Energy Optimization Techniques Using Cache Optimization techniques

Term Paper Report

By

Vaibhav Kaushik(2018H1030128P)

Sachin P C(2018H1030140P)

on

25th April 2019

under the guidance of

Mayuri A. Digalwar

Assistant Professor, CSIS Dept, BITS Pilani

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI (Rajasthan)

INDEX

1. ABSTRACT

As the technology is increasing the cache implementation is becoming much better and hence the performance of the processors are increasing rapidly but with the increase in the cache sizes per die area the leakage energy is increasing which is also affecting the applications run, where the batteries are powered off quickly and also many other problem. In this paper, we refer to some of the papers which talk about various techniques, which helps in decreasing the leakage current using the cache optimization techniques. One can observe that the energy leakage reduces considerably in many techniques used.

2. INTRODUCTION

With the latest processors, the leakage energy is increasing and cache are one of the main reasons causing high energy leakage. As the technology has progressed in designing processors, the leakage of energy has increased considerably because of the increase in the power density and also by the increase in size of die area of caches. Because of the increased sizes of the last level caches in the new transistors, the energy consumption in the last level caches have increased rapidly. Hence, this directly affects the performance of the processors. Steps has to be taken to reduce the power consumption in the last level caches. The leakage energy affects the applications considerably as they will make the devices powered off pretty quickly if the battery backup is not that good. Hence, avoidance of leakage current is one of the major priorities while building processors. Having said that, the caches produce very high leakage current and hence, in this paper we refer to techniques which works on reducing the leakage energy by applying some techniques on the caches. One observation which can be made with the caches is that, if the cache blocks are “ON” for more number of time, then the power consumed will be more and hence the leakage energy will be more. Hence, based on this, we can also notice that not all the cache blocks are used all the time. The usage of the cache block depends on the application. Hence, if the application is not using all the caches, it will be more useful to disable the unused cache blocks and to enable it when required. So, maNy papers referred refers to the dynamic cache block enabling and disabling which increases the idleness of the cache blocks which thereby decreases the leakage energy.

3. TECHNIQUES

In this section, we explain some of the cache optimization techniques which decreases the leakage energy.

a. Existence Of Cache Is Not There:

Starting from the very beginning, when we were not having very high end computational requirements and technology was not so advanced, there used to be no presence of cache.

Fetching data from hard disk or main memory incurred good amount of energy usage as things like capacitance overhead of device I/O, board traces and larger memory components. It was not much of a concern in earlier time as the memory accesses were less due to not so much intensive calculations required. But, with increasing complexity of programs and computations required the access started to increase and increased the energy consumption. A better way of accessing data was required that accessed data fast and also leads to lesser energy consumption.

b. On Chip Cache Introduction:

We now came across a memory named as cache that was less in size, fast in access and consumed less power, with introduction of on chip cache the execution speed also got better and we were having both performance and energy advantage. As time went on, we noticed a requirement of increasing the on chip cache size. But, the problem was that continuously increasing on chip cache size led to increase in latency of access as more logic depths to decode addresses was required and also there was increase in capacitance on bit-lines and word-lines. Moreover, with more speed requirement putting on more transistors, more sized cache became increasingly demanding, but so much things on a single die required better technological efforts. It was noticed that trying to put so many things on a single die led to leakage in energy.

c. Multi-Level Cache Introduction:

With increase in leakage of energy due to decreasing size and increasing transistors, a step to have multi-level cache was taken so that the on chip cache size can be reduced and a larger size can be given to second level cache. This is what being used in modern computing with optimizations upon it. Memory bit array partitioning or dividing techniques reduce bit array power dissipation due to large capacitance on bit-lines and word-line [1].

Having independently addressable, smaller modules that are partitions of memory leads to increase in energy efficiency, we call this technique as MDM (Multi Divided Module).

One more technique for increase in energy efficiency is of partitioning the cache into levels of L0 and L1, where L0 is smaller cache that is faster and consumes less power, L1 cache is a kind of support to L1 cache that comes into picture when we miss in L1.

d. Resizable Cache Designing:

Since now in present era, we are having a variety of applications that use cache to provide user a better experience, we see that there is a requirement of varying the ever since static nature of cache size, i.e. we need to vary the size of cache according to needs of application and turning off unused part of the cache that is unnecessarily leading to energy wastage[2]. We call this technique Resizable Cache Designing.

This can be implemented in two variants:

- ➢ Cache Organization:

  i) Selective ways: varies cache's set-associativity

  ii) Selective sets: with this variation in number of sets can be achieved.

- ➢ Resizing Strategy:

  i) Static: Set cache size before application starts running.

  ii) Dynamic: Observe application usage and resize when necessary.

All these different techniques differ in

(1) Cache size range.

(2) Granularity of resizing.

(3) Set-associativity at different resizing.

(4) Complexity in implementation.

In case of selective ways, we set the cache size before execution (static resizing) of the application and thus it benefits us only across applications having different cache utilization, whereas in selective sets we decide the cache size and its resizing by monitoring cache usage and miss ratios, this way we can achieve cross application and within application cache resizing benefits.

Below figures explain how selective ways and selective sets are organized.

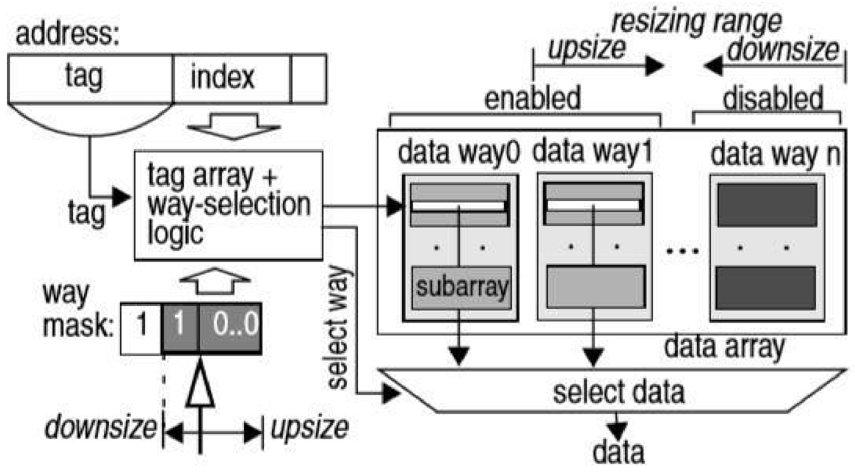


Fig 1. Selective ways with way mask as the basis for masking whole subarray

We can see each of associative ways can be enabled or disabled in selective ways. In normal set associative caches, data array is designed as cache ways having subarrays. Given a way, with help of way-mask subarrays can be enabled/disabled. Hardware or software sets the way-mask to adjust the number of ways used by cache.

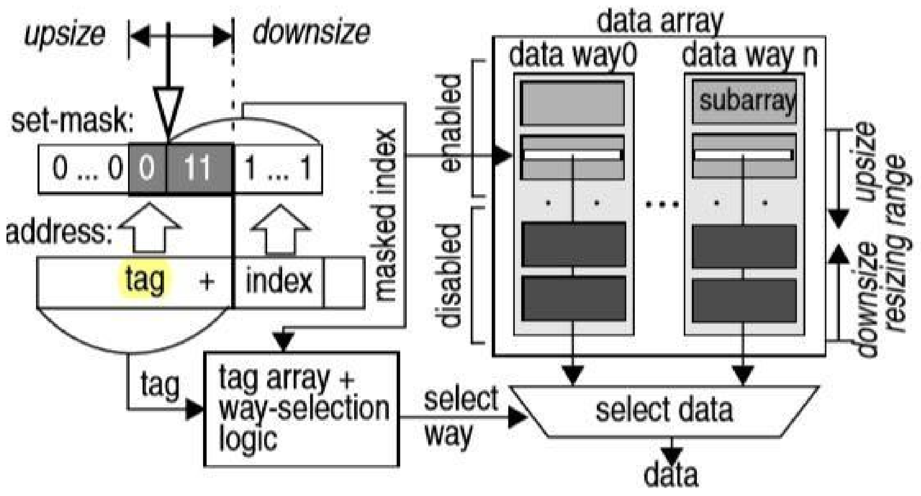


Fig 2. Selective sets with set mask as the basis for masking subarray in data ways

Alternatively, cache sets are enabled/disabled in selective sets. Above figure shows us how change of index and tag bits in selective-sets can change cache sets numbers and the cache block size, it is evident that this can be achieved by use of set mask.

