Non Redundant Data Cache

Carlos Molina[Ψ], Carles Aliagas[Ψ], Montse García[Ψ], Antonio González[ϕλ], and Jordi Tubella[ϕ]

[Ψ] Dept. Eng. Infomàtica i Matemàtiques
Universitat Rovira i Virgili
Tarragona - SPAIN

[ϕ] Dept. d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Barcelona - SPAIN

[λ]Intel Barcelona Research Center
Intel Labs-Univ. Politècnica de Catalunya
Barcelona - SPAIN

E-mail: cmolina@etse.urv.es, caliagas@etse.urv.es, mgarciaf@etse.urv.es, antonio@ac.upc.es, jordit@ac.upc.es

Abstract

Current microprocessors spend a huge percentage of the die area to implement the memory hierarchy. Moreover, cache memory is responsible for a significant percentage of the total energy consumption. This paper presents a novel data cache design to reduce its die area, power dissipation and latency. The new scheme, called Non Redundant Cache (NRC), exploits the immense amount of value replication observed in traditional data caches. The NRC cache significantly reduces the storage requirements by avoiding the replication of values. Results show that the NRC cache reduces the die area in a 32%, the power dissipation by 14% and the latency by 25%, while maintaining the miss ratio of a conventional cache.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles - *cache memories*.

General Terms

Design and Performance.

Keywords

Value Replication, Cache Memories, Compression, Low Power.

1. Introduction

As the scale of integration continues to grow and applications use larger working sets, on-chip cache memories of microprocessor become larger. On average, caches spend close to 50% of total die area [6]. Moreover, chip multiprocessor designs are a promising approach for increasing the throughput, but both on-chip memories and processing cores compete for the die area an the area occupied by one affects the amount left for the other [7]. On the other hand, some authors [2] [5] have reported that caches may be responsible for 10% to 20% of total power dissipated by a processor. In processor design several trade-offs are necessary for obtaining a good balance between cost and performance. For high productions volumes, cost can be associated with area chip, so a way to reduce the cost is to reduce area requirements. On the other hand, power dissipation is becoming a critical issue for microprocessors. Power dissipation determines the cost of the cooling system and ultimately it may limit the performance of the microprocessor. Dynamic power dissipation of on-chip memories is strongly related to its area, whereas static power dissipation depends on the number of transistors. In this paper, we present a novel cache design, the Non Redundant Cache, that reduces the storage requirements of data cache by offending the significant amount of replication that is present in conventional cache designs. This storage savings result in significant die area savings, power reduction and latency decrease.

The rest of the paper is organized as follows. Section 2 evaluates the value replication in conventional data caches. 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.

2. Data Value Replication

Several studies [8],[9],[11],[15] have pointed out that value replication is common in different parts of a processor. In this work, we focus on value replication in the data caches during the execution of programs. Different configurations of first- and second-level data cache memories have been simulated with sizes ranging from 1KB to 4 MB, and different degrees of associativity. The conclusion of all these simulations is that there is significant value replication on data caches at any given instant of time. It is observed that the percentage of value replication increases when caches become bigger. It is also observed that value replication degree is not affected by associativity. For instance, Figure 1 shows the value replication of a 256 KB direct-mapped Level 2 data cache. The simulation environment and the list of benchmarks is detailed in section 4.2

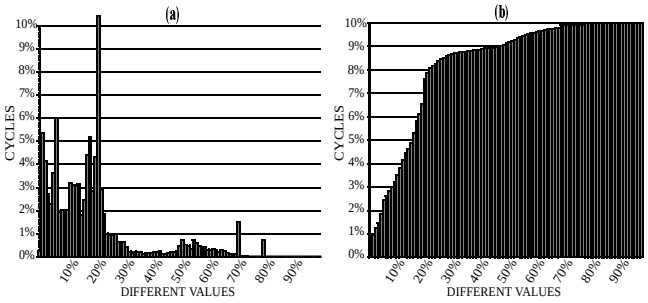

Figure 1. Average Histograms of a 256KB Level 2 Data Cache

To obtain this data, the content of the data cache is analyzed every cycle and the percentage of replicated 64-bit values is obtained. The X-axis represents the percentage of different values (0% means that all the values in cache are the same and 100% means that all the values in cache are different). The Y-axis represents the percentage from the total execution time that the corresponding percentage of different values has been observed. Figure 1.a is the histogram of the degree of variability whereas Figure 1.b shows the accumulated distribution.

