Value Compression to Reduce Power in Data Caches

Carles Aliagas[1], Carlos Molina[1], Montse Garcia[1], Antonio Gonzalez[2,3], and Jordi Tubella[2]

[1] Departament d'Enginyeria Informàtica i Matemàtiques,
Universitat Rovira i Virgili. Tarragona-SPAIN
{caliagas, cmolina, mgarciaf}@etse.urv.es
[2] Departament d'Arquitectura de Computadors,
Universitat Politécnica de Catalunya. Barcelona-SPAIN
{antonio,jordit}@ac.upc.es
[3] Intel Barcelona Research Center,
Intel Labs-Univesitat Politècnica de Catalunya. Barcelona-SPAIN

Abstract. Cache memory represents an important percentage of the total energy consumption of today's processors. This paper proposes a novel cache design based on data compression to reduce the energy consumed by the data cache. The new scheme stores the same amount of information as a conventional cache but in a smaller physical storage. At the same time, the cache latency is preserved, thus no performance penalty is introduced. The benefits of energy vary from 15.5% to 16%, and the reduction in die area ranges from 23% to 40%.

1 Introduction

Caches occupy a very important part of die area in most processors. Percentages of around 50% are relatively common. On the other hand, some authors have reported that caches may be responsible for 10%-20% of total energy consumed by a processor.

The objectives of this work are twofold: to reduce the area of the cache and its energy consumption while maintaining the same amount of information. We present a novel cache architecture that provides significant advantages in terms of these metrics.

The key idea is based on the observation that many values stored in the data cache can be represented with a small number of bits instead of the full 32-bit or 64-bit common representation. Using less bits to store the data, together with a simple encoding/decoding scheme, allows the processor to reduce the storage required for a given amount of data, and thus, reduces the area and energy consumption of the memory.

The rest of the paper is organized as follows. Section 2 discusses and evaluates schemes for compressing data values. The proposed cache architecture is presented in section 3 and evaluated in section 4. Section 5 outlines some related work and finally, section 6 summarizes the main conclusions of this work.

A detailed description of section 2 and 3 is in the following technical report *http://www.ac.upc.es/pub/reports/DAC/2003/UPC-DAC-2003-10.ps.Z*

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 616–622, 2003.

2 Data Value Compression

Data values have significant redundancy in their representation. In this section we quantify this phenomenon and discuss some different types of data compression that can be used to take advantage of this redundancy.

In general, we focus on 64-bit and 32-bit wide data values. This allows for a huge rang of values to be represented. However, the significant bits of these values are relatively few. We avoid to store non-significant bits using: integer data compression and address data compression.

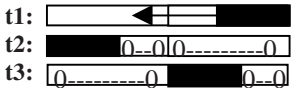

Fig. 1. Integer data compression

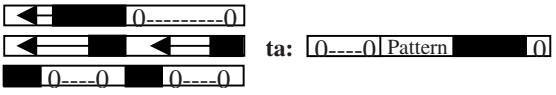

Fig. 2. Address data compression

Integer data compression is applied when the data can be represented with a few bits and the rest of the bits are all ones or zeros. Figure 1 shows the different types of data compression applied to 64 bits. We will only store in the cache the *black* part (significant bits). The arrow indicates sign extension, and the "0..0" means that the field is all zeros.

Another type of data are addresses stored in memory. In this case, we apply address data compression. There are three sources of addresses: data pointers, code pointers and stack pointers. The compiler distributes the memory logical space in three major components: code, data and stack. These addresses are 64-bit wide, but only a part of their bits is variable. The rest of bits remain unaltered and are common to all addresses. We try to recognize the pattern of the common part and store only the variable part. Figure 2 depicts the components of the address data compression. We have experimentally observed that with 8 patterns we can represent 83% of total value addresses.

These 7 types of data compression allow us to compress data form 64 to 22 bits. This can be done for 70% of all data stored in a 16KB data cache, being *type 1*(t1) and *type address*(ta) the most significant. (see framework described in section 4)

3 Pattern Cache

The Pattern Cache is proposed to store data values in a compressed form. Thus the proposed cache architecture reduce the storage area and energy consumption.

