확장 버퍼 캐쉬의 설계 및 성능 평가

홍 원 기[†]

요 약

VLIW 프로세서는 간단한 하드웨어 구조로 인해 저전력 및 고성능을 제공하여 임베디드 시스템에 매우 적합한 프로세서 구조로 인식되고 있다. 그러나 VLIW 프로세서는 동시에 수행 가능한 명령어들의 집합인 명령어 패킷 길이가 일정하지 않기 때문에 메모리 접근 지연 시간이 늘어나는 문제점을 안고 있다. 이는 가변 길이의 명령어 패킷으로 인해 일부 명령어 패킷이 두개의 캐쉬 블록에 걸쳐 있게 되고(스트래들 명령어 패킷), 이러한 명령어 패킷을 읽어 오기 위해 두 번의 캐쉬 접근이 요구되기 때문이다. 본 논문에서는 명령어 인출 대역폭을 높여줄 뿐만 아니라 명령어 캐쉬의 전력 소모를 낮춰주는 확장 버퍼 캐쉬를 제안한다. 확장 버퍼 캐쉬는 메인 캐쉬와 함께 스트래들 명령어 패킷의 일부를 저장하기 위한 소량의 확장 버퍼 캐쉬를 갖고 있으며 스트래들 명령어 패킷으로 인해 추가적으로 발생하는 캐쉬 접근을 줄여준다. 실험 결과 스트래들 명령어 패킷으로 인한 캐쉬 접근을 줄여 줌으로써 확장 버퍼 캐쉬는 기존 명령어 캐쉬에 비해 약 5~19%의 성능·전력·비용 향상을 가져옴을 확인할 수 있었다.

Design and Performance Evaluation of Expansion Buffer Cache

Won-Kee Hong[†]

ABSTRACT

VLIW processor is considered to be an appropriate processor for the embedded system, provided with high performance and low power consumption due to its simple hardware structure. Unfortunately, the VLIW processor often suffers from high memory access latency due to the variable length of I-packets, which consist of independent instructions to be issued in parallel. It is because of the variable I-packet length that some I-packets must be placed over two cache blocks, which are called straddle I-packets, so that two cache accesses are required to fetch such I-packets. In this paper, an expansion buffer cache is proposed to improve not only the instruction fetch bandwidth, but also the power consumption of the I-cache with moderate hardware cost. The expansion buffer cache has a small expansion buffer containing a fraction of a straddle packet along with the main cache to reduce the additional cache accesses due to the straddle I-packets. With a great reduction in the cache accesses due to the straddle packets, the expansion buffer cache can achieve 5~19% improvement over the conventional I-caches in the Delay·Power·Area metric.

키워드 : VLIW, 임베디드 시스템(Embedded System), 명령어 캐쉬(Instruction Cache), 명령어 인출 대역폭(Instruction Fetch Bandwidth), 전력 소모(Power Consumption)

1. Introduction

ILP processors improve the system performance by executing several independent instructions simultaneously. Recently, the rapid advance of the semiconductor and network technology makes the ubiquitous and mobile computing be realized. The embedded system is getting more and more important as a core platform for the ubiquitous and mobile computing. The VLIW processor, which is one of several ILP processors, is in the limelight as a processor for the embedded and mobile system since it can achieve high performance with low power consumption due to the simple hardware circuit[3].

Basically, the VLIW processor is performed by the instruction-packet (I-packet) basis. The I-packet is a group of independent instructions to be issued in parallel that are determined at compile time. Because the I-packets are different from one another in the number of instructions, their lengths are not fixed. The variable I-packet length does not assure that the starting or ending address of a particular I-packet is always matched with the address of a block boundary in the cache. That is, many I-packets happen to be stored in the two consecutive cache blocks, which are called straddle I-packets. In the conventional cache, the straddle I-packets increase the overall instruc-

※ This research was supported by the Daegu University Research Grant, 2004.

† 정 회 원 : 대구대학교 정보통신공학부 교수

논문접수 : 2004년 8월 27일, 심사완료 : 2004년 12월 2일

tion fetch latency since every reads of straddle I-packets requires the two sequential cache accesses, which is called double access.

In order to decrease the instruction fetch latency due to the straddle I-packets, the bank cache [9] was proposed. It divides a cache into two banks and stores instructions in each bank in the interleaving fashion. Every I-packet requests from the processor fetches the corresponding front block and the next rear block from the two banks respectively and simultaneously[1]. Therefore, it can remove the double accesses due to the straddle I-packets. However, it increases the overall power consumption by accessing two banks at every cache accesses. Moreover, it requires a post-fetch stage to be placed between the fetch stage and the decode stage in the processor's execution pipeline in order to rearrange the fetched two blocks from each banks. The additional pipeline stage increases the branch misprediction penalty since it defers the detection of misprediction. The branch penalty can countervail the performance improvement gotten by the removal of double access. In addition, the delayed detection of misprediction may cause a bad effect on the power dissipation since it can increase the fetch of useless I-packets. The power consumption becomes one of the most important constraints to be resolved in processor design as the embedded and mobile computers are prevailed [4, 5, 10].

In this paper, an expansion buffer cache is presented that consists of a conventional main cache and a small direct-mapped addressing buffer called an expansion buffer. It decreases the instruction fetch latency by effectively reducing the number of double accesses, while it achieves a very close or evenlower power consumption than the conventional cache. The expansion buffer stores only the tail I-packets. It is accessed along with the main cache only when a requested I-packet is known as a straddle I-packet in order to reduce power consumption due to the expansion buffer access. It is roughly detected whether an I-packet is the straddle I-packet or not by using the offset address of program counter. If the expansion buffer contains the tail I-packet, the double access can be avoided. Because the instructions read from the expansion buffer are always in order, it does not need an additional fetch stage for instruction rearrangement in the instruction pipeline. Therefore, it can also minimize the branch misprediction penalty and the number of unnecessary cache accesses.