It can be observed that a huge degree of value replication exists. For instance, Figure 1.b shows that during 80% of the execution time, less than 25% values of the cache are different. This means that on average, a value is stored four times. Note also that this cache has never more than 80% of different values. Section 4.2.1 analyzes in more detail the percentage of value replication that can be achieved for individual benchmarks under different scenarios. Based on the observation that many values stored in the data cache are repeated, a novel cache design is proposed and evaluated. The study concentrates on second-level data caches since they occupy a very important part of the die area in most processors.

3. Non Redundant Cache

This section outlines the main characteristics of the proposed cache. First, a general description is presented and finally, a simple encoding/ decoding scheme that may be applied to the cache is also discussed.

3.1. General Description

The underlying concept of our proposal is based on the observation that data caches exhibit a high percentage of value replication at any given instant of time. In this way, a new cache architecture that we refer to as Non Redundant Cache is proposed in order to reduce storage area, power dissipation and latency. Figure 2 depicts the proposed design.

The Non Redundant Cache is divided into two areas: the tag and the data areas. The tag area or Pointer Table (PT) stores pointers to the data area or Value Table (VT). PT is indexed as a conventional data cache. Each PT entry has two fields: a *tag* that identifies the memory address stored in each line, as in a conventional cache, and the *pointer* field that determines for each 64-bit word in the line, the entry of VT where the value may be found. Note that additional storage is required to

ISLPED'03, August 25-27, 2003, Seoul, Korea.

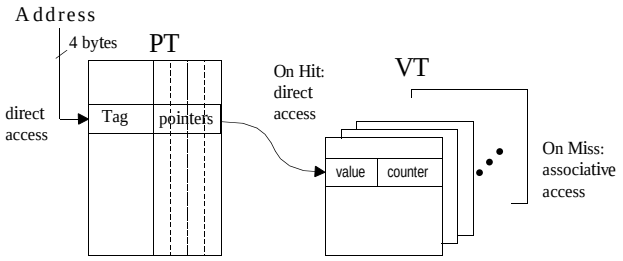

Figure 2. Block Diagram of the Non Redundant Cache

maintain pointers but on the other hands, values are not replicated in the VT. Thus, any word of any cache line of PT that has the same value, points to the same location in VT. The location of a value in the VT is determined by the value itself, i.e., the VT is indexed by values. Each VT entry has also two fields: a full 64-bit data value and a counter. This counter maintains the number of pointers from PT that points to that value. When the counter reaches its maximum value, no further lines are allowed to point to this VT entry. If a new line needs to point this value, the value is replicated and stored in a new VT entry with its counter initialized to 1. We have experimentally confirmed that using 4-bit saturating counters less than 1% of the values of the VT have some replica (note that narrow values, which are some of the most repeated ones, are not stored in the VT but they are stored in the PT, as described in the next section).

The behavior of the Non Redundant Cache may be summarized as follows:

1. Miss on Read or Write: the new line is brought from the upper level of the memory hierarchy. At this time, all the values are tried to be stored in the VT. Finally, PT has to be updated with the new tag and pointers.
   The search for room in the VT is simple and may be summarized as follows. First, it is performed a value search in the VT. If the value is found, the associated counter is increased and that entry will determine the content of the pointer in the PT. If the value does not exist, a free entry is required. A VT entry is considered as free if its associated counter is zero. If no free entry is available, the value cannot be stored in the VT.
   Replacement in the PT is implemented in the same way as in a conventional cache. When replacing a line from the PT, its associated pointers are used to decrement the counters of the corresponding values.
2. Hit on read: if there is a hit in the PT, the associated pointer provides the index to the VT where the value will be found.
3. Hit on write: the new value is searched in VT as explained above and a new VT entry is allocated if it is not found. The counter of the old value is decreased.