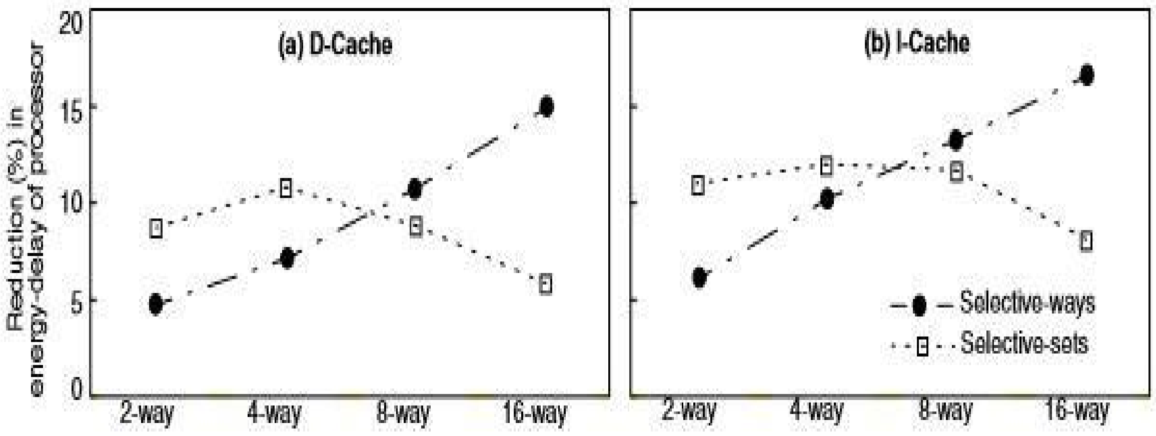


Fig 3. Energy Reduction percentage versus ways

This figure explains that when number of sets are not varied, but we allow to vary set associativity then selective ways performs better. Although, selective ways perform better with high associativity, it does not perform well with full associativity.

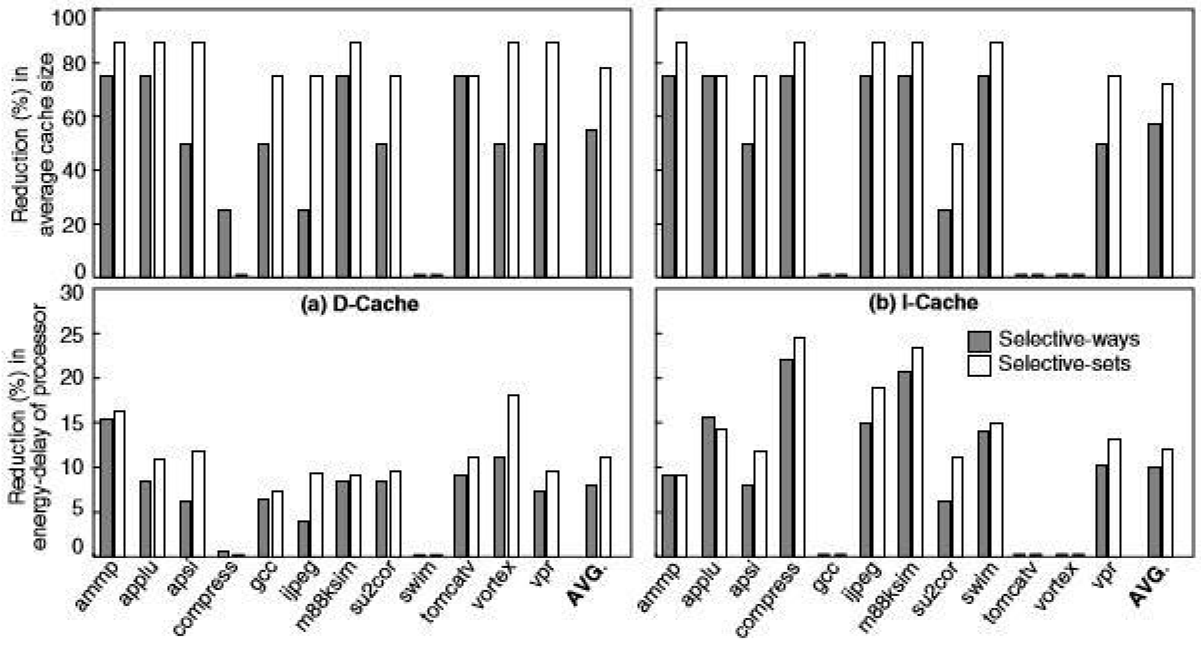


Fig 4. Energy Reduction percentage versus different application and changing sets, constant set associativity of 4 way.

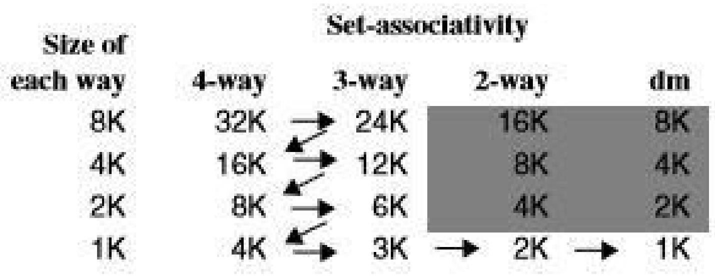
We can see that on keeping setting set associativity constant and varying sets across different applications, selective sets perform better than selective ways.

Since every technique has its own advantage and disadvantage, we will now see a combination of two technologies, namely selective sets and selective ways.

- *Hybrid-selective-sets-selective ways:*

As the choices of cache sizes offered by each of the resizable cache organizations is less and neither of them completely contains the other, we need better choices of cache sizes, this is because with selective-ways we can have cache sizes that are multiples of way size and with selective-sets powers of two**.**

So, the aim now becomes to provide the flexibility of having closet available cache size according to application need. We can see from the table below, how the hybrid technique can provide more set choices for required cache sizes.



	Set-associativity			
Size of each way	4-way	3-way	2-way	dm
8K	32K	24K	16K	8K
4K	16K	12K	8K	4K
2K	8K	6K	4K	2K
1K	4K	3K	2K	1K

Fig 5. Table showing with arrows the possible cache sizes with hybrid technique

We observe from this table that from a 4-way set associative cache having size 32K and a subarray size of 1K, a hybrid cache gives us lot of possible combinations like 1K, 2K, 3K, 4K, 6K, 8K, 12K, 16K, 24K, and 32K sizes[2]. On the other hand sizes of 8K, 16K ,24K ,32K are offered using selective-ways cache and 4K,8K,16K,32K using selective-sets[2].

We can also see the improvements that are being achieved by using hybrid technique both in cases where we increase set associativity and keep it constant.

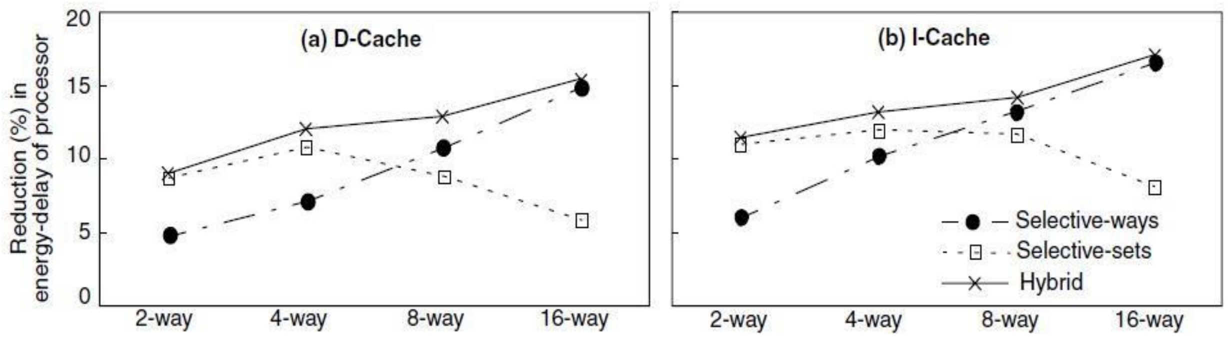


Fig 6. Energy Reduction percentage versus ways

It is evident from the graph shown that hybrid approach is performing better than both selective sets and selective ways.

e. Cache Partitioning and Reconfiguration:

Till now we have seen that we have improved the cache utilization according to usage by the application. Now let us look at multiprocessor environment, we look at approach of dynamically configuring the on chip cache and partitioning the shared cache. Aim in this technique is to find the beneficial cache configuration and portioning factors for private and shared cache respectively.

