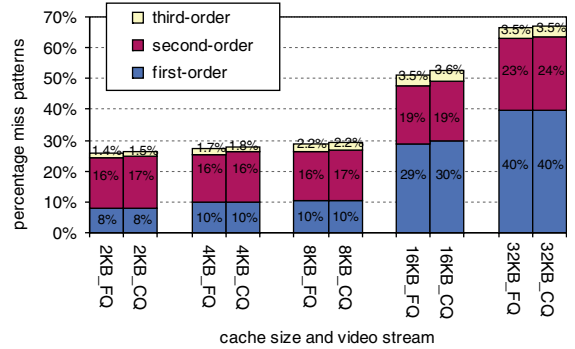

Fig. 1. The percentage of first-, second- and third-order miss patterns relative to the total number of cache misses for QCIF foreman (FQ) and container (CQ) video streams. This is shown for different cache sizes.

sequence $8, 4, 8, 4, 8, 4$ and the corresponding inter-miss interval sequence $40, 50, 40, 50, 40, 50$.

Note that in the above definition of an *m-order* miss pattern, we require that the inter-miss intervals be equal. In practice though, approximate equality of the inter-miss intervals is sufficient.

Fig. 1 shows the miss patterns that are observed for the MPEG4 Visual stream decoder using the foreman and container QCIF video streams (we refer to Section 4 for a detailed description of the experimental setup). The results for the other video streams are very similar. This graph shows the percentage of first-, second- and third-order miss patterns. For example, for the 16KB cache, 30% of all cache misses are first-order miss patterns, 19% are second-order miss patterns, and 3% are third-order miss patterns. Figs. 2–4 show the pattern length distribution for the first-, second- and third-order miss patterns, respectively, for the foreman QCIF video stream. For example, for the 16KB D-cache, see Fig. 2, we observe that 55% of the first-order miss patterns have a pattern length between 1000 and 2000, and 20% have a pattern length between 100 and 500. For the second-order miss patterns, the pattern length seems to be fairly large. The third-order miss patterns on the other hand, have significantly shorter pattern lengths (see Fig. 4). We conclude from this section that the cache miss patterns that are observed are simple, repetitive and have a fairly long period. As such, we can exploit this notion to drive the prefetch mechanism.
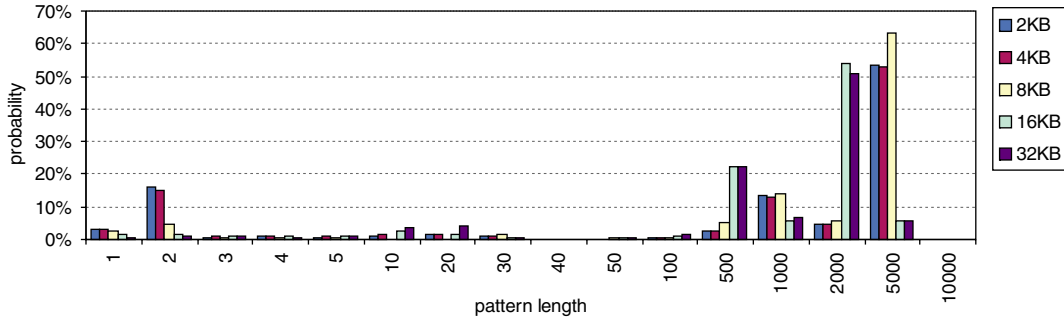
Fig. 2. Pattern length distribution for the first-order miss patterns for different cache sizes.
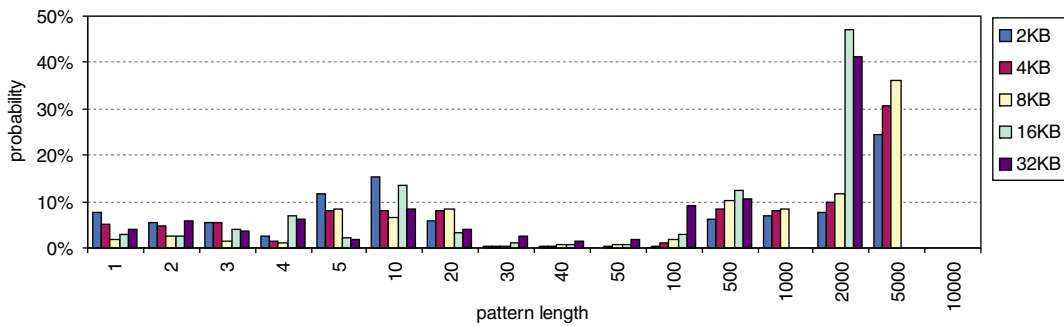