Note that a line in the PT may not have all its pointers valid. An invalid pointer is produced when its associated value cannot be stored in the VT. The indexing function for the VT is an important parameter for the performance of the Non Redundant Cache. When a new entry of VT is required, the least significant bits of the value (ignoring the least significant two) determines the set. In that case, associativity of 8 is considered, in order to reduce aliasing. This configuration works fairly well but is still far from the behavior of the full associative approach. It is beyond the scope of this paper the study of better indexing functions.

Note finally that the VT does not need to store all the bits of a value. The least significant bits are implicitly given by the set that the value occupies (in the same way as the tags of a cache do not contain the least significant bits).

3.2. Data Value Inlining

Since the tag area of the Non Redundant Cache provides some storage for pointers, a further enhancement to the Non Redundant Cache can be applied. In particular, narrow values (i.e., values that can be represented with a small number of bits) can be inlined into the tag area. The idea is that if a value can be represented in the number of bits allocated for a pointer in the PT, instead of storing the value in the VT and set up a pointer from the PT to the VT, the value can be directly stored in the pointer location itself. Narrow values are very common [4], [12] so this simple extension may provide significant benefits. We will refer to a value stored in the PT as a *narrow value*.

This improvement does not only enlarge the logical capacity of VT. It also reduces latency and power dissipation because the access to the VT is avoided for narrow values.

The extensions to support narrow values are quite simple. Each value

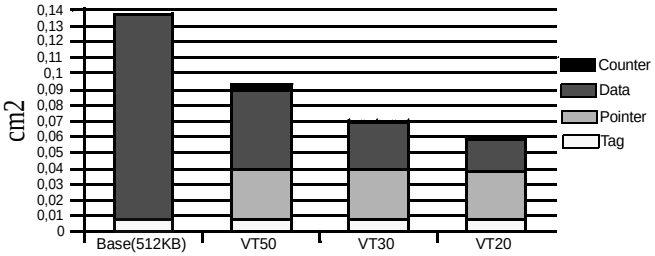

Figure 3. Cache area

of a new line brought on a cache missed is tested. This basically consists of an OR and an AND gate applied to the most significant bits to check whether they are all ones or all zeroes. If a value is narrow, there is not need to search for store in the VT; it is stored in the corresponding pointer field. A bit in each pointer field is devoted to indicate whether its content is a pointer or a narrow value. The search for room in the VT is always done after a Miss. For each value in the line a narrow test is performed. If the value is in the selected range, there is no need for a search in the VT. Value is managed as narrow and compressed in the pointer field of the PT, setting an additional narrow bit. On the other hand, if there is a hit on the PT and the value is narrow, the value is provided by the PT and the access to the VT is avoided. The value obtained from the PT is sign-extended before being sent to the processor data path.

Finally, note that in case of a miss, if none of values of the new line can be either inlined or stored in the VT because of the lack of space, the line is just not stored in the Non Redundant Cache.

4. Performance Evaluation

4.1. Static Analysis

4.1.1 Die Area

This subsection presents an evaluation of the cache area, latency and power dissipation of the Non Redundant Cache. For these evaluations the CACTI tool version 3.0 [10] is used, with the appropriate extensions for the new structures. The assumed technology is 0.09µm. The evaluation focuses on the tag and data blocks, since they represent the vast majority of the cache area. For instance, the total area of a 512KB direct cache with 32 bytes per line is: 0.1386 cm^2 whereas the data and tags occupy 0.1376 cm^2 (99.3%).

The PT has the same number of lines and associativity as the conventional cache used as baseline. Note that the PT and the baseline cache have the same tag area, but the PT also stores the pointer fields. The number of bits in a pointer field depends on the size of the VT. On the other hand, the VT is managed as a table when accessed through a pointer of the PT (i.e., the pointer indicates the precise location of the value), and it is indexed as a set-associative cache when new values have to be stored.

Figure 3 shows the total die area for different cache configurations for a 32-byte cache line size. The baseline considered corresponds to a 512KB L2 data cache. Several Non Redundant Cache configurations are depicted. VTxx means a Non Redundant Cache with a VT capacity reduced to xx% of the baseline size (e.g., VT30 means a VT capacity reduced to 30%). Different colors in each bar represent the contribution of tags, pointers and data. As expected, the achieved area reduction is significant in all cases. For instance, VT30 achieves a reduction of 49% with respect to the 512 KB baseline cache. Below, it is discussed how these reductions interact with the miss ratio. Statistics for other cache capacities ranging from 256KB to 4MB are detailed in Table 1.