1) The instructions of the straddle I-packet in the front block is the *head I-packet* and those in the rear block is the *tail I-packet*.

The expansion buffer cache is compared with the conventional direct-mapped cache and the bank cache, and evaluated in terms of memory access latency, power consumption, and the hardware cost using the IMPACT tool [11]. Recently, as the mobile computing and ubiquitous computing based on the low power embedded system is rapidly growing, the power consumption appears as one of the important performance metrics, along with the execution time and the hardware cost. The Delay · Power metric, an integrated metric to evaluate a system in terms of the execution time and the power consumption, has been proposed and cited widely in the literature [17, 18]. In this paper, the Delay·Power·Area metric is used for the integrated evaluation of the execution performance, power consumption and the hardware cost. As an extended metric of the Delay·Power, it is computed by multiplying the normalized memory access latency, power consumption and chip area. The experimental results show that the expansion buffer can achieve maximum Delay(Power improvement of 10% over the conventional direct-mapped caches and 19% over the bank caches.

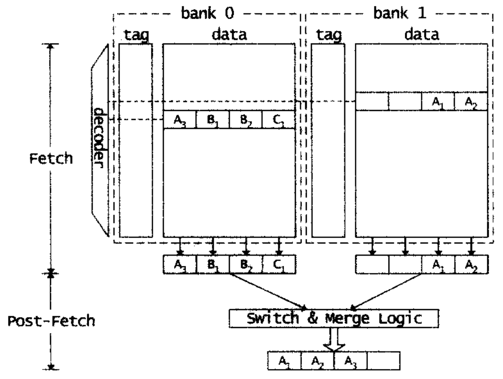
This paper is organized as follows : Section 2 discusses related work and explains the structure and the fetch mechanism of the bank cache in more detail. The organization and operation flow of expansion buffer cache will be introduced in Section 3. Section 4 presents the experimental methodology and results measured with various metrics. Some final remarks will be drawn in Section 5.

2. Related Work

There have been several studies to achieve high-band width instruction fetching. The branch address cache [12] and the collapsing buffer cache [13] support fetching of non-contiguous blocks with a multi-ported, multi-banked, or multiple copies of the I-cache. However, this leads to multiple fetch groups that must be aligned and collapsed at fetch time, which can increase the fetch latency. The trace cache [1, 2] does not require fetching of non-consecutive basic blocks from the I-cache, as it stores dynamically constructed sequences of basic blocks in a special purpose cache. It performs all instruction alignment and collapsing at completion time. At completion time, a fill unit constructs traces of instructions to be stored in the trace cache. It optimizes the fetch latency by shifting the complexity of multiple-branch prediction, multiple fetch groups, and instruction alignment and collapsing to the completion time.

Multipath execution schemes [6-8] were proposed to reduce the misprediction penalties incurred by branches. They fetch and execute instructions from both paths after the branch. By fetching and executing from both possible branch targets, they assure that the branch cannot be predicted incorrectly at all. However, it requires tremendous hardware complexity since it always keeps track of instructions from multiple paths that reside in the pipeline at the same time.

The above schemes were designed mainly for the superscalar processors but the bank cache [9] was proposed for VLIW processor. (Figure 1) shows an example of the bank cache organization in a four-issue VLIW processor, where a cache block can accommodate four instructions. Because an I-packet can be placed over two cache blocks, both cache banks are always accessed at the same time. One of the cache banks contains the corresponding block in which first instructions of the requested I-packet reside, and the other bank holds the sequential successive block in which the last instructions are stored. Because an instruction is of variable length and does not always begin on a block boundary in memory, the low-order bits are used as an offset to index to the start of the instruction in the cache block. When an instruction address is presented to address the I-cache, the cache address decoder selects consecutive blocks in both cache banks.



(Figure 1) The organization of the bank cache

As shown in (Figure 1), an I-packet from the bank cache will pass through two pipeline stages such as the fetch and the post-fetch stage. In the fetch stage, the sequential successive block in another bank is read along with the corresponding cache block addressed by the program counter. In the post-fetch stage, the rearrangement of two fetched blocks should be made via block swapping, because the memory instructions within an I-packet will be executed in the order that is consistent with their sequential left-to-right order. Then, the rearranged blocks are merged and the requested I-packet is extracted.

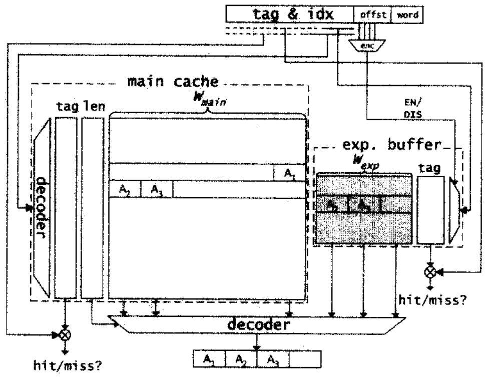
The additional pipeline stage leads to the heavier penalty of branch misprediction, because the detection of misprediction is deferred due to the insertion of a post-fetch stage between the instruction fetch stage and the branch detection stage. In addition, the delayed detection of misprediction as well as the double access has a detrimental effect on power dissipation. This is because the number of unnecessary I-packets referenced during the time between the mispredicted branch and the next I-packet on the correct path will be increased.

3. Expansion Buffer Cache

The objective of designing the expansion buffer cache is to decrease the memory access latency and power consumption due to the double access for straddle I-packets with the additional cost of chip area. This is achieved by placing a small buffer along with a main cache in the memory hierarchy, which minimizes the number of double accesses for straddle I-packets and removes the post-fetch stage in the instruction pipeline.