The Pattern Cache is divided into two areas: the tag and the data areas. The tag area or Location Table (LT) stores compressed values and pointers to the data area or Value Table (VT). LT is indexed as a conventional data cache. Each LT entry has two fields: a tag that identifies the memory address stored in each line, as in a usual cache, and the location field that determines for each 64-bit word in the line where and how

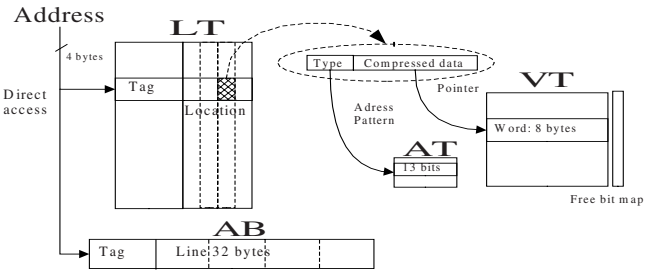

Fig. 3. Pattern cache block diagram

to find its value. Each location field has two parts: type and compressed data. The first one determines how to find the value and the second one is a compressed data or a pointer to VT (Table 1 summarizes the different types of compression). On the other hand, a VT entry has a full 64-bit data value.

Table 1. Compression Type Codification

Type	Information	Type	Information	Type	Information	Type	Information
0000	Pointer	0100	Data: type4	1000	Address: prefix0	1100	Address: prefix4
0001	Data: type1	0101	Data: type5	1001	Address: prefix1	1101	Address: prefix5
0010	Data: type2	0110	Data: type6	1010	Address: prefix2	1110	Address: prefix6
0011	Data: type3	0111	Not used	1011	Address: prefix3	1111	Address: prefix7

There are three ways of obtaining the data values: a) a value may be stored in the LT entry itself using an integer compressed form; b) a value may be stored partly in the LT entry and partly in the Address Table (AT) using an address compressed form; and c) the value may be stored in the Value Table (VT) in a non-compressed form.
The behavior of the Pattern Cache may be summarized as follows:

1. Miss on Read or Write: the new line from memory is stored in LT + VT or the Assisting Buffer (AB). Those 64-bits words that can be compressed are stored directly in LT, the others are stored in VT. The AB store the full line when one of 64-bits words cannot be store in LT-VT. The AB also works as a victim cache for lines replaced from the LT [7]
2. Hit on read: if there is a hit in LT, the location field provides either the value or a pointer to the VT. If the value is compressed it is necessary to restore the full value. With Integer Data Compression (Figure 1) only sign extension and zero concatenation is necessary. With Address Data Compression it is necessary to access the Address Table (AT in Figure 3) to obtain the pattern to restore the full value.
3. Hit on write: there are four cases: a) The replaced value is in the VT and the new value cannot be compressed, the latter is stored in the location of the former. b) The replaced value is in the VT and the new value can be compressed, the later is stored in the LT and the VT entry is invalidated. c) The replaced value is in the LT and the new value can be compressed, the later is stored in the location of the former. d) The replaced value is in the LT and the new value cannot be compressed, the value

is tried to be stored in an empty entry of the VT. If there is no space in the VT, all the line is moved to the AB and the entries in the LT and the VT are invalidated.

The compression test requires very few hardware effort and only on Miss or Write. For the integer type compression it is necessary an AND gate to detect a set of ones and an OR gate to detect a set of zeros. For address data compression, an AND gate with some entries inverted is needed for each different pattern. We consider those gates and the AT table (8 entries of 13 bits) negligible in terms of area and energy consumption.

4 Performance and Evaluation

4.1 Static Analysis

This subsection presents an evaluation of the cache area, access time and energy consumption of the Pattern Cache. These evaluations are done by means of the CACTI tool version 3.0 [9]. The input parameters of the CACTI tool have been adapted to model the structure of the Pattern Cache. The assumed technology is 0.09µm.

The baseline cache is a 16KB direct-mapped cache with 32-bytes line size, with a Victim Cache(VC) of 16 entries full associative. For the Pattern cache: the LT has the same number of lines and associativity as the base cache. Both tags are identical. The location field has 22 bits for each word in a line. And the VT is managed as a direct mapped cache with 8-byte line size. The AB has the same structure of the VC. *VTxx* means that the VT capacity has been reduced to xx% of the baseline size.