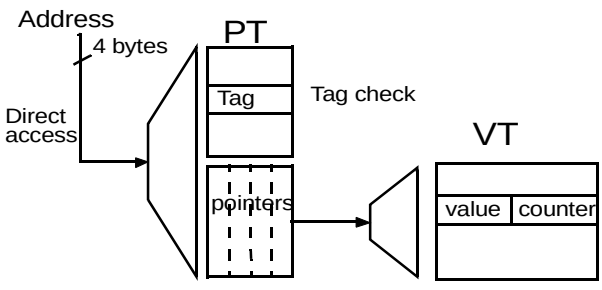

Figure 4. Non Redundant Cache critical path

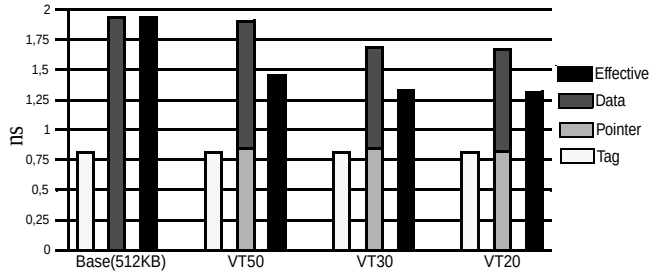

Figure 5. Latency

4.1.2. Latency

The access time of the various cache architectures was also evaluated with the CACTI tool. Figure 4 depicts the critical path of an access to the Non Redundant Cache.

When an address is sent to the Non Redundant Cache, a direct-mapped access (other indexing functions are also possible) is performed to the PT, which provides the tag and the pointers at the same time. The corresponding pointer is selected and then used to access the VT. The tag comparison is done in parallel with the access to the VT.

If there is a miss, an access to the next level of memory is started. As the values of the line arrive from the next memory level, their corresponding ranges are checked and depending on that, each value is decided to be stored in the VT or just inlined in the PT. If the cache line is stored in the Non Redundant Cache, a line from the Non Redundant Cache is replaced. The replaced entry in PT is read and all their valid pointers determine the counters to be decreased in the VT.

The behavior of the Non Redundant Cache for a hit may be summarized as follows: (a) The PT is accessed and the corresponding tag and pointers are checked. (b) If the value is not inlined, an access to VT is performed. Total access time is the maximum of tag access and comparison or the pointer access and the VT access.

Figure 5 shows the different time components of a cache access. The same configurations as in Figure 3 have been considered. Three bars are shown for each configuration. The first bar depicts the time required for the tag check. The second bar determines the time required for accessing the value. Note that this bar is split into two components for the Non Redundant Cache: pointer access and VT access. Finally, the third bar represents the effective data access time, considering the percentage of accesses (see Figure 8) that are satisfied by the PT and the percentage that requires an access to the VT.

The latency of the Non Redundant Cache is reduced significantly as the size of the VT decreases. Overall, the Non Redundant Cache offers a much shorter latency than a conventional organization. For instance, VT30 achieves a reduction in access time of 49% with respect to the 512 KB baseline cache. The latency advantage of the Non Redundant cache is even greater for larger cache sizes as it is reported below.

4.1.3. Energy Consumption

Energy consumption has also been evaluated by extending the CACTI tool for the Non Redundant Cache. The Non Redundant Cache provides significant benefits in terms of energy consumption since the energy consumption of memory structures depends on their area and the total number of bits read/written, among other things. Figure 6 shows the energy consumed per each access for different types of accesses. *Hit inlined* corresponds to an access that finds the value inlined in the pointer field. *Hit VT* corresponds to an access for which the value is obtained from the VT. This involves an access to the PT plus an access to the VT. *Hit* is the average energy per access for hits, considering the percentage of accesses that hit in the pointer field and the percentage that hit in the VT. *Miss* corresponds to an access that