Fig. 3. Pattern length distribution for the second-order miss patterns for different cache sizes.
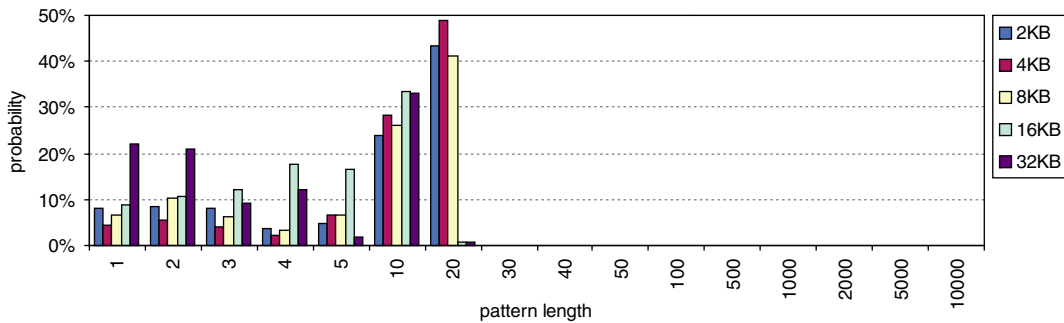

Fig. 4. Pattern length distribution for the third-order miss patterns for different cache sizes.

4. PDP: Pattern-driven prefetching

PDP exploits the *m*-order miss patterns as observed in the data memory access behavior of the MPEG4 Visual stream decoder. PDP adapts itself dynamically 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* detects the beginning of an *m*-order miss pattern. Once a miss pattern is detected, the *block loader* prefetches ahead.

To detect the beginning of the first-order miss pattern, the miss pattern detector compares two

consecutive inter-miss strides S_0 and S_1, and their corresponding inter-miss intervals I_0 and I_1. If they are equal to each other, i.e., $S_0 = S_1$ and $I_0 = I_1$ (recall that approximate equality of inter-miss intervals is sufficient), this indicates the beginning of a new first-order miss pattern, namely ($\langle S, I \rangle \langle S, I \rangle \cdots$). Detecting the beginning of a higher-order miss pattern is similar. In our evaluation setup we only consider first- and second-order miss patterns; we do not consider third-order miss patterns because of their low occurrence (less than 3.5%, see Fig. 1).

The block loader initiates a prefetch operation within c clock cycles after the occurrence of a miss, with c the inter-miss interval minus the main memory latency. The address to prefetch from is obtained from the memory address that caused the last miss plus the inter-miss stride. Note that under given circumstances main memory latency can be bigger than the inter-miss interval. In that case, the prefetch action is initiated immediately after the current cache miss. By doing so, fetching the cache block is anticipated which reduces the average load latency and which in its turn improves performance significantly.

There are two conditions under which PDP stops prefetching. First, when a new cache miss occurs; this suggests that the current miss does not correspond well to the previously detected miss pattern. Second, when the line that is prefetched is not accessed before the next prefetch action is initiated. This stop condition involves one single flip-flop that is set on each prefetch and reset on each access to the most recently prefetched cache line. If this flip-flop is still set when a new prefetch action is initiated, prefetching is stopped. This second stop criterion eliminates prefetching ahead when the miss pattern does no longer occur. Note that PDP for first-order and second-order miss patterns requires very little additional hardware:

- two adders to calculate the inter-miss strides and intervals,
- four comparators to detect first-order and second-order miss patterns,
- six registers to hold the latest cache miss addresses, inter-miss strides and inter-miss intervals,

- two adders to calculate the next prefetch address and the cycle to initiate the next prefetch action.

5. Experimental setup

In this paper, we use Sim-Panalyzer[1] which is a power/performance simulator for the ARM ISA based on the SimpleScalar simulator [4]. The processor simulation model closely resembles the Intel XScale microprocessor [1] (see Table 1). For the data cache, we considered five different configurations ranging from 2KB to 32KB in order to quantify the sensitivity of our prefetch mechanism to the cache configuration (see Table 2). Further, we assume a non-blocking cache which means no multiple outstanding cache misses are supported—this is a viable assumption for many contemporary embedded processor systems. Sim-Panalyzer computes performance as well as energy consumption from cycle-level simulations. Since Sim-Panalyzer only calculates on-chip energy consumption, we extended Sim-Panalyzer to also incorporate energy consumption due to accesses to (off-chip) main memory as well. This is done by adding 4.95 nJ per access to main memory. As such, for a line size L, the additional energy consumption for a cache miss is 4.95 nJ $\cdot L$. This memory energy consumption model is taken from [12].