DCR (Dynamic cache reconfiguration) is a concept that focuses on changing the cache configurations at runtime. We generally focus on applying DCR for on chip caches. L2 caches in multiprocessor environment are shared and parallel running tasks increase contention in L2 cache.

So we require concept of cache partitioning so that we can dedicate partitioned part to cores accordingly. It is found out that applying DCR in L1 cache impacts cache partitioning decisions for shared L2 cache. Interestingly, both DCR and CP are important factors for energy conservation.

It is not only DCR or CP that alone can greatly reduce energy consumption, but also when and how these techniques are being applied. In some of the works already done, Suh et al. [3] showed partitioning of shared cache by replacement unit by hardware counters utilization. Qureshi et al. [4] did monitoring of cache utilization of each application and proposed less overhead incurring CP methods. Kim et al. [5] used dynamic and static partitioning techniques to provide fair cache sharing.

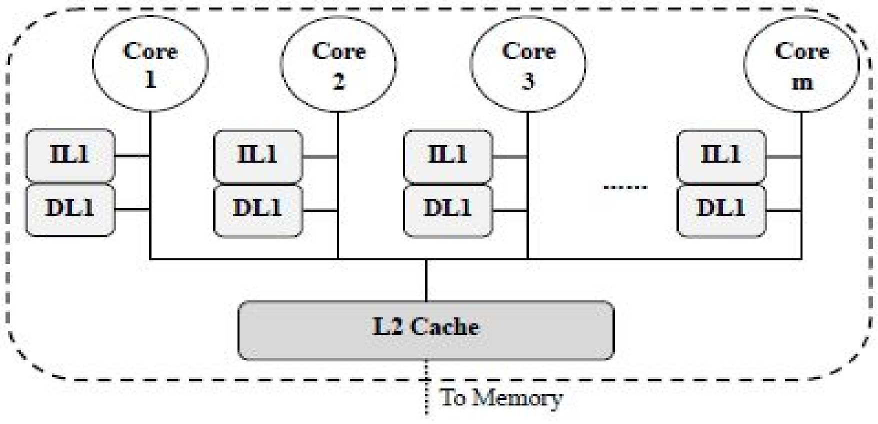


Fig 7. Multiprocessor cache architecture with shared and private caches

This architecture we assume is highly configurable in terms of line line size ,effective capacity, and associativity. To change cache size we use gate V_{dd} technique by shutting down the banks, associativity can be logically changed by means of concatenating neighboring ways. Configuration for line size can be done by specifying number of fetched unit-length blocks.

Here we use a way based partition technique in shared cache. These portions of the cache are assigned to each core. Although, in each individual portion LRU replacement takes place by maintaining age bit.

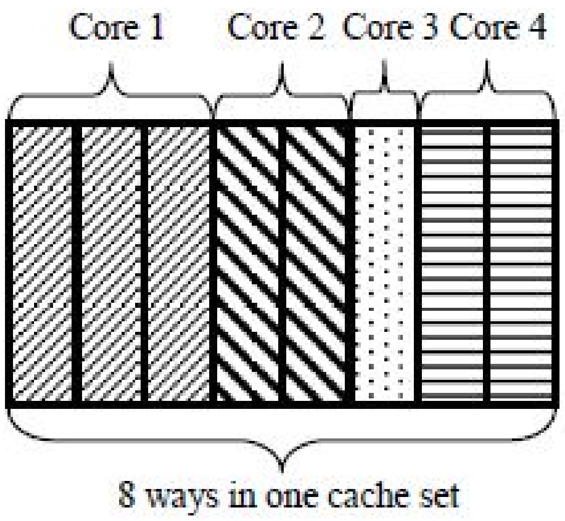


Fig 8. Cache partitioning based on ways

We model cache energy consumption E as $E = E_{static} + E_{dynamic}$. Where E_{dnamic} is dependent on cache miss and accesses, i.e. $E_{dynamic} = n_{accesses} . E_{access} + n_{misses} . E_{miss}$

Where n_{access} and n_{miss} respectively represent number of cache accesses and misses.

Also, $E_{miss} = E_{memside} + E_{block\_fill} + E_{Up\_stall}$.

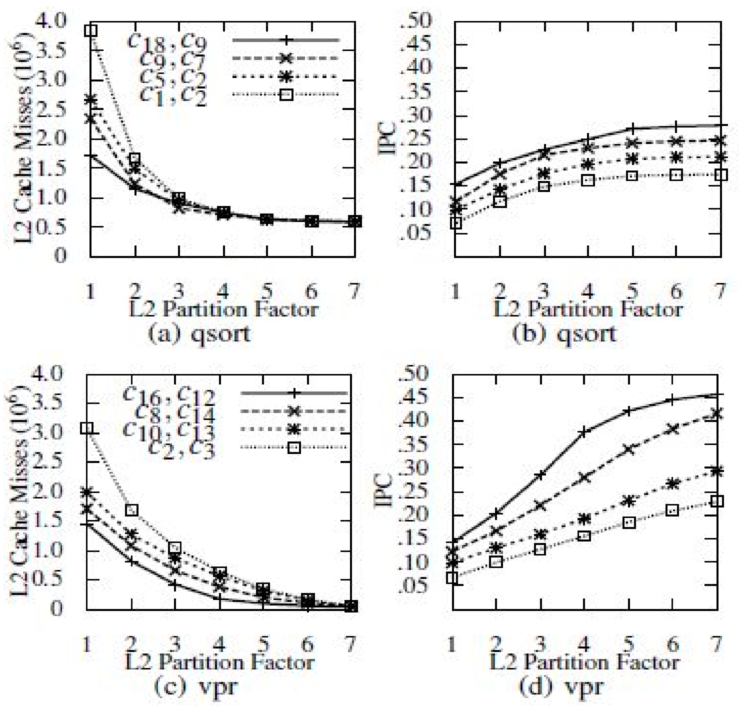
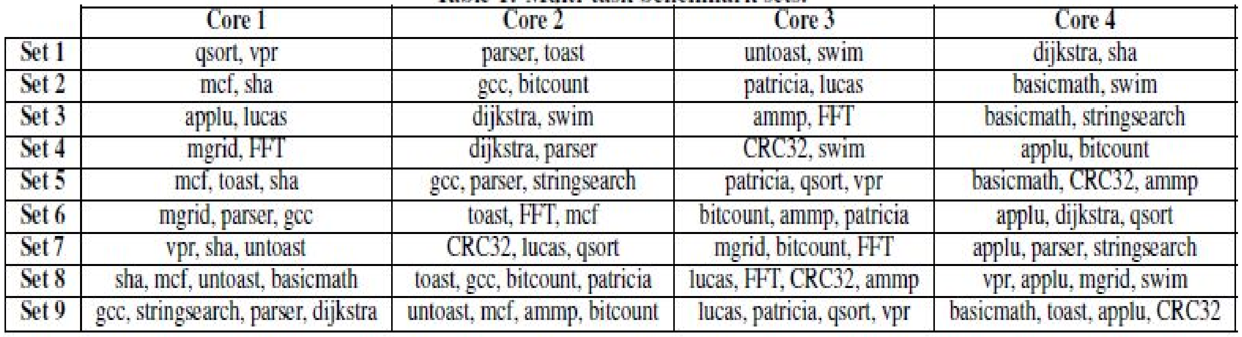


Fig 9. L1 DCR impact on L2 CP in performance.

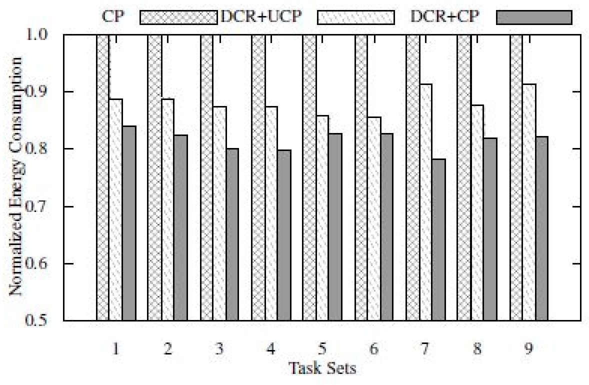
It is evident from the above figure that L1 DCR decisions greatly affect L2 CP. Optimization of energy is performed using static profiling. If we search for the best combination of cache configuration for L1 cache and partitioning for L2 cache in an exhaustive manner then a lot of time is wasted in doing this. Therefore we need to perform tasks based analysis as tasks are not application specific except for the resources that are contented. So, while using L2 cache partitioning cores can be viewed as uniprocessors with x-way associative memory.