3.1 Organization of the Expansion Buffer Cache

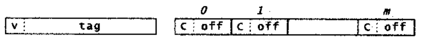
The expansion buffer cache is equipped with a conventional main cache and a small expansion buffer as shown in (Figure 2). The expansion buffer as well as the main cache is addressed by the direct-mapped addressing scheme. Because I-packets, each of which consists of multiple concurrent instructions, are variable in length and do not always begin at the boundary of a block in the cache, the offset address, which is the low-order bits of the requested I-packet address, is used to locate the I-packet in the cache block. While the tag array of the expansion buffer holds only the tag address for each entry, that of the main cache contains the I-packet length information as well as tag addresses, in order to support an I-packet fetch and a Next Program Counter (NPC) computation. The tag address is possessed by each cache block, but the I-packet length information is maintained on an instruction basis. (Figure 3)(a) shows the I-cache structure, i.e., a tag field and a length field entry for the main cache, where each cache block contains *m* instructions. A carry bit and an offset field are associated with each instruction to indicate the length of the corresponding I-packet as shown in (Figure 3)(b). It is used for completing the NPC computation while fetching an I-packet from the I-cache [14]. The carry bit is set to one if the next sequential I-packet does not reside in the same cache block. Each offset field has the offset of the next sequential I-packet within its cache block. In order to compute the NPC, the corresponding carry bit of the starting instruction in a requested I-packet is added to the tag and index, and the offset value is copied to the offset field of the Program Counter (PC).



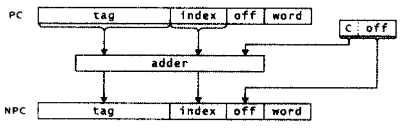
(Figure 2) The organization of the expansion buffer cache

The expansion buffer width (W_{exp}) is chosen to be less than the length of an I-packet (n). Specifically, W_{exp} is the expansion buffer width such that $0 < W_{exp} < n$, where n is the maximum number of instructions in an I-packet, because each entry is used for storing only a portion of a straddle I-packet. For example, in a four-instruction I-packet processor, the expansion buffer can be constructed as one-, two-, or three-instruction width. The expansion buffer can be constructed as a fully associative buffer with a Content Addressable Memory (CAM) structure to reduce conflict misses, but in this study, Static Random Access Memory (SRAM) is selected for the expansion buffer, to curtail hardware cost and power dissipation.

In fact, it is only necessary to retrieve the expansion buffer for straddle I-packets. Access to the expansion buffer incurs the additional power dissipation. In order to reduce the number of unnecessary accesses to the expansion buffer, the offset encoder, which is used for selectively enabling the expansion buffer, is provided. The offset encoder examines whether the block offset of the program counter belongs within the $[W_{main}-(n-1)]^{th}$ offset in the main cache block (W_{main} represents the maximum number of instructions that a main cache block can contain). If not so, the decoder in the expansion buffer will be enabled to retrieve the instructions within the I-packet. Although the offset encoder is on the critical path, the overall access latency will not be prolonged because the access time to the much smaller expansion buffer, which is much faster than that of the main cache, is short enough to check for a straddle I-packet.



(a) tag and length fields.



(b) NPC computation(off : offset, c : carry bit)

(Figure 3) The structure of tag and I-packet length field entry for the main cache and NPC computation

Because memory instructions, which are dependent on one another, are allowed to be in the left-to-right order within the same I-packet by the ILP compiler, it must be assured that they are executed in their original order. In the expansion buffer cache, the column decoder is implemented by simply extending the column decoder of the main cache to select n consecutive instructions up to those in the expansion buffer. Because the expansion buffer stores only the rear part of an I-packet belonging to the sequential successive block, the order of instructions in an I-packet is always maintained. Therefore, it is not necessary to reorder the instructions in the I-packet fetched from the expansion buffer cache. It means that an additional pipeline stage for the instruction rearrangement is not required. The instructions fetched from the main cache and the expansion buffer pass through the column decoder, where the instructions corresponding to the requested I-packet are extracted by the block offset address and the length information.

3.2 Operational Flow

Cache management for accessing the I-packets, except the straddle I-packets, is the same as for the conventional direct-mapped cache. When a straddle I-packet is demanded, the main cache is searched for the front part of the straddle I-packet and at the same time, the expansion buffer is searched for its rear part. Depending on the hit or miss in the main cache and the expansion buffer, the following operational flow is performed.

① *Hit in both the main cache and the expansion buffer* : The I-packet can be fetched by simply accessing the corresponding instructions from both the main cache and the expansion buffer.

② *Hit in the main cache but miss in the expansion buffer*: It means that the front part of the straddle I-packet is held in the main cache, but its rear part is not in the expansion buffer. Hence, the main cache is searched again for the successive cache block containing the rear part, with one cycle penalty. If not in the main cache, it is read from the lower level memory. The rear part of the straddle I-packet belonging to the successive block is stored in the expansion buffer along with the supply of the demanded I-packet to the processor.

③ *Miss in the main cache* : the corresponding block is fetched from the lower level memory. In addition, if the successive block does not reside in the main cache, it is read from the lower level memory. After that, the rear part of the straddle I-packet is copied into the expansion buffer.

For example, let us assume that the maximum number of instructions in an I-packet is four and each cache block can accommodate eight instructions, as shown in (Figure 2). It is supposed that each entry in the expansion buffer can contain up to three instructions. If an I-packet A which consists of three instructions, A1, A2, and A3, is requested, the decoder of the expansion buffer will be driven to retrieve the corresponding entry, since the offset address of the I-packet A (seven) is greater than $W_{main}-(n-1)$, i.e., five. (Figure 2) shows that the expansion buffer contains the rear parts of the I-packet A, i.e., A2 and A3, which are stored in the successive block of the main cache. Therefore, instead of being accessed twice to fetch the straddle I-packet, the expansion buffer cache extracts A1 from the main cache, and A2 and A3 from the expansion buffer through the column decoder at the same time.