For the MPEG4 Visual stream decoder, we use the MoMuSys reference software (version 2, ISO/IEC 14496-5:2001) [2,3]. We use three different video sequences as input from three different levels of complexity: *foreman*, *news*, and *container*. The frame resolutions are CIF (352×288) and QCIF (176×144). These small frame sizes are typical for embedded systems given the small displays of portable devices. For each of those videos and for each of those resolutions, we decode four VOPs: I, P, P and P. We only decode four frames because the variability between different frames from the same type is relatively small. In other words, we can drastically reduce the total simula-

---

Table 1
Micro-architecture configuration simulation model

IFQ, RUU and LSQ sizes	8
Branch predictor	128-entry bimodal and 128-entry direct-mapped BTB
Processor width	1 decode, 2 issue and 2 commit
L1 instruction cache	32KB 32-way set-associative with 32 byte blocks
Memory access latency	32
Memory access bus width	8
Functional units	1 integer ALU, 1 integer multiply, 1 data memory port, 1 floating-point ALU and 1 floating-point multiply
Execution mode	in-order

Table 2
Data cache configurations

Data cache size	# of sets	Line size	Associativity
2KB	128	16	1
4KB	256	16	1
8KB	256	16	2
16KB	256	32	2
32KB	256	32	4

tion time without sacrificing accuracy by only simulating four VOPs.

6. Results

In this section, we evaluate PDP. We first show by how much PDP removes cache miss patterns, and by how much PDP reduces the data cache miss rate and average load latency. Second, we quantify the performance speedup gained from PDP. Third, we measure the amount of additional external memory accesses due to PDP. Fourth, we show that PDP reduces system-level energy consumption. Fifth, we evaluate PDP for multimedia applications other than the MPEG4 Visual stream decoder. Finally, we compare PDP versus stream buffer prefetching.

6.1. Data cache miss rate

Fig. 5 shows miss pattern percentages that are eliminated by applying PDP. We observe that

PDP is capable of removing more than 60% of all first- and second-order miss patterns. The remaining 40% miss patterns that are still present even after applying PDP, are due to the following reasons. First, miss patterns of unit length cannot be removed by our prefetching mechanism; PDP has to detect the miss patterns before it can initiate prefetches. Second, by applying PDP new miss patterns may be introduced. This is due to the fact that PDP removes cache misses thereby introducing new cache miss patterns which remain undetected by PDP. Fig. 6 shows the pattern lengths of the first-order miss patterns that remain after applying PDP. We observe that the pattern length of the remaining first-order miss patterns is very short, typically smaller than 10–20. For the second-order miss patterns, we observed that PDP is capable of eliminating all miss patterns with a length longer than 1.

Fig. 7 shows the reduction in data cache miss rate (in percent point) due to applying PDP. Fig. 8 shows the same data relative to the data cache miss rate (in percentage). As expected, the data cache miss rate decreases for larger caches; for a 2KB cache, the cache miss rate is around 6% whereas for the 32KB cache the miss rate varies between 0.5% and 1%. The number of cache misses eliminated through PDP also decreases for larger cache sizes (Fig. 7). However, relative to the cache miss rate, the percentage of eliminated cache misses increases (see Fig. 8). This is to be expected given the large number of miss patterns for larger cache sizes (see Fig. 1). We observe that for larger cache sizes, 16KB and 32KB, PDP removes around 33% and 45% of the data cache misses, respectively. For smaller cache sizes, 2KB to 8KB caches, the reduction in data cache miss rate varies from 12% to 17%.

6.2. Average load latency

In the previous section, we quantified the reduction in data cache miss rate through PDP. However, in practice it may be the case that PDP does not completely hide the memory latency on a cache miss, however, PDP can hide a substantial portion of the memory latency. Consider for example the case where the memory latency M is
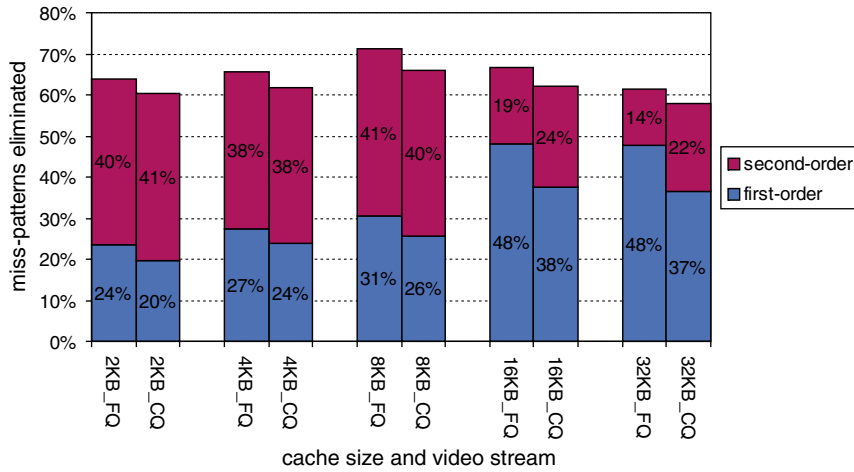
Fig. 5. The percentage eliminated first- and second-order miss patterns by PDP for different data cache sizes and the foreman and container QCIF video streams.
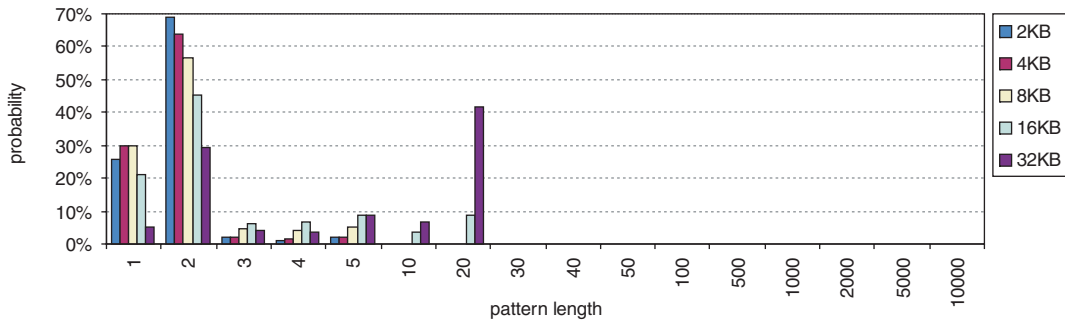