We will now look at some of the results from experiments that have been performed.

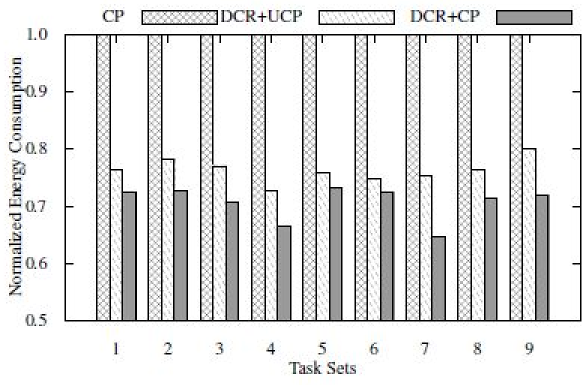
Table 1: Multi-task benchmark sets.



	Core 1	Core 2	Core 3	Core 4
Set 1	qsort, vpr	parser, toast	untoast, swim	dijkstra, sha
Set 2	mcf, sha	gcc, bitcount	patricia, lucas	basicmath, swim
Set 3	applu, lucas	dijkstra, swim	ammp, FFT	basicmath, stringsearch
Set 4	mgrid, FFT	dijkstra, parser	CRC32, swim	applu, bitcount
Set 5	mcf, toast, sha	gcc, parser, stringsearch	patricia, qsort, vpr	basicmath, CRC32, ammp
Set 6	mgrid, parser, gcc	toast, FFT, mcf	bitcount, ammp, patricia	applu, dijkstra, qsort
Set 7	vpr, sha, untoast	CRC32, lucas, qsort	mgrid, bitcount, FFT	applu, parser, stringsearch
Set 8	sha, mcf, untoast, basicmath	toast, gcc, bitcount, patricia	lucas, FFT, CRC32, ammp	vpr, applu, mgrid, swim
Set 9	gcc, stringsearch, parser, dijkstra	untoast, mcf, ammp, bitcount	lucas, patricia, qsort, vpr	basicmath, toast, applu, CRC32



(a)



(b)

The above two figures show the set of tasks that have been experimented and the results obtained in terms of energy consumption on increasing cache sets. Cache used in fig a) is 4KB 2 way set associative with 32 bytes block size and fig b) 4KB 4 way set associative with 64 bytes block size. It is evident from the above results that energy consumption has been reduced by combining DCR and CP.

Some Experimental Cache Organizations

f. Vertical Cache Partitioning (Block Buffering Approach):

This approach saves power as for every cache access it optimizes capacitance of access. How effectively block buffering can work although depends on the block sizes and spatial locality of applications[6]. Higher the spatial locality more will be the energy saving. We also need to effectively choose block size here as the small block size limits the amount of saved energy and larger block can result in increase of energy consumption due to bigger unused portion. Here processor checks for block hits and goes to cache only on misses.

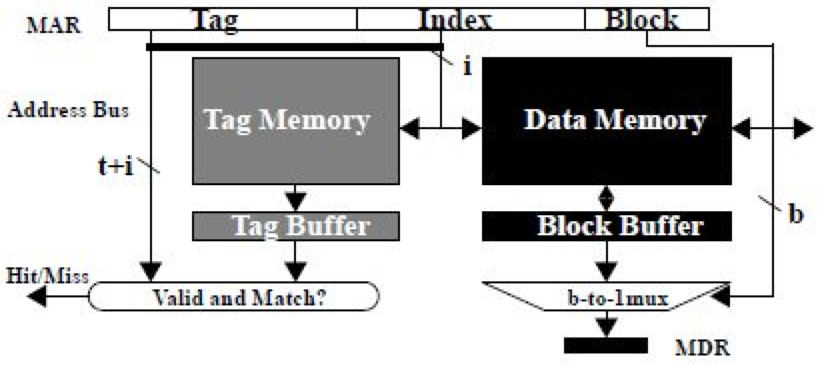


Fig 10. Block Buffering Concept

g. Horizontal Cache Partitioning:

Concept here is to partition the cache into segments and then power each of them individually [6]. This way only those segments having data are accessed and thus only necessary energy is consumed. Major advantage of this approach in comparison to block buffering is that effective hit time is comparably fast to conventional cache.

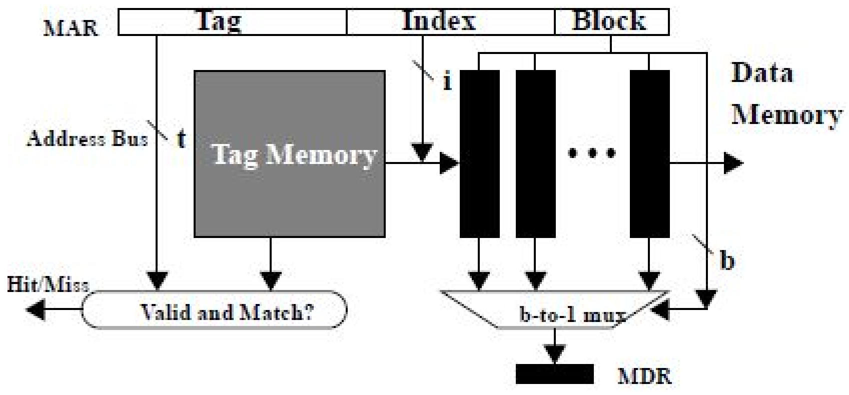
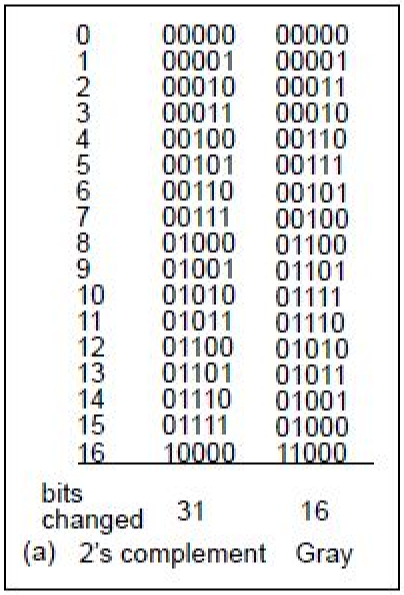


Fig 11. Cache sub banking

h. Gray Code Addressing:

Since traditionally we are using 2’s complement representation for memory addressing, switching of bits on accessing of seq. memory is not optimal[6]. We can work on this objective of optimizing bit switching activities. Using gray code is helpful as gray code is such a code in which by changing only 1 bit we can move from one given code to another. As an example, number of switches required for numbers from 0 to 16 with 2’s complement representation is 31 whereas with gray code addressing it is 16. We can have a look at the table below.



0	00000	00000
1	00001	00001
2	00010	00011
3	00011	00010
4	00100	00110
5	00101	00111
6	00110	00101
7	00111	00100
8	01000	01100
9	01001	01101
10	01010	01111
11	01011	01110
12	01100	01010
13	01101	01011
14	01110	01001
15	01111	01000
16	10000	11000
bits changed	31	16
(a)	2's complement	Gray

Fig 12. Table showing gray code and 2's complement bits for 1 to 16

Diagram below shows the classification of all techniques that we have addressed in the report.

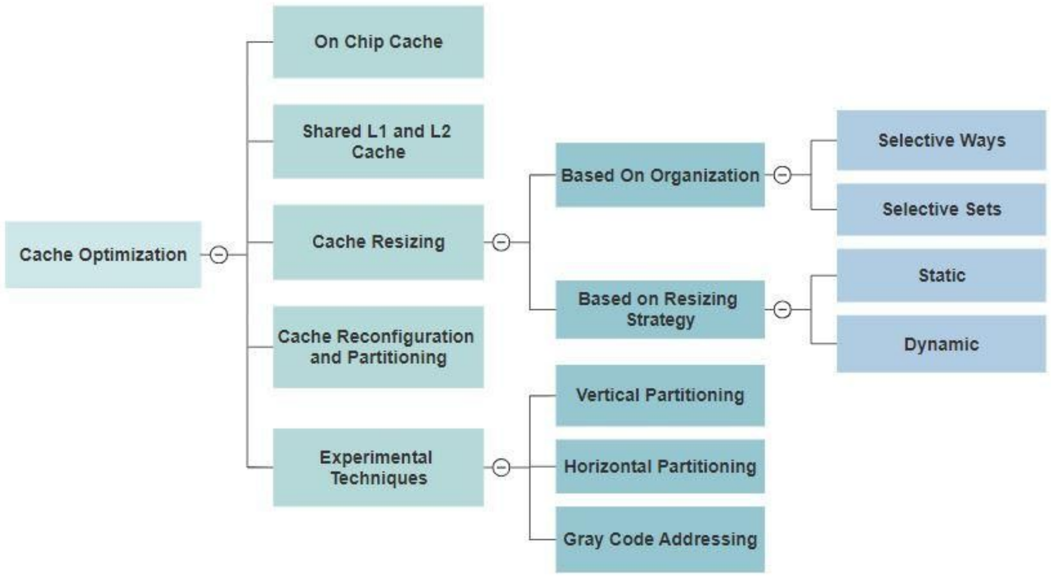


Fig 13. Classification of the techniques

i. Locality-Driven Architectural Cache Sub-banking for Leakage Energy Reduction[7]:

With the latest processors, the leakage energy is increasing and cache are one of the main reasons causing high energy leakage. So, in his method, the cache blocks are managed by enabling and disabling the cache blocks. This technique disables the cache to reduce the leakage energy. Here, the address blocks are mapped to the cache blocks and try to disable the cache blocks the most depending on the mapped cache blocks with the address blocks of the application run. The main aim of this technique is to reduce energy leakage and to do that, the idleness of the cache should be maximized. Once, a cache block is selected for idleness, it is given a voltage V_{min} which is the minimum voltage so that it can just be on but not able to store data. By giving V_{min}, to the cache, we are making the cache idle. Once, the cache has to be brought back from its idleness, then we give that block a voltage V_d, which is required for cache to be enabled.

Architecture:

The architecture created to perform this technique uses a cache controller to handle the cache blocks which consists of a tag field and a data field. The main objective of this cache controller is to identify the non contiguous address blocks and also to identify the cache blocks and to map each of the address ranges to one of its cache blocks.

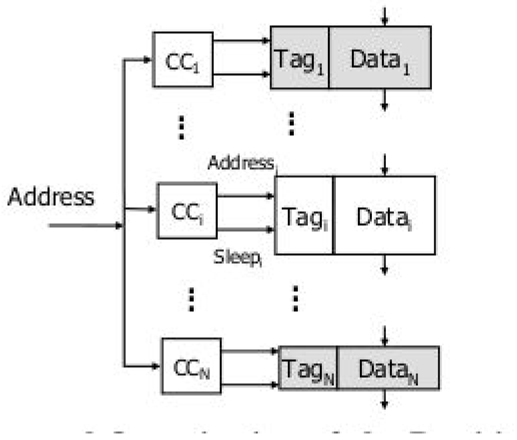


Fig 14. Cache organization using cache controller.

Methodology:

Two steps to be followed in this technique is:

1. The address ranges has to be found out which are non overlapping
2. Mapping of the address ranges with a corresponding cache block.

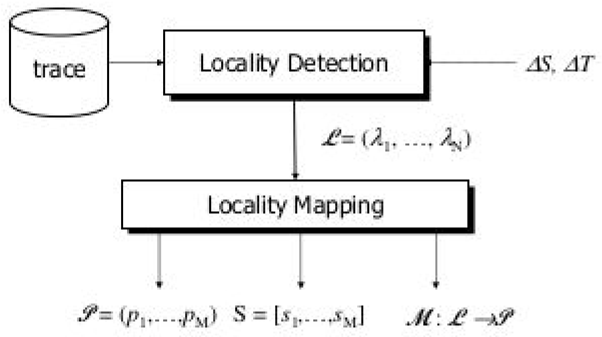

trace
Locality Detection
ΔS, ΔT
$\mathcal{L} = (\lambda_1, \ldots, \lambda_N)$
Locality Mapping
$\mathcal{P} = (p_1, \ldots, p_M)$
$S = [s_1, \ldots, s_M]$
$\mathcal{M} : \mathcal{L} \to \mathcal{P}$

Fig 15. Methodology of the given technique.

So, in this technique, we use a trace of the application memory to get the different address ranges and fetch it from the locality detection and then we map it with the cache blocks with the help of locality mapping.

Terminology:

Trace $T = \{ a_1, a_2, \ , \ , a_n\},\ where\ a_i$ is the address referred to at the clock cycle i.

$\Delta A = Address\ Range,\ \Delta R = Time\ Range$.

A binary matrix is created to map the address and time. The map will have a value of one when that particular block was accessed at that particular time, else zero.

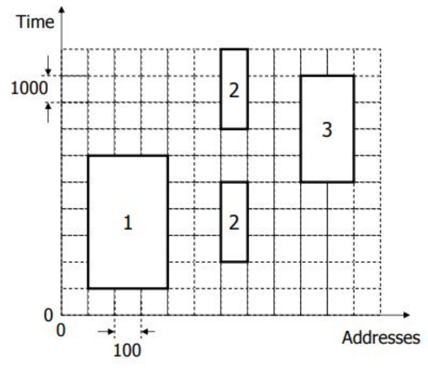
NOw, that locality is defined by $Z = \{la, ha\}, T_a$, where la and ha represents the range of address blocks accessed and Ta represents the list which contains the time duration at which it was accessed.

Consider an example, where

Z1 = { [100, 400]; {[1000, 6000]} }

Z2 = {[600, 700]; {[2000, 5000], [7000, 10000]}}

Z3 = {[900, 1100]; {[5000, 9000]}}, the locality plot for this is given by the following figure.



Detection of Locality:

Now, to get the values of the localities, we use the mapped binary matrix to find the non contiguous locations. Basically, we go on until the matrix value is zero and once we get a matrix value which is null or zero, we get the upper range of the address for that locality and hence it forms one address block. Similarly, we get all the address blocks, by repeating the same thing until all the clock cycles are over[9].

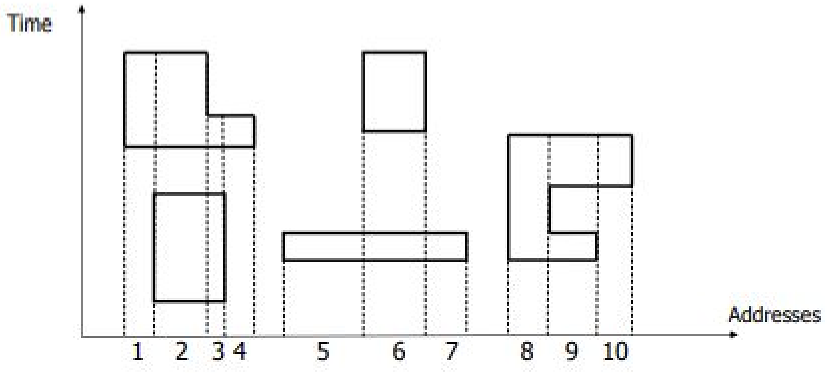


Fig 16: Locality Detection

The main objective of this implementation is to increase the idle time of the cache blocks and hence, we can increase the idle time of the cache blocks by merging the localities such that the time interval between the acce of those localities is more and by mapping them to the same cache block. This will increase the idle time of the cache blocks, because when these localities sharing the maximum number of time interval are pointed to the cache block, then they will all be accessing the same cache block and hence all other cache blocks can be turned off and hence the energy leaked will be minimum.

Now, the main task is to merge the localities and assign the blocks to those localities. As, finding all the possibilities will be a NP complete problem, in this technique the use a greedy approach to merge the localities.From the localities, a graph is created, where the vertices will represent the localities found and the edges represent the temporary merging of the localities. The weights of the graph indicates the time interval shared between those two end vertices.

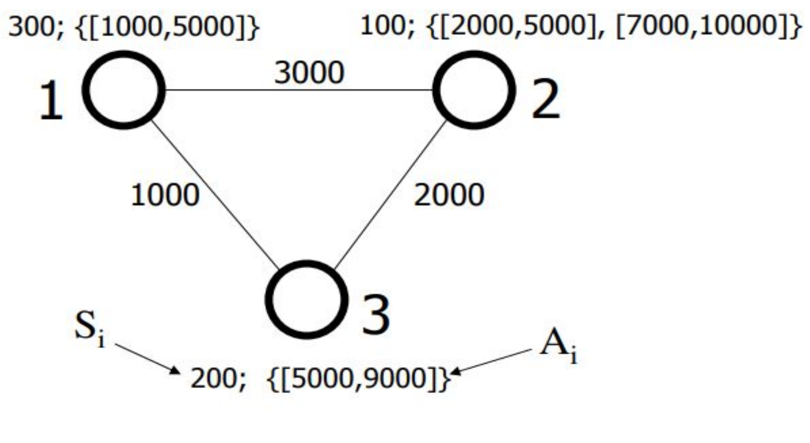


Fig 17.: Graph of the Localities

Now, the 2 localities are merged using a greedy approach and the one which is giving the lowest energy leakage, we choose that for merging and follow this step until we get the optimal energy leakage values. So, once all the steps are done, the merged localities are the mapped to the same cache block[10].