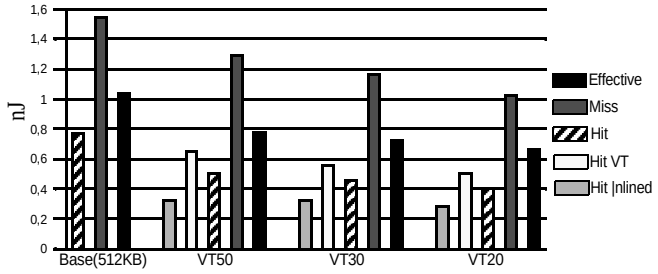

Figure 6. Energy consumption per access

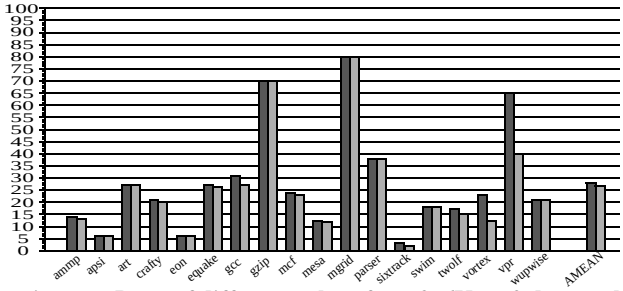

Figure 7. Perc. of different values for a 256KB L2 data cache

misses in the Non Redundant Cache and has to be served by the next level of memory. This involves two accesses to the PT (first access determines a miss, and second access updates the PT pointers) and a number of accesses to the VT that depends on the number of values that can be inlined (which have been obtained through simulation - see Figure 8 below). Finally, *Effective* shows the average energy consumed by access. Savings for caches ranging from 256KB to 4MB are presented below in Table 1. Note that the reduction in power is very significant across the whole range of capacities.

4.2. Dynamic Analysis

The simulation environment is built on top of the Simplescalar [3] Alpha toolkit, which has been modified to model the Non Redundant 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 the Compaq C compiler with -non_shared -O5 optimization flags (i.e, maximum optimization). Each program was run with the reference input set and statistics were collected for 1000 million of instructions after skipping an initial part of initializations.

4.2.1. Replication in Conventional L2 Caches

In this subsection, the degree of value replication in conventional L2 caches is studied. A 256 KB second level data cache has been considered.

Figure 7 shows the percentage of values that are different in a conventional organization. Two bars are depicted for each program. The first bar corresponds to the percentage of different values observed during 99% of the execution time. The remaining 1% is not considered to disregard the effect of rare cases that would have little influence in the total execution time. The second bar reports the percentage of values that are different and cannot be inlined. A value is considered to be able to be inlined if it can be represented with no more than 10 bits.

A significant degree of value replication is observed for most benchmarks. For the majority of benchmarks, less than 25% of the values of the L2 cache are different. *Gzip*, *mgrid* and *vpr* are the programs with the lowest degree of replication although it is not negligible at all. On average, only 28% of the values are different, which can be translated in significant gains for the Non Redundant Cache. Benefits are even greater for larger cache capacities as reported below. For instance, for a 4 MB data cache only 15% of the values are different.

4.2.2. Inlining Performance

Another result of Figure 7 is that inlining produces a slight reduction in the storage required by values that are different (i.e., the storage required in the VT of the Non Redundant Cache) for integer benchmarks, while its effect is almost null for FP programs. However, inlining has another important benefit. Since in general the inlined values are those that have the highest degree of repeatability (e.g., 0, 1