Fig. 6. The distribution of the pattern length of the remaining first-order miss patterns after applying PDP for different data cache sizes and the foreman QCIF video stream.
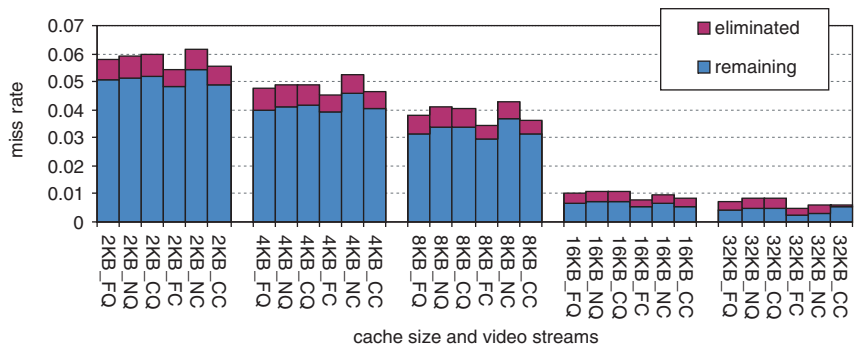

Fig. 7. The miss rate reduction in percent point through PDP for different cache sizes and video streams. ''Remaining'' shows the miss rate after applying PDP; ''eliminated'' shows the miss rate reduction; ''remaining'' plus ''eliminated'' shows the miss rate without applying PDP.
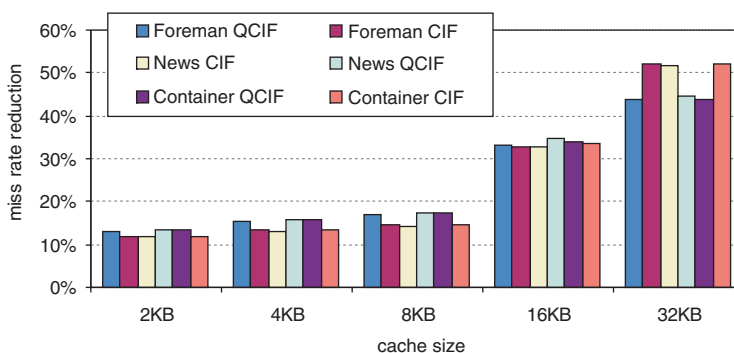
Fig. 8. Data cache miss reduction through PDP for different cache sizes and video streams.

larger than the inter-miss interval I, i.e., $M > I$. In that case, PDP is unable to timely prefetch the cache block from memory into the cache—a timely prefetch requires that $M \leqslant I$. (Recall that the prefetch is initiated after the previous cache miss; and the cache is a non-blocking cache.) In case $M > I$, PDP will hide I cycles of the memory latency M.

Fig. 9 quantifies the average load latency reduction through PDP. This graph clearly shows that PDP substantially reduces the average load latency, up to 2.5× for the 2KB cache.

6.3. Performance

Fig. 10 shows the IPC improvement in percentage relative to the baseline IPC (without prefetching). Fig. 11 shows the improvement in raw IPC over the baseline configuration without prefetching.