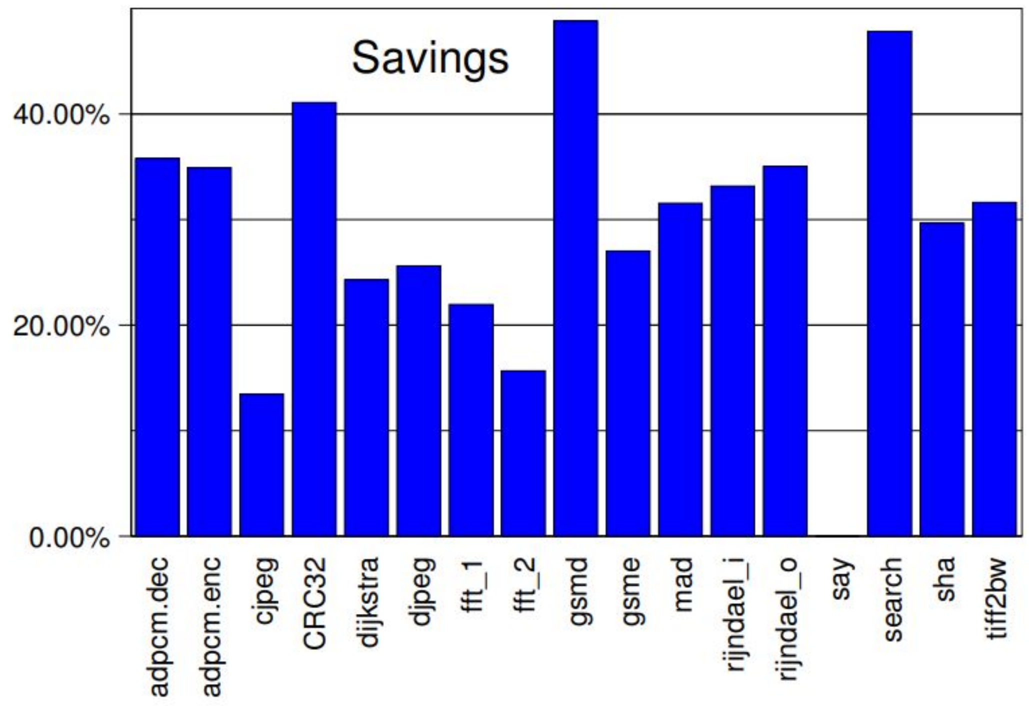
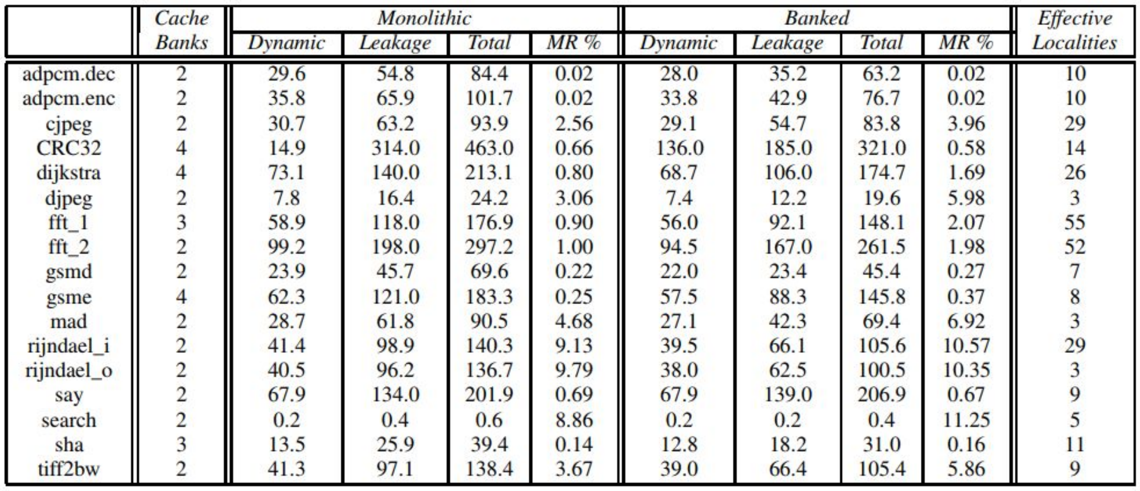


Fig 18. Banked Cache Energy Leakage Savings



	Cache	Monolithic				Banked				Effective
	Banks	Dynamic	Leakage	Total	MR %	Dynamic	Leakage	Total	MR %	Localities
adpcm.dec	2	29.6	54.8	84.4	0.02	28.0	35.2	63.2	0.02	10
adpcm.enc	2	35.8	65.9	101.7	0.02	33.8	42.9	76.7	0.02	10
cjpeg	2	30.7	63.2	93.9	2.56	29.1	54.7	83.8	3.96	29
CRC32	4	14.9	314.0	463.0	0.66	136.0	185.0	321.0	0.58	14
dijkstra	4	73.1	140.0	213.1	0.80	68.7	106.0	174.7	1.69	26
djpeg	2	7.8	16.4	24.2	3.06	7.4	12.2	19.6	5.98	3
fft_1	3	58.9	118.0	176.9	0.90	56.0	92.1	148.1	2.07	55
fft_2	2	99.2	198.0	297.2	1.00	94.5	167.0	261.5	1.98	52
gsmd	2	23.9	45.7	69.6	0.22	22.0	23.4	45.4	0.27	7
gsme	4	62.3	121.0	183.3	0.25	57.5	88.3	145.8	0.37	8
mad	2	28.7	61.8	90.5	4.68	27.1	42.3	69.4	6.92	3
rijndael_i	2	41.4	98.9	140.3	9.13	39.5	66.1	105.6	10.57	29
rijndael_o	2	40.5	96.2	136.7	9.79	38.0	62.5	100.5	10.35	3
say	2	67.9	134.0	201.9	0.69	67.9	139.0	206.9	0.67	9
search	2	0.2	0.4	0.6	8.86	0.2	0.2	0.4	11.25	5
sha	3	13.5	25.9	39.4	0.14	12.8	18.2	31.0	0.16	11
tiff2bw	2	41.3	97.1	138.4	3.67	39.0	66.4	105.4	5.86	9

Fig 19. Comparison between monolithic and banked ache energy leakage

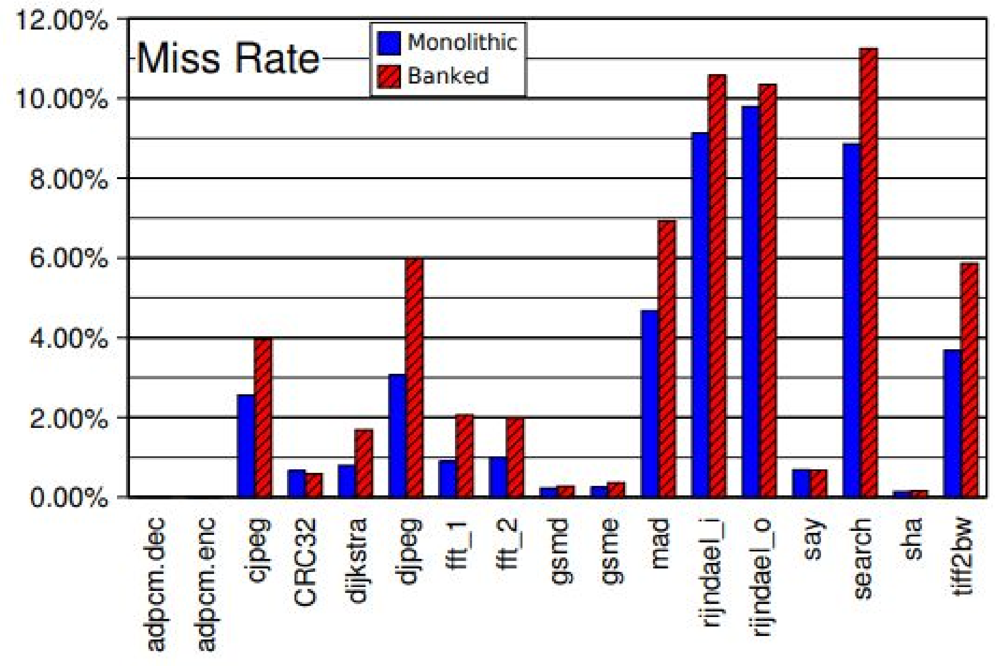


Fig 20. Miss rate of Banked and Monolithic designs

j. Static Energy Reduction by Performance Linked Dynamic Cache Resizing[8]