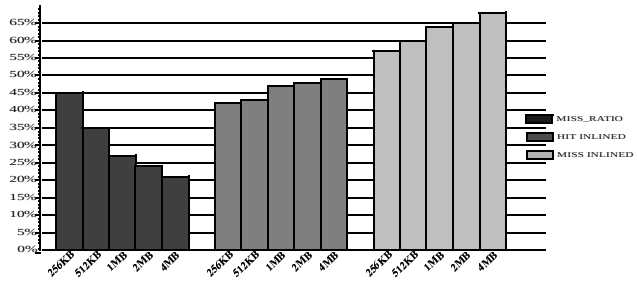

Figure 8. Miss ratio, hit inlined and miss inlined

	Die Area Reduction			Energy Consumption Reduction			Access Time Reduction			Number of Misses Increment		
Cache	VT50	VT30	VT20	VT50	VT30	VT20	VT50	VT30	VT20	VT50	VT30	VT20
256KB	32%	47%	54%	13%	18%	23%	8%	17%	22%	2.8%	7.7%	11.5%
512KB	33%	49%	57%	25%	30%	37%	25%	31%	32%	4.3%	11.6%	16.5%
1MB	31%	46%	55%	19%	26%	33%	25%	30%	30%	6.8%	14.4%	20.5%
2MB	33%	46%	55%	12%	18%	23%	38%	47%	51%	6.5%	14.8%	21.8%
4MB	31%	48%	55%	0%	14%	18%	32%	42%	48%	5.1%	11.8%	19.5%
AMEAN	32%	47.2%	55.2%	13.8%	21.2%	26.8%	25.6%	33.4%	36.6%	5.1%	12%	17.9%

Table 1. Comparison results

and -1 are usually the most frequent values) [15], inlining these values allow for the use of a very small counter in the VT table (see section 3.1).

Figure 8 presents further statistics for caches ranging from 256KB to 4MB. The dark grey bars show the miss ratio. The medium grey bars depict the hit ratio to inlined values. Note that the percentage increases as caches become bigger as the range of inlined values increases, since the pointers in the PT require more bits to point to a bigger VT. On average, between 42% and 49% of the memory requests do not need to access the VT. Finally, the light grey bars show the percentage of values that are brought from the next memory level on a miss and can be inlined. On average about two thirds of the values can be inlined and this percentage increases as caches becomes bigger. These clearly points out that inlining is a very effective technique.

4.2.3. Miss Rate vs. Die Area

In order to determine the performance potential of the Non Redundant Cache, the following scenario is considered: a second level data cache ranging from 256KB to 4MB. All cache configurations are considered to be direct-mapped with a cache line of 32 bytes. The PT is dimensioned to store the same number of lines as the base configurations but reducing the VT to 50%, 30% and 20% of the original size. The VT is dimensioned to store values of 8 bytes, so there are 4 pointers in each entry of the PT, in addition to the tag. Each line of the VT stores an 8-byte value and a 3-bit as counter that determines the number of pointers from the PT that refer to that value.

Table 1 summarizes the main statistics for each cache configuration being considered. The results show that the Non Redundant Cache is a very effective design for the second level cache. For instance, a configuration with a VT with 50% of the capacity of the baseline results in an average die area reduction of 32%, a energy consumption reduction of 13.8%, an access time reduction of 25.6%, and a very minor increase (5.1%) in number of misses. The increment in misses reported in Table 1 for VT30 and VT20 is only due to three benchmarks of the eighteen simulated, which correspond to the three benchmarks with the lowest replication degree (see Figure 7): *mgrid*, *gzip* and *vpr*. For the remaining fifteen benchmarks, the Non Redundant Cache does not cause any increase in misses with respect to a conventional cache.

Figure 9 plots the miss ratio vs. area for different configurations. The X-axis shows the total area required in cm^2. The line labelled as 100% corresponds to a conventional cache. The other lines correspond to a Non Redundant Cache with different sizes of the VT ranging from 20% to 50% to the reference cache. The different dots in a line correspond to configurations with a different number of entries in the PT, corresponding to the number of entries of a conventional cache with 256KB, 512KB, 1MB, 2MB and 4MB.

The results show that the Non Redundant Cache outperforms the base configuration. For a fixed die area, the Non Redundant Cache provides a significant reduction in miss ratio. For instance a 256KB cache base configuration has a 45% miss ratio, whereas the Non Redundant Cache has a miss ratio of 39%, 38% and 37% for a VT reduced to 50%, 30% and 20% respectively. Alternatively, the Non Redundant Cache provides the same miss ratio as a conventional cache but with a smaller area. For instance, a 256KB base configuration with a 45% miss ratio occupies 0.072 cm^2, whereas the Non Redundant Cache achieves the same miss ratio with a die area of 0.054, 0.048 and 0.044 cm^2 for a VT reduced to 50%, 30% and 20% respectively.