For the smallest cache size (2KB), PDP improves performance by 40–45%; this is because of the significant reduction in the average load latency (see Fig. 9). For the 4KB and 8KB cache, performance is improved by 20–25% and 10–12%, respectively. For the largest cache sizes (16KB and 32KB), the improvement in IPC is in the range of 1–2.5%. The reason why the IPC improvement for larger cache sizes is limited is due to the limited reduction in the average load latency (see Fig. 9). An interesting observation that can be made from Fig. 11 is that the performance for the 2KB cache plus PDP is better (by 12–15%) than for the 4KB cache without PDP. Similarly, the performance for the 4KB cache plus PDP is better (by 2–4%) than for the 8KB cache without PDP. As such, we conclude that for small caches, enhancing the memory subsystem with PDP is a better design option than doubling the size of the cache.
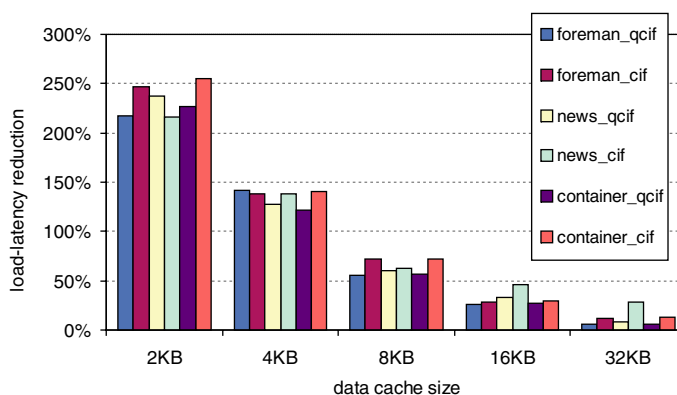

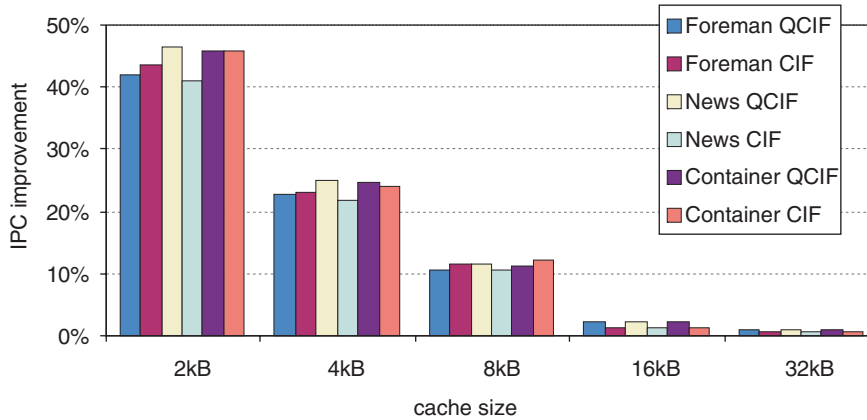Fig. 9. Load latency reduction through PDP.

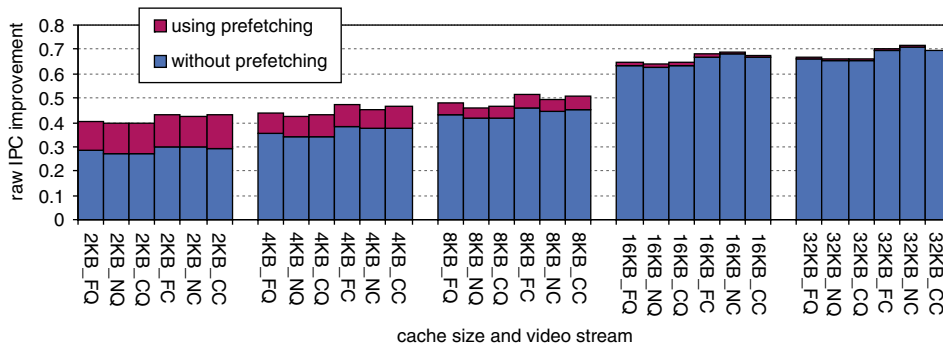Fig. 10. Percentage IPC improvement through PDP for varying data cache sizes.

Fig. 11. Raw IPC improvement through PDP for varying data cache sizes.

6.4. External memory accesses

It is well known that prefetching introduces additional accesses to higher levels in the memory hierarchy. The stop criterion is crucial in this respect to timely stop prefetching so that the number of external memory accesses is limited. This is an important issue when taking energy consumption into account. Accesses to external memory are costly due to the energy consumed on the bus as well as in external memory. As such, it is important to limit the amount of additional external memory accesses. Fig. 12 depicts the increase in external memory accesses due to PDP. We observe that the number of external memory accesses only slightly increases by a maximum of 0.6%.