4. Performance Evaluation

The details of the simulation environment and empirical results in terms of memory access latency, power consumption, and hardware cost are presented in this section. As the energy dissipation is now considered as one of the important metrics to assess a processor, execution performance alone is not a good measurement to evaluate an I-cache model. In this study, the Delay·Power metric is used to combine both execution performance and power consumption into a single metric. Moreover, the performance of the expansion buffer cache is measured with a new metric, Delay·Power·Area, which considers the impact of not only the execution time and the power consumption, but also the hardware cost for the processor evaluation. The conventional direct-mapped cache architecture (CONV) and the bank cache architecture (BANK) are selected as the architectures with which to compare the expansion buffer cache (EXP).

4.1 Simulation Environment

All simulations in this study are conducted using the IMPACT tool set from the University of Illinois. The simulation architecture is based on a four-issue pipelined VLIW machine. For fair comparisons with the BANK, which suffers from branch misprediction more significantly than do the other I-cache models, a 512-entry Branch Target Buffer (BTB) with correlation branch prediction is considered [15]. The correlation branch predictor provides high branch prediction accuracy. This is because while most branch predictors only consider the past behavior of the current branch instruction, the correlation branch predictor considers the behavior of other branches to predict the target of the current branch. It requires the Global Branch History register with a counter table, which contains the outcome of the *n* most recent branches. This information is combined with the address of the branch under consideration to index the counter table.

<Table 1> Simulation parameters

System Parameters	Value
CPU clock	400MHz
I-cache size	16Kbytes
I-cache block size	32, 64bytes
D-cache size	Infinite
Memory latency	15CPU cycles
Memory bandwidth	1.6Gbytes/sec

<Table 1> shows the basic system parameters for simulation. Two sizes of a cache block, e.g., 32 bytes and 64 bytes, are analyzed to evaluate the impact of straddle instructions, and the number of straddle I-packets differs depending on the cache block size chosen. It is assumed that the data cache access is always hit and there is no second-level cache. The bus bandwidth is assumed to be 64 bits per cycle, and the latency taken for the first word to come back from the main memory is assumed to be 15 processor cycles.

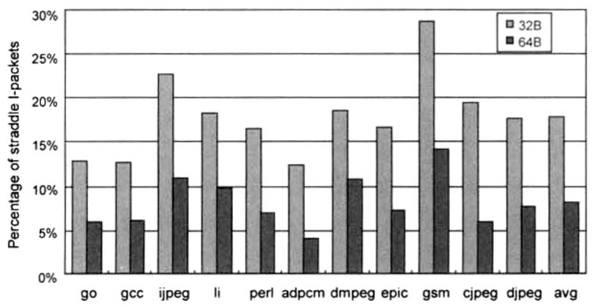
Five out of eight SPECint95 benchmarks and six other applications from MediaBench are used for performance analysis of each I-cache model. All benchmarks are compiled with superblock optimization by the IMPACT compiler. <Table 2> shows the benchmarks, the training and input data sets, the dynamic instructions analyzed, the percentage of dynamic branches, and the misprediction ratio.

<Table 2> Characteristics of benchmarks

	training set	input set	dynamic instructions	%dynamic branch	misprediction ratio (%)
go	2stone9.in	5stone21.in	247201051	19.5	26.0
gcc	amptjp.i	varsasm.i	10400034	26.1	15.4
ijpeg	vigo.ppm	penguin.ppm	231201344	7.4	10.5
li	train.lsp	test.lsp	167200845	21.2	7.2
perl	jumble.pl, jumble.in	scrabble.pl, scrabble.in	25653135	22.7	8.1
adpcm	clinton.pcm	clinton.pcm	5821861	27.8	26.8
dmpeg	test.m2v	mei16v.m2v	8600063	11.3	2.3
epic	testimg.pgm	testimg.pgm	9792564	14.8	1.0
gsm	clinton.pcm	S_16_44.pcm	15400104	8.6	5.3
cjpeg	testimg.ppm	monalisa.ppm	10000051	17.9	5.9
djpeg	testimg.jpg	monalisa.jpg	10400034	5.8	6.2

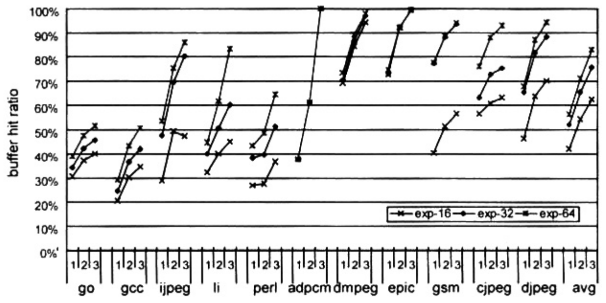
4.2 Experimental Results

(Figure 4) shows the access ratio of the straddle I-packets over the total number of cache accesses as the cache block size varies in the CONV. Because the double access is caused by the straddle I-packets, the double access ratio is equal to the percentage of the straddle I-packet over the total number of cache accesses. In general, small block size induces more accesses to the straddle I-packets than large block size. The straddle I-packets in the 32-byte block cache correspond to about 18% of the total dynamic I-packets on average, while those in the 64-byte block cache account for about 8.2%.