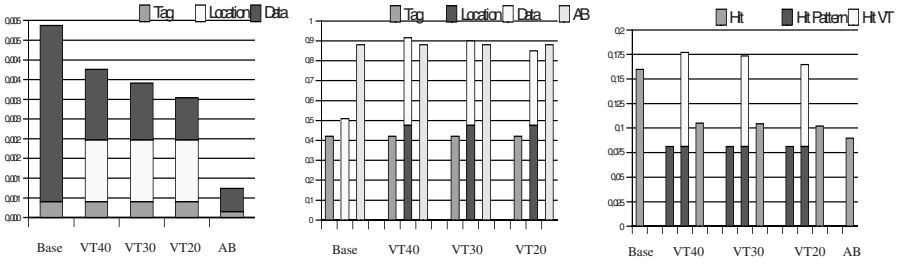

Fig. 4. (a) Die area (cm2) (b) Access time(ns) (c) Energy consumption (nJ)

Figure 4.a shows the total die area. Different colors in each bar represent the contribution of *tags*, *location field* and *data*. The rightmost bar represents the area of the VC o AB, that has to be added to all cache configurations, including the baseline. As expected, the achieved area reduction is significant in all cases. For instance, VT30 achieves a reduction of 26% with respect to the baseline cache. Below, it is discussed how these reductions interact with the miss ratio.

The access time on Hit is shown in Figure 4.b. Notice that, every cache configuration has three bars. First bar determines the time required for tag check. Second bar determines the time required to access the value (in a compressed form o through VT). Finally, last bar determines the access time to the VC or AB. In order to determine the total access time, the maximum of those three bars has to be considered. The main

conclusion of this study is that the AB determines the access time. This is only true for caches of less than 128KB.

Finally Figure 4.c shows the energy consumption for a cache hit. In particular, the *Hit pattern* corresponds to the energy consumed by an access in the Pattern Cache that hits in LT. *Hit VT* corresponds to the energy consumed by the Pattern Cache when the value is obtained from VT (LT + VT). *Hit* is the average consumption of the Pattern Cache considering the percentage of *Hit pattern* and *Hit VT* during the execution of programs. Simulations have shown that on average, 75% of the time the value is obtained from LT.

Statistics for cache capacities ranging from 4KB to 64 KB are detailed in Table 2.

4.2 Dynamic Analysis

The simulation environment is built on top of the Simplescalar [4] Alpha toolkit that has been modified to model the Pattern Cache.

The following Spec2000 benchmarks have been considered: crafty, eon, gcc, gzip, mcf, parser, twolf, vortex and vpr from the integer suite; and ammp, apsi, art, equake, mesa, mgrid, sixtrack, swim and wupwise from the FP suite. The programs have been compiled with maximum level of optimization. The reference input set was used and statistics were collected for 1000 million of instructions after skipping the initializations.

4.3 Analysis of Results

This subsection presents and analyzes the results obtained for different memory level hierarchy configurations in terms of die area, energy consumption and miss ratio.

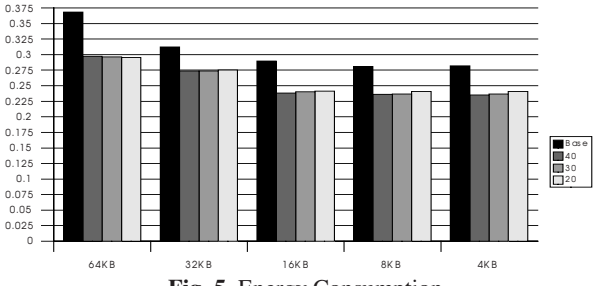

Fig. 5. Energy Consumption

Miss Rate vs. Power Consumption

In order to determine the performance benefits of the cache design proposed in this paper, the scenario described in previous subsection is considered. The size of cache varies from 4KB to 64KB.

Figure 5. shows the energy consumption of the memory hierarchy. Each bar denotes the average energy consumed per access for different cache configurations, which is obtained through the following formula:

$$EC = (Hit\_ratio * DL1\_EC) + (Miss\_ratio * (2 * DL1\_EC + DL2\_EC)) \quad (1)$$

DL1_EC is the average energy consumed by an access to the first level data cache. DL2_EC is the energy consumed by an access to the second level cache. This cache is assumed to be 512KB. Note that this expression computes the total energy by considering the consumption of the first level data cache on hit, and the consumption of the first and the second level data cache on miss. A miss produces two accesses to DL1 (one for detecting the miss, and another for storing the values) and one access to DL2 to obtain the data.

As expected, the Pattern Cache outperforms the base model in terms of energy consumption. The main contribution to this reduction is due to the LT, which provides the data in 75% of the hits. Note also that the VT reduction hardly affects power dissipation since decreasing the VT size increases miss ratio and more data must be provided by the second level cache.