6.5. Energy consumption

Fig. 13 shows the energy consumption reductions that are obtained through PDP. Recall that these numbers are overall system (on-chip plus off-chip) energy reductions. For the smallest data cache size (2KB), the energy reduction is as high as 20%. For the 4KB data cache, the energy reduction varies around 11.5%; for the 8KB data cache, an energy reduction is achieved of around 7%. Note that the energy consumption reduction, for example 20% for the 2KB cache, is not as high as the IPC increase which is 40% for the same example. This is due to the fact that the total execution time reduces proportionally to the increase in IPC thereby reducing the energy consumption.
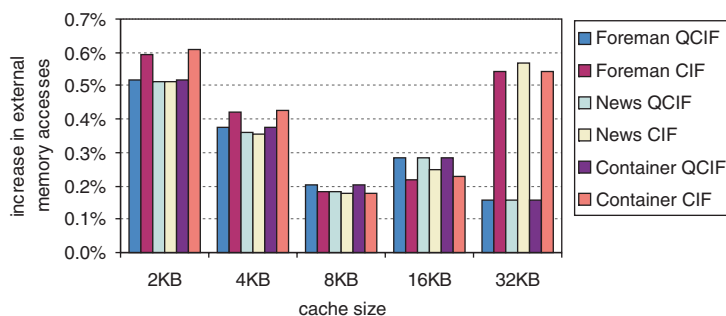
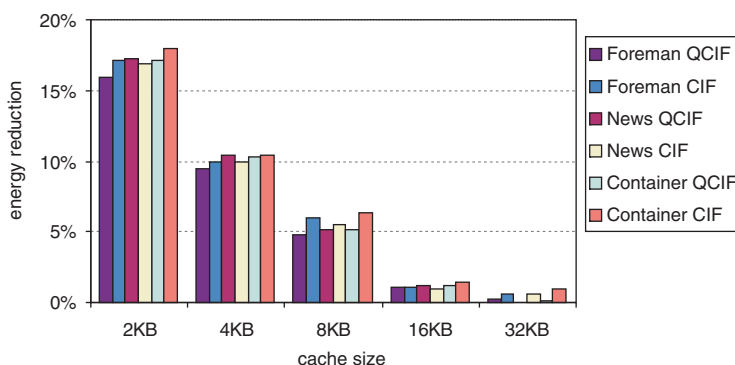Fig. 12. Increase in external memory accesses due to PDP.

Fig. 13. Energy reduction through PDP.

However, due to the higher IPC, the energy consumption per cycle increases. By consequence, an increase in IPC does not result in a proportional decrease in energy consumption.

6.6. PDP for other multimedia applications

As mentioned in the introduction, the design of PDP was initially motivated by the cache miss patterns as observed in the MPEG4 Visual stream decoder. Until now, we only considered the MPEG4 Visual stream decoder as our benchmark. The purpose of this section is to verify whether PDP could also be used for multimedia applications other than the MPEG4 Visual stream decoder. We considered a selection of MediaBench I and II benchmarks [15] in our analysis. Intuitively, PDP should work well if cache miss patterns exist in other multimedia applications. As such, a first step in our analysis is to verify whether cache miss patterns

exist. For the benchmarks that we studied we observed at least three benchmarks exhibiting a significant amount of cache miss patterns, namely *epic*, *unepic* and *ghostview* (see Fig. 14). Fig. 15
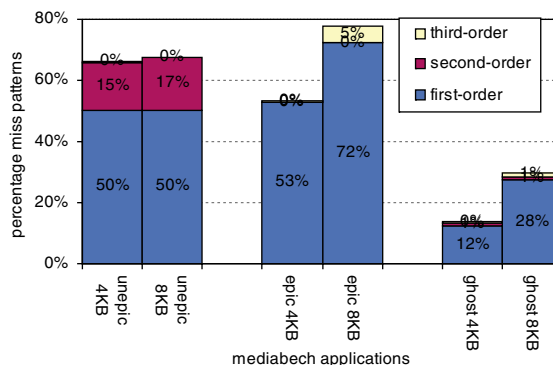

Fig. 14. The percentage of first-, second- and third-order miss patterns relative to the total number of cache misses for different data cache sizes and the tools.
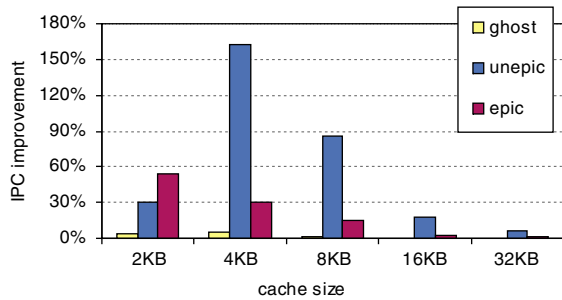
Fig. 15. Percentage IPC improvement through PDP for varying data cache sizes and various multimedia applications.
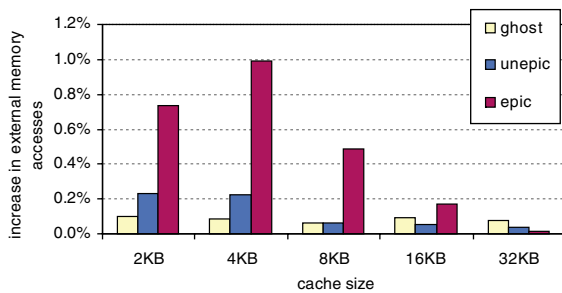

Fig. 17. Percentage of IPC improvement using PDP versus stream buffer for the QCIF container video stream.