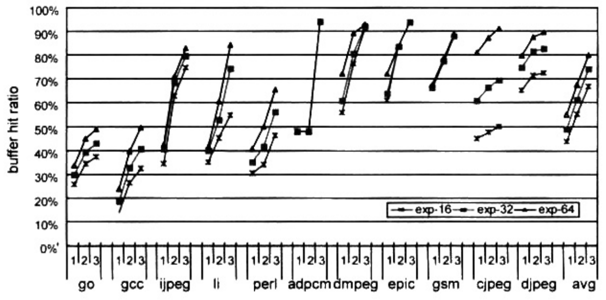


(Figure 4) Percentage of straddle I-packets

In order to achieve performance improvement in the EXP, the expansion buffer is used to prevent the main cache from being doubly accessed due to the straddle I-packets. We examined the expansion buffer hit ratio as the buffer width and the number of entries changed. The number on the X-axis in (Figure 5) indicates the buffer width and several expansion buffers with a different number of entries, e.g., 16 entries (EXP-16), 32 entries (EXP-32), and 64 entries (EXP-64). (Figure 5) shows that, for a given hardware cost, extending the expansion buffer width is better than increasing the number of buffer entries. Specifically, in the MediaBench applications such as adpcm, dmpeg, and epic, even the EXP-16 of three-instruction width can almost completely avoid the double access. However, in spite of the buffer expansion in the ijpeg, the buffer hit ratio is decreased due to the tremendous conflict miss in the 32-byte block EXP. The average hit ratio in EXP-64 of three-instruction width is 83% and 86% in the 32-byte and 64-byte cache block, respectively, while those in the EXP-16 are about 62% and 73%, and those in the EXP-32 about 76% and 80%.

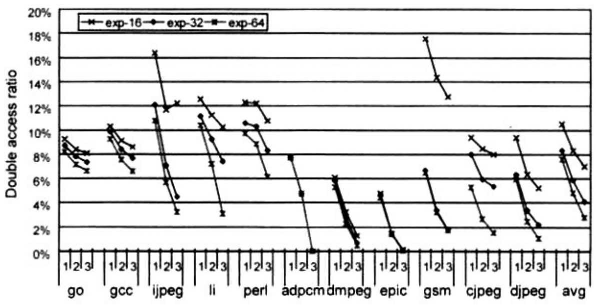


(a) 32-byte cache block

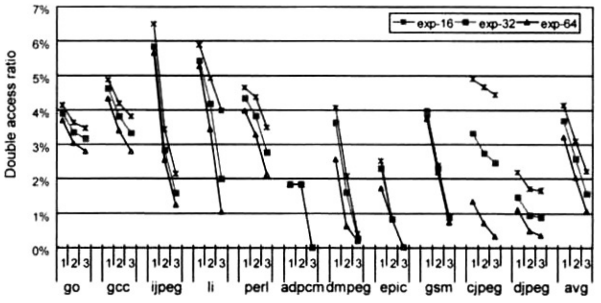


(b) 64-byte cache block

(Figure 5) Buffer hit ratio

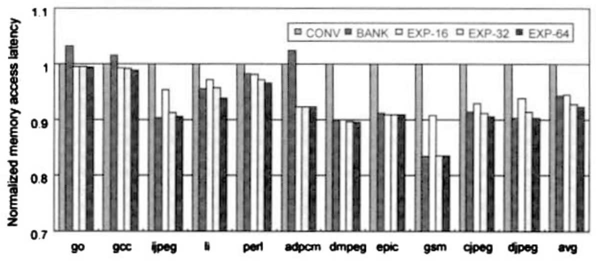


(a) 32-byte cache block

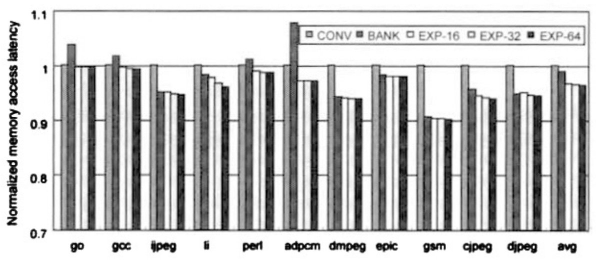


(b) 64-byte cache block

(Figure 6) Double access ratio



(a) 32-byte cache block



(b) 64-byte cache block

(Figure 7) Normalized memory access latency

(Figure 6) shows that the double access ratio, which is the percentage of the double accesses over all the cache accesses, is decreased as the number of entries and the buffer width are increased. The double access ratio is decreased down to about 2.77%~7.01% in the 32-byte cache block, and 1.06%~2.21% in the 64-byte cache block, on average, depending on the number of buffer entries. It means that the expansion buffer can reduce the amount of double accesses in the CONV by more than 61%, which is calculated from the CONV's access ratio of the straddle I-packets over the total number of cache accesses and the EXP's double access ratio.

The decreased double access ratio due to the expansion buffer causes a positive effect on the memory access latency. (Figure 7) shows the memory access latencies in each I-cache model normalized by that in the CONV. As shown in (Figure 7), the EXPs, where all of them have the three-instruction-width buffer, can achieve a performance average gain of about 3%~8% over that of the CONV. Although the BANK gets rid of the double accesses completely, its memory access latency is equal to or even higher than that of the EXP, due to the high misprediction penalty.

4.3 Low Power Consumption

To estimate power consumed by the I-caches, the analytical model developed by Kamble and Ghost [15] was used and adapted to the expansion buffer cache and the other comparison I-caches. It gives an accurate power estimate since it uses the runtime statistics such as hit/miss counts, fraction of read/write requests, and assumes stochastic distributions for signal values.