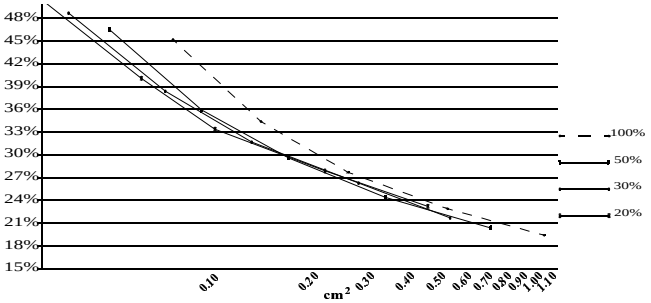

Figure 9. Miss ratio vs. die area for second level data caches

5. Related Work

Several studies [8],[9],[11],[15] report the significant degree of value replication in different parts of superscalar processors. Several recent works focus on the value replication phenomenon for data caches. Zhang *et al.* [15] coined the term *frequent value locality*. They observed that a few values appear frequently in the memory locations involved in a large fraction of memory accesses. Based on this observation, they 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. Both caches are accessed in parallel to provide memory values. Yang *et al.* [14] present a similar cache design called CC (Compression Cache) where each line can hold one uncompressed line or two cache lines that have been compressed to at least half of their lengths. A modification of the FVC is proposed in [13] in order to improve energy efficiency.

Significance compression is used by Brooks *et al.* [2], Villa *et al* [12] and Canal *et al.* [4] to reduce power dissipation, not only in data cache but in the full pipeline.

Black *et al* [1] introduce a mechanism called block-based trace cache that also uses pointers to avoid replication in the trace cache.

6. Conclusions

We have shown the high degree of value replication that is present in conventional data caches. This value replication increases as caches become bigger. The underlying concept of our proposal is based on leveraging this phenomenon in order to reduce the area, power dissipation and access time. We have presented a novel data cache design called Non Redundant Cache which avoids the replication of values. The Non Redundant Cache also includes a simple compression scheme based on inlining narrow values for values that require less bits than their corresponding pointers. Simulation results show that the Non Redundant Cache outperforms conventional caches in terms of power dissipation, access time and die area at the expense of a very minor increase in miss ratio.

7. References

[1] B. Black, B. Rychlik, and J. Shen, "The Block-Based Trace Cache". In Proceedings of the 26th *Int. Symp. on Computer Architecture*, 1999.

[2] D. Brooks, V. Tiwari and M. Martonosi, "Watch: A Framework for Architectural Level Power Analysis and Optimization". In Procs. of the 27th *Int. Symp. on Computer Architecture*, 2000.

[3] 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

[4] R.Canal,A.González,and J.E.Smith,"Very Low Power Pipelines using Significance Compression". In Procs. of the 33th *Int. Symp. on Microarchitecture*, 2000.

[5] M.Gowan, L.Brio and D. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor". In Procs. of the 35th Design Automation Conference,1998.

[6] http://www.sandpile.org/, "The world's leading source for pure technical x86 processor information".

[7] J. Huh, D.C. Burger, and S.W. Keckler, "Exploring the Design Space of Future CMPs". In Proceedings of the *Int. Conference on Parallel Architectures and Compilation Techniques*, 2001. .

[8] M.H. Lipasti, C.B. Wilkerson and J.P. Shen, "Value Locality and Load Value Prediction", I*n* Procs. of the *7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1996.

[9] C. Molina, A. González and J. Tubella, "Reducing Memory Traffic Via Redundant Store Instructions". In Procs. of the *Int. Conf. on High Performance Computing and Networking*, 1999.

[10] P. Shivakumar, N. P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power and Area Model". *Western Research Lab (WRL), Research Report 2001/2.*

[11] A. Sodani and G.S. Sohi, "Dynamic Instruction Reuse". In Procs. of the 24th *Int. Symp. on Computer Architecture*, 1997.

[12] L. Villa, M. Zhang, and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction". In Procs. of the *33th Int. Symp. on Microarchitecture*, 2000.

[13] J.Yang, and R. Gupta, "Energy Efficient Frequent Value Data Cache Design", In Procs. of the *35th Int. Symp. on Microarchitecture*, 2002.

[14] J.Yang, Y. Zhang and R. Gupta, "Frequent Value Compression in Data Caches". In Procs. of the *33th Int. Symp. on Microarchitecture*, 2000.

[15] Y. Zhang, J. Yang and R. Gupta, "Frequent Value Locality and Value-Centric Data Cache Design". In Procs. of the *9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2000.