Fig. 16. Increase in external memory accesses due to PDP for various multimedia applications.

Fig. 18. Data cache miss rate reduction using PDP versus stream buffer for the QCIF container video stream.

shows the performance speedup obtained from PDP. For unepic and a 4KB cache we observe a speedup up to 160%. Fig. 16 quantifies the increase in external memory accesses in terms of percentage. Again, we observe that PDP incurs a very small increase in external memory accesses, less than 1%. It is also important to note that the performance for the other multimedia applications having a small number of cache miss patterns (not shown here) was unaffected by PDP.

6.7. Comparing PDP versus stream buffer

In this section we present a comparison of PDP versus stream buffers. A stream buffer, as originally proposed by Jouppi [7], is a FIFO buffer that prefetches a stream of sequential cache blocks. On a cache miss, the stream buffer initiates a prefetch of the next sequential cache block. The stream buffer then continues prefetching sequential cache blocks, as memory bandwidth permits, until the stream buffer is full. We evaluated the stream buffer
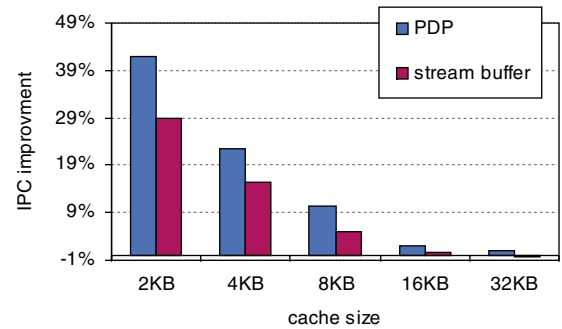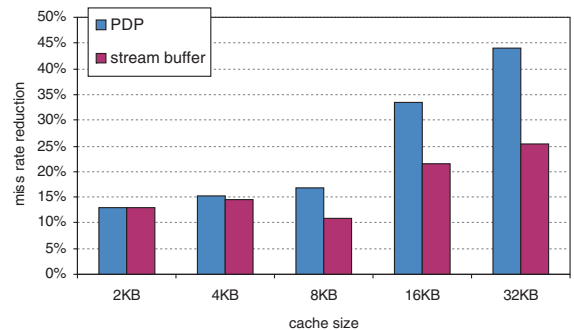
for a different number of entries (from 2 to 8). We only present the results for 2 entries since the performance of 4 and 8 entries was very close to 2 entries; and the 2-entry stream buffer results in significantly fewer external memory accesses. Fig. 17 shows the performance improvements obtained by PDP versus stream buffers; PDP clearly attains higher performance improvements (42% for a 2KB cache) than the stream buffer (29% for a 2KB cache). Fig. 18 shows the reduction in data cache miss rates (note that we consider a hit in the stream buffer as a hit in the cache). PDP clearly attains higher cache miss rate reductions for larger caches. For smaller caches (2KB and 4KB) however, the cache miss rate reduction is similar for PDP versus the stream buffer. Note that although the cache miss rate reduction is similar for small caches, the IPC improvement is much higher for PDP than for the stream buffer. This is due to the significantly lower number of additional external mem-

ory accesses for PDP compared to the stream buffer. These additional external memory accesses occupy the external bus and by consequence increase the average memory access time. We measured an up to 70% increase in external memory accesses for the stream buffer; recall that the number of additional external accesses was less than 1% for PDP. This obviously also has a positive effect on overall energy consumption since PDP has less bus accesses and less external memory accesses. This makes PDP a better design option than the stream buffer for embedded processors.

7. Conclusion

In this paper we have proposed a simple data prefetch mechanism called PDP that was shown to be effective for video processing (the MPEG4 Visual stream decoder in our setup) as well as other multimedia applications running on constrained embedded processors. The basic idea of PDP is to detect miss patterns in the sequence of data cache misses. These miss patterns are based on the inter-miss stride (the memory address stride between two data cache misses) and the inter-miss interval (the number of clock cycles between two data cache misses). We have defined the concept of the m-order miss pattern. These miss patterns are exploited in PDP to drive the prefetch mechanism, i.e., when a miss pattern is detected, a prefetch action is initiated. One particular important design issue for our PDP mechanism was the stop criterion in order to reduce cache pollution and the number of additional memory accesses (typically less than 1%). Experimental results using cycle-level power/performance simulations running various MPEG4 video streams show that significant performance increases and energy decreases can be obtained using PDP for embedded processors. For example, a 40%, 20% and 10% performance increase in conjunction with a 17%, 10% and 5% overall system (on-chip plus off-chip) energy reductions are observed for 2KB, 4KB and 8KB caches, respectively. Our experimental results indicate that multimedia applications (other than the MPEG4 Visual stream decoder for which PDP was initially designed) can also benefit significantly

from PDP; we report performance improvements up to 160%. Finally, we compared PDP versus stream buffer prefetching. We conclude that the performance improvement obtained through PDP is significantly higher than for the stream buffer. This is mainly due to the lower number of additional external memory accesses for PDP compared to the stream buffer.