According to [15], the main sources of power are composed of the following four components : E_{bits}, E_{word}, E_{output}, and E_{input}, which denote the energy dissipation for bit-lines, word-lines, address and data output-lines, and address input-lines, respectively. When $N_{bit,pr}$, $N_{bit,r}$, $N_{bit,w}$ are the total number of bit line transitions due to precharging, reads and writes, CA is the total number of cache accesses, N_{aout}, N_{din} are the number of transitions on the address and data line drivers, and N_{ainput} is the number of transitions in the address input lines, the overall energy dissipation in a cache can be defined as follows :

$$E_{cache} = E_{bits} + E_{word} + E_{output} + E_{input},$$
$$E_{bit} = 0.5 \cdot V_{dd}^2 \cdot [N_{bit,pr} \cdot C_{bit,pr} + (N_{bit,w} + N_{bit,r}) \cdot C_{bit,r/w} + CA \cdot N_{columns} \cdot (C_{g,Q_{bo}} + C_{g,Q_{pb}} + C_{g,Q_P})]$$
$$E_{word} = V_{dd}^2 \cdot CA \cdot C_{wordline},$$
$$E_{output} = 0.5 \cdot V_{dd}^2 (N_{aoutput} \cdot C_{aoutput} + N_{dinput} \cdot C_{dinput}),$$
$$E_{ainput} = 0.5 \cdot V_{dd}^2 \cdot N_{ainput} \cdot (2 \cdot (b+1) \cdot B \cdot C_{in}, dec + C_{awire}),$$

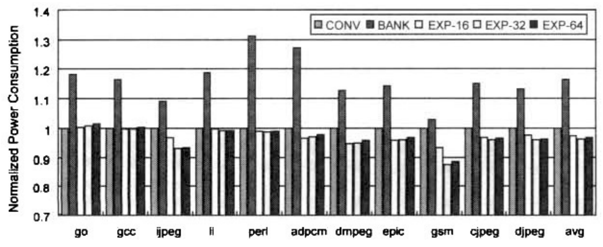
where $C_{bit,pr}$ and $C_{bit,r/w}$ are the effective load capacitance of the bit lines during precharging and read/write to the cell respectively, C_{g,Q_x} is the gate capacitance of transistor Q_x, $C_{wordline}$, C_{aout} and C_{din} are load capacitance of the word, address, and data line driver, B and b are the numbers of cache blocks and banks, $C_{in,dec}$ is the gate capacitance of the first level of decoder, and C_{awire}

is the wire capacitance of the common address lines feeding the decoder.

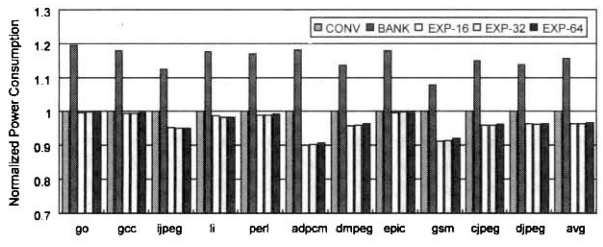
〈Table 3〉 Transition counts for power equation

	Main Cache	Expansion Buffer
CA	$N_{hit}+N_{miss}+N_{double\_access}$	N_{buf_access}
$N_{bit,pr}$	$0.5\cdot(N_{hit}+N_{miss}+N_{double\_access})\cdot N_{columns}$	$0.5\cdot N_{buf\_access}\cdot N_{columns}$
$N_{bit,r}$	$0.5\cdot(N_{hit}+N_{miss}+N_{double\_access})\cdot N_{columns}$	$0.5\cdot N_{buf\_access}\cdot N_{columns}$
$N_{bit,w}$	$0.5\cdot N_{miss}\cdot N_{columns}$	$0.5\cdot N_{buf\_miss}\cdot N_{columns}$
$N_{aoutput}$	$0.5\cdot N_{miss}\cdot N_{addr\_bus}$	N/A
N_{dinput}	$0.5\cdot N_{miss}\cdot N_{inst\_bus}$	N/A
N_{ainput}	$0.5\cdot(N_{hit}+N_{miss}+N_{double\_access})\cdot W_{addr\_bus}$	$0.5\cdot N_{buf\_access}\cdot W_{addr\_bus}$

Based on this equation, the analytical models for estimating power dissipation of each I-cache model can be obtained using the transition counts. In the CONV and EXP, the number of the double accesses, N_{double_access}, due to the straddled instruction is added to the overall cache accesses. The number of columns, $N_{columns}$, in the BANK is twice the size of that in the CONV cache, while the number of rows decreases by 50%. The energy dissipation due to the accesses to the expansion buffer in the EXP should be considered. The number of accesses to the expansion buffer, N_{buf_access}, is defined apart from the cache accesses to the main cache because the expansion buffer access is performed only when a requested instruction has the possibility of extending over two cache blocks. The transition counts used for the analytical model are summarized in <Table 3>. The parameter N_{buf_miss} denotes the number of expansion buffer misses.



(a) 32-byte cache block

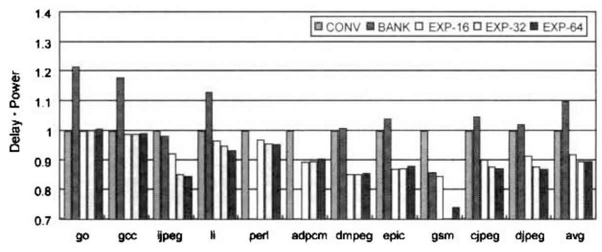


(b) 64-byte cache block

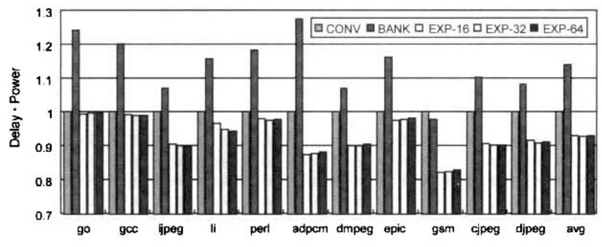
(Figure 8) Normalized power consumption

(Figure 8) presents the normalized power consumption in each I-cache architecture. In most applications, except gcc and go, the EXP achieves lower power than CONV, irrespective of its number of entries. Because in the gcc and go, the portion of the straddle I-packets in the overall dynamic instructions is relatively small and conflict misses due to large code size frequently happen in the expansion buffer, the impact of the expansion buffer on the reduction of double accesses is relatively small, so that the power consumption in the EXP is nearly equal to or even slightly higher than that in the CONV. It is shown that the BANK requires high power consumption due to the simultaneous accesses to the two cache blocks. The power consumption in the EXP is about 1.20 times lower than that in the BANK, while the CONV is about an average of 1.16 times lower than the BANK.