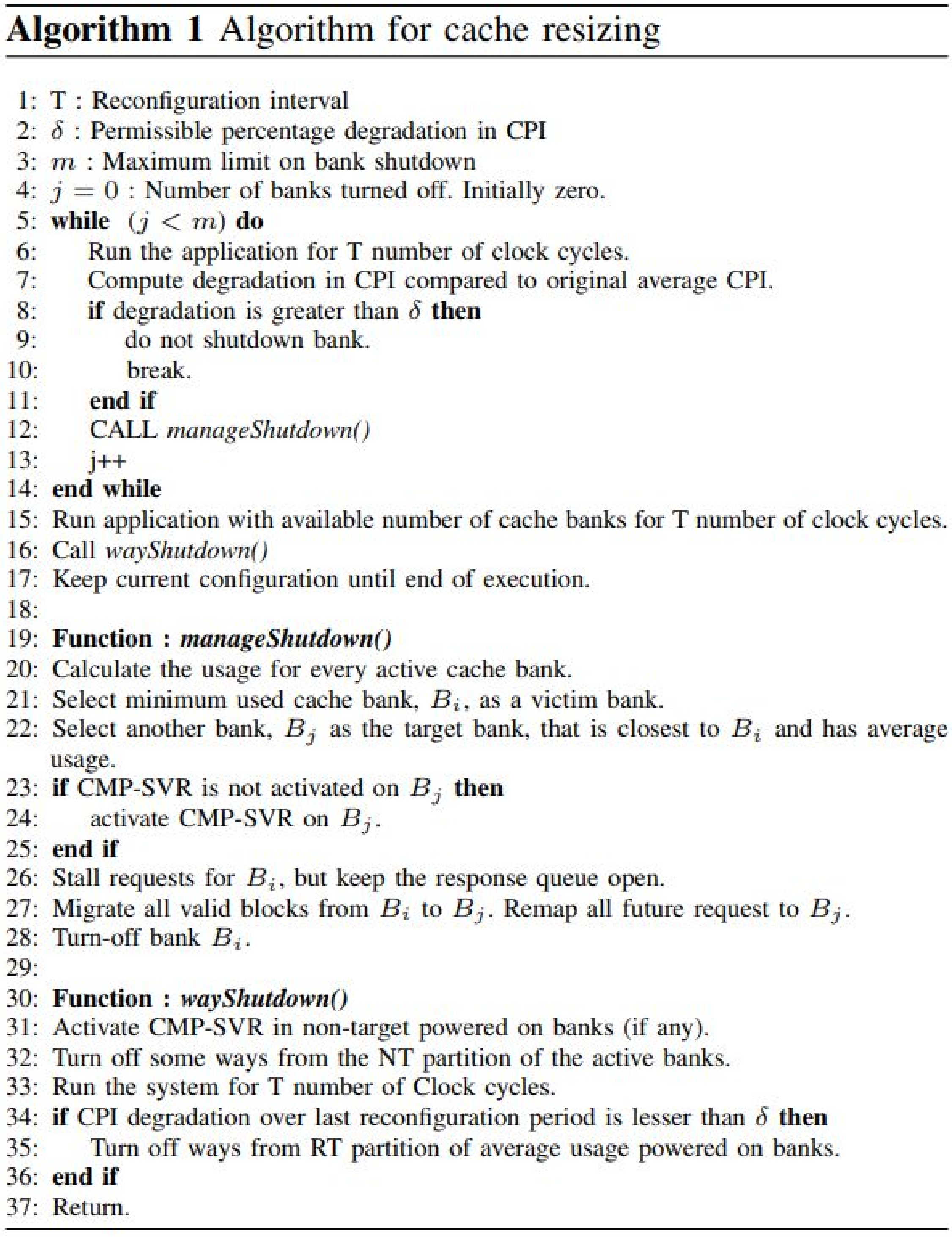
As the technology has progressed in designing processors, the leakage of energy has increased considerably because of the increase in the power density and also by increase the size. Because of the increased sizes of the last level caches in the new transistors, the energy consumption in the last level caches have increased rapidly. Hence, this directly affects the performance of the processors. Steps has to be taken to reduce the power consumption in the last level caches. One thing to be noticed is that not all the time, full caches are required. In many cases, the cache required will be much less than the total cache present. Hence, in this technique, the cache are dynamically resized by deciding the working set of the cache depending on the work assigned to the processor. With this technique, once we decide that some of the blocks in the cache are not required, we can make that blocks to consume very less power by not allowing access to those blocks. When the request for accessing cache increases, we can enable some of the cache blocks, which were disabled, thereby dynamically increasing and decreasing the size of cache. This in turn reduces the leakage energy produced by the last level caches. The techniques that can be used to reduce the cache size dynamically are “way shutdown” and “cache bank shutdown”. We can disable the caches who are least used. To overcome the problems of lesser cache sizes, different techniques are applied. This technique gives around 70% reduction in power which was leaking for 4 MegaBytes 8 way set associative level 2 cache. Also, it gives an average of 35% reduction in leakage power for “EDP”.

The technique used in this approach is implemented using three phases. Namely,

- BANK SHUTDOWN
- WAY SHUTDOWN
- ASSOCIATIVITY MANAGEMENT.

In Bank shutdown, the cache blocks which are least accessed are completely shut down to decrease the leakage energy. But the performance degrades when cache block disabled is made to enable quickly[11].

So, to overcome this, we use way shutdown, where the cache blocks with average access are not completely removed, instead the associativity is decreased. But the problem here is that by decreasing the associativity of the cache blocks, the number of cache misses increases and hence leads to the decrease in the performance. To overcome this, we apply associativity management so that the associativity reduced doesn't lead to the decrease in the performance of the processors.



Algorithm 1 Algorithm for cache resizing

```
 1: T : Reconfiguration interval
 2: $\delta$ : Permissible percentage degradation in CPI
 3: $m$ : Maximum limit on bank shutdown
 4: $j = 0$ : Number of banks turned off. Initially zero.
 5: while ($j < m$) do
 6:     Run the application for T number of clock cycles.
 7:     Compute degradation in CPI compared to original average CPI.
 8:     if degradation is greater than $\delta$ then
 9:         do not shutdown bank.
10:         break.
11:     end if
12:     CALL manageShutdown()
13:     j++
14: end while
15: Run application with available number of cache banks for T number of clock cycles.
16: Call wayShutdown()
17: Keep current configuration until end of execution.
18:
19: Function : manageShutdown()
20: Calculate the usage for every active cache bank.
21: Select minimum used cache bank, $B_i$, as a victim bank.
22: Select another bank, $B_j$ as the target bank, that is closest to $B_i$ and has average
    usage.
23: if CMP-SVR is not activated on $B_j$ then
24:     activate CMP-SVR on $B_j$.
25: end if
26: Stall requests for $B_i$, but keep the response queue open.
27: Migrate all valid blocks from $B_i$ to $B_j$. Remap all future request to $B_j$.
28: Turn-off bank $B_i$.
29:
30: Function : wayShutdown()
31: Activate CMP-SVR in non-target powered on banks (if any).
32: Turn off some ways from the NT partition of the active banks.
33: Run the system for T number of Clock cycles.
34: if CPI degradation over last reconfiguration period is lesser than $\delta$ then
35:     Turn off ways from RT partition of average usage powered on banks.
36: end if
37: Return.
```

The algorithm uses an interval called as reconfiguration interval for which amount of time, the application is run and the energy consumption is calculated. If the calculated energy leakage is more than the threshold value, way shut down is applied and the associativity of one of the average accessed cache block is reduced and again the application is run for the reconfiguration interval. Now, again the energy leakage value is checked and if it is less than the threshold leakage, same steps are performed again. When run for the first time, if the energy consumption is less than the threshold and the number of bank shut down cache blocks is less than the threshold, then the cache block with least recently used is determined and then the data present in it is transferred into a nearest cache block which has less data in it. The process repeats until the application is run. By this way, we are reducing the leakage current by dynamically making the cache go to sleep state and is turned on when needed. However, this technique might hamper the performance of the processor if the cache blocks made to sleep and again turned on in very less time again and again. Hence, should take care of this condition.