From these numbers and the results in the previous section, we can conclude that the Pattern Cache is an effective architecture for first level data caches in terms of power, die area and access time. Table 2 summarizes all the statistics for each cache configuration being considered. For instance, a reduction of VT to 40% produces an average die area reduction of 19%, a energy consumption reduction of 16% and a very minor (1.5%) increase in the miss ratio.

Table 2. Statistics for different capacities

	Die Area Reduction			Energy Consumption Reduction			Miss Ratio Increment		
Cache	VT40	VT30	VT20	VT40	VT30	VT20	VT40	VT30	VT20
4KB	16%	18%	27%	16.5%	16%	14.5%	1.2%	2.4%	8.6%
8KB	22%	26%	36%	16%	15.5%	14%	1.4%	2.8%	10.0%
16KB	20%	26%	33%	18%	17%	16.5%	1.6%	3.2%	11.0%
32KB	17%	27%	35%	12.5%	12%	12%	1.7%	5.2%	12.2%
64KB	20%	28%	36%	19%	19.5%	20%	1.8%	5.5%	12.9%
AMEAN	19%	25%	33%	16%	16%	15.5%	1.5%	3.8%	10.9%

5 Related Work

Zhang et al. [13] propose the design of the FCV (Frequent Value Cache). The FVC only contains frequently accessed values stored in a compact encoded form and it is used in conjunction of a traditional Direct-Mapped Cache. Yang et al [12] present a similar cache design and evaluation called CC (Compression Cache) where each line can hold one uncompressed line or two cache lines compressed to at least half. A modification of the FVC is proposed in [11] in order to improve energy efficiency.

Significance compression is used by Brooks et al. [3] and Canal et al. [5] to reduce power dissipation, not only in data cache but in the full pipeline. Brooks et al. present a mechanism that dynamically recognizes whether the most significant bits of values are just sign extension and in this case, the functional units operate just on the least significant bits. Canal et al. propose a new significance compression scheme that ap-

pends two or three extension bits to each data value to indicate the significant byte positions. Other different data compression are also presented in [1],[8] and [10].

6 Conclusions

We have introduced a novel data cache architecture called Pattern Cache. The proposed cache applies different types of compression to values in order to reduce the required storage. Simulation results show that in a data cache, close to 70% of the values stored in 64-bit may be compressed to 22 bits. We have shown that the Pattern Cache significantly reduces power dissipation and die area with a minimum loss in terms of miss ratio while maintaining the same access time as a conventional cache.

References

1. B. Black, B. Rychlik, and J. Shen, "The Block-Based Trace Cache". In Proc. of the 26th ISCA, 1999
2. D. Brooks, V. Tiwari and M. Martonosi, "Watch: A Framework for Architectural-Level Power Analysis and Optimization". In Proc. of the 27th ISCA, 2000
3. D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance", In Proc. of the 5th Int. Sym. on HPCA. 1999
4. D. Burger, T.M. Austin and S. Bennet, "Evaluating Future Microprocessors: The SimpleScalar Tool Set". Technical Report CS-TR-96-1308. University of Wisconsin, 1996
5. R. Canal, A. González, and J.E. Smith, "Very Low Power Pipelines using Significance Compression". In Proc. of the 33th Annual Int. Sym. on Microarchitecture, 2000
6. M.Gowan, L.Brio and D. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor". In Proc. of the 35th Design Automation Conference,1998
7. N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associate Cache and Prefetch Buffers". In Proc. of the 17th ISCA, 1990
8. S. Y. Larin, "Exploiting Program Redundancy to Improve Performance, Cost and Power Consumption in Embedded Systems", Ph.D. Thesis, ECE, North Carolina State U., 2000
9. P. Shivakumar, N. P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power and Area Model". Western Research Lab (WRL), Research Report 2001/2
10. L. Villa, M. Zhang, and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction". In Proc. of the 33th Annual Int. Sym. on Microarchitecture, 2000
11. J.Yang, and R. Gupta, "Energy Efficient Frequent Value Data Cache Design", In Proc. of the 35th Annual Int. Sym. on Microarchitecture, 2002
12. J.Yang, Y. Zhang and R. Gupta, "Frequent Value Compression in Data Caches". In Proceedings of the 33th Annual International Symposium on Microarchitecture, 2000
13. Y. Zhang, J. Yang and R. Gupta, "Frequent Value Locality and Value-Centric Data Cache Design". In Proc. of the 9th Annual Int.Conference on ASPLOS, 2000