(Figure 9) shows the Delay·Power for each I-cache model. The Delay is the normalized memory access latency based on that of the CONV. The Power is also normalized by the amount of power consumed in the CONV. Although the BANK is better than the CONV in terms of memory access latency, it consumes much more energy than the CONV. Therefore, the Delay·Powers of the 32-byte and 64-byte block BANK are about 1.09 and 1.14 times higher than those of the CONV. On the other hand, because the EXP retains low power consumption with high performance, it achieves up to about 10% and 19% improvement in the Delay·Power over the CONV and the BANK, respectively.



(a) 32-byte cache block



(b) 64-byte cache block

(Figure 9) Evaluating I-cache systems using Delay(Power metric)

4.4 Hardware Cost

The on-chip cache area model suggested for a set-associative cache by Mulder et al. [16] is employed in the analysis of hardware cost. It considers not only the area for data storage, but also the areas for the control logic, address tags, comparators, and status bits. The total cache area, A_{cache}, can be calculated as follows :

$$A_{cache} = A_{cl} + A_{tag} + A_{data},$$

where $A_{cl}, A_{tag}, A_{data}$ denote the area for control logic, tag and status bits, and data array, respectively. The areas of all components are then converted into an equivalent register area. The area unit, rbe, equals the area of the register cell. The SRAM cell area is 0.6 rbe and the assumed size of the control logic is 130 rbe. The data array area of $N_{data\_row} \times N_{data\_col}$ size is

$$A_{data} = 0.6(N_{data\_row} + L_{sense\_amp})(N_{data\_col} + W_{driver})\text{rbe},$$

where L_{sense_amp} is the length of the bit-line sense amplifiers, and W_{driver} is the width of the driver. It is assumed that they are equal to 6 rbe respectively. In the tag array including status bits, the area for comparators, L_{comp}, will be added, which is assumed to be 6 rbe. Thus, the tag array area is

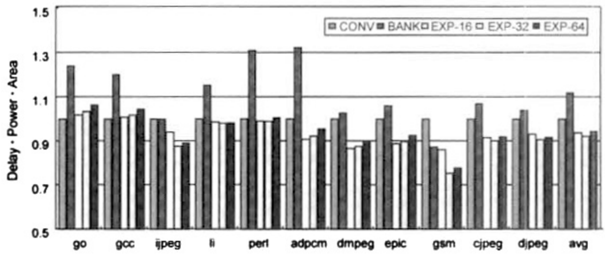
$$A_{tag} = 0.6(N_{tag\_rows} + L_{sense\_amp} + L_{comp})(N_{tag\_col} + N_{status\_col} + W_{driver})\text{rbe},$$

where N_{tag_col}, N_{atatus_col}, are the width of tag and status bits, respectively, and N_{tag_row} is the number of tags.

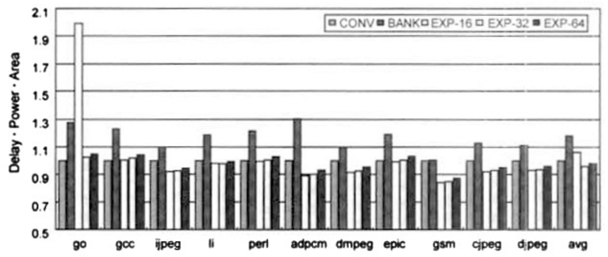
<Table 4> shows the chip areas for each I-cache model with 32-byte and 64-byte-block in rbe. In general, large block size causes the overall hardware complexities to be slightly increased, since a sense amplifier is suspended on each bit-line. As shown in <Table 4>, the increase of block size from 32bytes to 64bytes results in about an average 0.3% hardware cost increase. The hardware complexity of BANK is 1.4 times in a 32-byte cache block, and 2.7 times in a 64-byte cache block, as large as that of the CONV. This is due to the increase of the area for sense amplifiers, tag comparator and decoder. The increase of hardware complexity in the EXP is mainly due to the expansion buffer. The EXPs with 16, 32, and 64 buffer entries give cost increases of 1.7%, 2.9%, and 5.3%, respectively, compared to the CONV, while the BANK induces an average 2% cost increase.

〈Table 4〉 Chip areas for 32 byte-block and 64 byte-block I-caches

	CONV	BANK	EXP-16	EXP-32	EXP-64
32Bytes	99795	101177	101544	102728	105082
64Bytes	99891	102598	101624	102797	105132



(a) 32-byte cache block



(b) 64-byte cache block

(Figure 10) Evaluating I-cache systems using Delay · Power · Area metric

It is very difficult to establish one concise and clear metric, which combines all three factors of memory access latency, power consumption, and hardware cost. This is because the weight of each factor varies according to the processor design philosophy. In this study, it is supposed that all the factors have the same weighting. A new metric, Delay·Power·Area, which is the product of the normalized memory access latency, power consumption and chip area by those of the CONV, is presented to evaluate each I-cache model. As shown in (Figure 10), the EXP reduces the Delay·Power·Area in the CONV by about 7.8%, while the BANK increases it by 18%.