Acknowledgements

References

[1] Intel corporation, The Intel XScale microarchitecture technical summary. Available from: <ftp://download.intl.com/design/intelxscale/XSacleDatasheet4.pdf>.

[2] International Standard ISO/IEC 14496-2, Information Technology—Coding of Audio–Visual Objects—Part 2: Visual, second ed., 2001-12-01.

[3] International Standard ISO/IEC 14496-5, Reference Software, second ed., 2001-12-15.

[4] D. Burger, T.M Austin, The SimpleScalar tool set, version 2.0, Computer Architecture News (June) (1997) 13–25.

[5] D. Callahan, K. Kennedy, A. Porterfield, Software prefetching, in: Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, 1991, pp. 40–52.

[6] J. Fu, J. Patel, B. Janssens, Stride directed prefetching in scalar processors, in: Proceedings of the 25th International Symposium on Microarchitecture, December 1992, pp. 102–110.

[7] N.P. Jouppi, Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers, in: Proceedings of the 17th Annual International Symposium on Computer Architecture, May 1990, pp. 364–373.

[8] S. McKee, Z. Fang, M. Valero, An MPEG-4 performance study for non-SIMD, general purpose architectures, in: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2003), March 2003.

[9] S. Palacharla, R.E. Kessler, Evaluating stream buffers as a secondary cache replacement, in: Proceedings of the 21st Annual International Symposium on Computer Architecture, April 1994, pp. 24–33.

[10] P. Reungsang, S.K. Park, S.-W. Jeong, H.-L. Roh, G. Lee, Reducing cache pollution of prefetching in a small data cache, in: Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors, September 2001, pp. 530–533.

[11] H. Sbeyti, S. Niar, L. Eeckhout, Adaptive prefetching for multimedia applications in embedded systems, in: Proceedings of the 2004 Design, Automation and Test in Europe Conference and Exhibition (DATE'04), Vol. 1, February 2004, pp. 1350–1351.

[12] W.-T. Shiue, C. Chakrabarti, Memory exploration for low power embedded systems, in: Proceedings of the 36th Annual ACM IEEE Design Automation Conference (DAC), June 1999, pp. 140–145.

[13] W. Tang, R. Gupta, A. Nicolau, A. Veidenbaum, Fetch size adaptation vs. stream buffer for media benchmarks, in: Third Workshop on Media and Streaming Processors (in conjunction with MICRO-34), December 2001.

[14] D.F. Zucker, R.B. Lee, M.J. Flynn, Hardware and software cache prefetching techniques for MPEG benchmarks, IEEE Transactions on Circuits and Systems for Video Technology 10 (5) (2000) 782–796.

[15] C.Lee, M. Potkonjak, W.H. Mangione-Smith, Media-Bench: a tool for evaluating and synthesizing multimedia and communications systems, in: Proceedings of the 30th International Annual Conference on Microarchitecture, December 1997, pp. 330–335.

Hassan M. Sbeyti received the Dipl.-Ing degree in Electrical Engineering and Information Science form the Ruhr Universitt Bochum (Germany) in 1993. He worked many years in the development of PC I/O devices and their drivers. In 2001, he received the DEA (Masters) in Informatics, Modeling and Intensive Calculation from AUF and the Lebanese University in Beirut. He started his PhD at the University of Valenciennes (ISTV, LAMIH ROI) in 2002. His main research interests include computer architecture and memory optimization of embedded systems.

Smail Niar obtained his PhD degree in Computer Science from the University of Lille, France, in 1999. He currently is an associate professor in computer science at the University of Valenciennes, France. His main area of research is computer architecture. He is interested by all aspects of the design and evaluation of computer systems for high performance as well as for embedded systems.

Lieven Eeckhout was born in Kortrijk, Belgium in 1975. He received the Engineering degree and PhD degree in Computer Science from Ghent University, Belgium, in 1998 and 2002, respectively. Lieven Eeckhout currently is a Postdoctoral Fellow of the Fund for Scientific Research—Flanders (Belgium) (F.W.O.—Vlaanderen). His research interests include computer architecture, performance analysis and workload characterization.