5. Conclusion

It is very important, with respect to power consumption and execution performance, to deliver instructions efficiently to the ILP processor. However, the variable length I-packet is an obstacle to the instruction supply for the VLIW processor. This is because a fetch of a straddle I-packet, which lies over two cache blocks due to the variable I-packet length, requires two cache accesses in the conventional I-cache. In order to solve this problem, the bank cache, which consists of two cache banks, stores instruction blocks by turn in each bank. It can remove the double accesses due to the straddle I-packet, but it requires an additional pipeline stage, which induces high misprediction penalty. In addition, the unnecessary cache accesses are increased by two-bank accesses and the delayed detection of misprediction.

In this paper, an expansion buffer cache is introduced to improve the instruction fetch bandwidth from the I-cache. It has a small expansion buffer, which contains a fraction of straddle I-packet, which, along with the main cache, avoids the sequential successive block accesses. A three-width expansion buffer can reduce the number of double accesses by an average 61%~85%, depending on the number of buffer entries in the four-issue VLIW machine. Thus, the expansion buffer cache achieves a reduction of 3%~8% in the memory access latency. In addition, the power consumption decreases in the expansion buffer cache because it decreases unnecessary cache accesses. In this study, the expansion cache buffer is measured with an integrated metric, Delay · Power · Area, to make an evaluation by combining three important factors : memory access latency, power dissipation, and hardware cost. Simulation results show that, compared with the direct-mapped and two-way set associative cache, the expansion buffer cache with 32-entry buffer makes 4%~8% and 17%~19% improvements to the Delay · Power · Area, respectively.

References

[1] D. H. Friendly, S. J. Patel and Y. N. Patt, "Alternative fetch and issue policies for the trace cache fetch mechanism," in Proc. of *Int'l Symp. on Microarchitecture*, pp.24-33, Dec., 1997.
[2] E. Rottenberg, S. Benett and J. E. Smith, "Trace cache : a low latency approach to high bandwidth instruction fetching," in Proc. of *Int'l Symp. on Microarchitecture*, pp.24-34, Dec., 1996.
[3] L. Geppert and T. Perry, "Transmeta's magic show," *IEEE Spectrum*, pp.26-33, May, 2000.
[4] P. Grun, N. Dutt and A. Nicolau, "Access Pattern Based Local Memory Customization for Low Power Embedded Systems," in Proc. of Design *Automation and Test in Europe Conference*, Mar., 2001.
[5] H. Michael and et al., "L1 data cache decomposition for energy efficiency," in Proc. of *Int. Symp. on Low Power Electronics and Design*, pp.10-15, 2001.
[6] E. Hao, P.-Y. Chang, M. Evers and Y. Patt, "Increasing the instruction fetch rate via block-structured instruction set architectures," in Proc. of *Int'l Symp. on Microarchitecture*, pp.191-200, Dec., 1996.
[7] A. Seznec, S. Jourdan, P. Sainrat and P. Michaud, "Multiple block ahead branch predictor," in Proc. of *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 1996.
[8] A. Klauser and D. Grunwald, "Instruction fetch mechanisms for multipath execution processors," in Proc. of *Int'l Symp. on Microarchitecture*, pp.38-47, Dec., 1999.
[9] T. M. Conte and et al., "Instruction fetch mechanisms for VLIW architectures with compressed encodings," in Proc. of *Int'l Symp. on Microarchitecture*, pp.201- 211, Dec., 1996.
[10] N. S. Kim and et al., "Leakage Current : Moore's Law Meets Static Power," *IEEE Computer*, pp.68-75, Dec., 2003.
[11] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu, "IMPACT : An architectural framework for multiple-instruction-issue processors," in Proc. of *Int'l Symp. on Computer Architecture*, pp.266-275, May, 1991.
[12] T. Y. Yeh and Y. N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," in Proc. of *Int'l. Symp. on Computer Architecture*, pp. 257-266, 1993.
[13] T. Conte, K. Menezes, P. Millis and B. Patell, "Optimization of instruction fetch mechanism for high issue rates," in Proc. of *Int'l Symp. on Computer Architecture*, pp.333-344, June, 1995.
[14] S. Banerjia, K. N. Menzes and T. M. Conte, "NextPC computation for a banked instruction cache for a VLIW architecture with a compressed encoding," Technical Report. Dept. of Electrical and Computer Engineering, North Carolina State University, June, 1996.
[15] M. B. Kamble and K. Ghose, "Energy-efficiency of VLSI cache : a comparative study," in Proc. of *Int'l Conf. on VLSI Design*, pp.261-267, Jan., 1997.
[16] J. M. Mulder, N. T. Quach and M. J. Flynn, "An area model for on-chip memories and its application," *IEEE Journal of Solid-State Circuits*, Vol.26, No.2, pp.98-106, Feb., 1991.
[17] M. Horowitz, T. Indermaur and R. Gonzalez, "Low-power digital design," in Proc. of *IEEE Symp. Low Power Electron*, pp.8-11, Oct., 1994.
[18] W. Tang, R. Gupta and A. Nicolau, "Power savings in embedded processors through decode filter cache," in Proc. of *Int. Conf. on Design Automation & Test in Europe*, pp.443-448, Mar., 2002.

홍 원 기
e-mail : wkhong@daegu.ac.kr
1995년 연세대학교 전산과학과(학사)
1997년 연세대학교 컴퓨터과학과(석사)
2001년 연세대학교 컴퓨터과학과(박사)
2001년~2002년 UC Irvine (미) 박사후과정
2002년~2004년 LG 전자 디지털 미디어 연구소 선임연구원
2004년~현재 대구대학교 정보통신공학부 전임강사
관심분야 : 임베디드 시스템, 실시간 시스템, 무선 센서 네트워크 등