The bank usage of the system is given by $L2BankUsage(i, j) = \frac{totalAccesses(i, j)}{totalL1Misses(j)} \times 100\%$

4. RESULTS

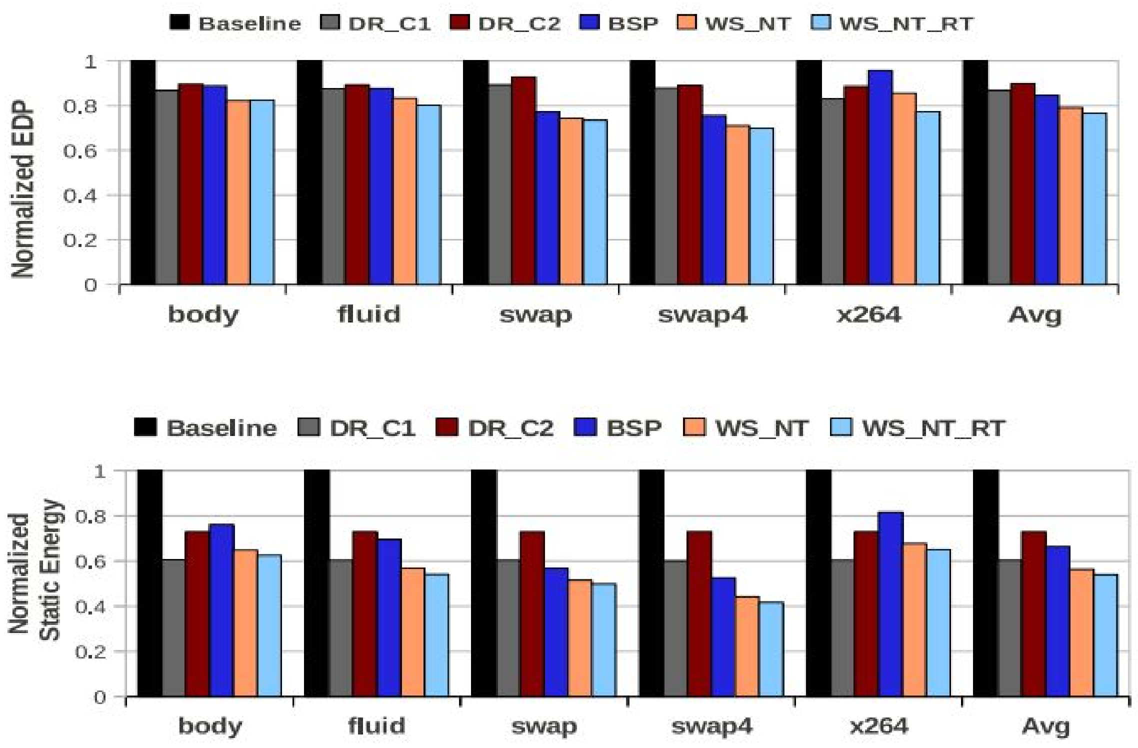


Fig 21.: Comparison of different techniques of Normalized EDP and Normalized static energy of 4 MegaBytes 4-way associative cache

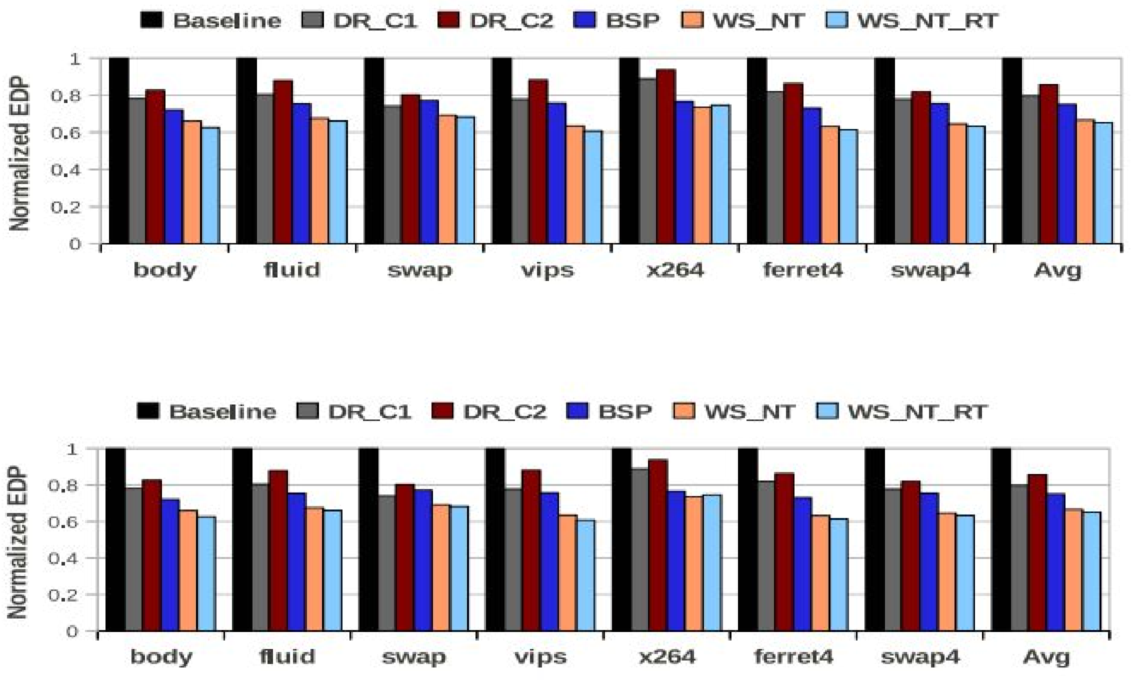


Fig 22. Comparison of different techniques of Normalized EDP and Normalized static energy of 4 MegaBytes 8-way associative cache

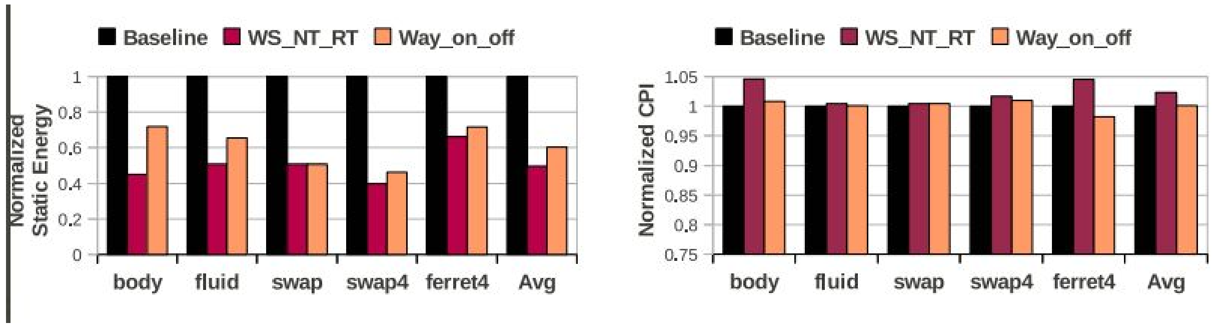
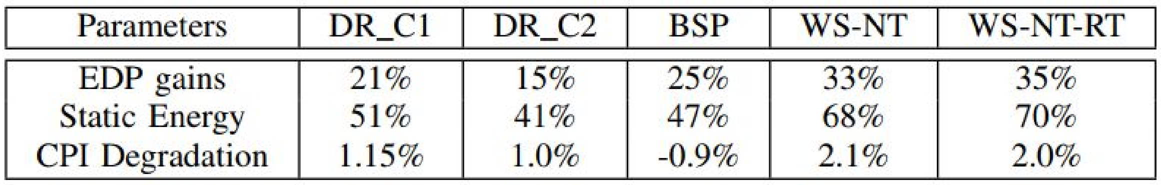


Fig 23. Normalized Static energy and CPI for 2 MegaBytes Level 2 cache



Parameters	DR_C1	DR_C2	BSP	WS-NT	WS-NT-RT
EDP gains	21%	15%	25%	33%	35%
Static Energy	51%	41%	47%	68%	70%
CPI Degradation	1.15%	1.0%	-0.9%	2.1%	2.0%

Fig 24. Comparison of 4-MegaBytes L-2 cache with different techniques.

5. Conclusion

We saw different techniques till now starting from introduction of cache memory, multi level caches, caches that can be resized, reconfigured and partitioned. All these techniques had advantages in their own aspects, while resizing dynamically was always not possible due to overhead of extra calculations, resizing statically could not every time adapt to changing needs. Selective-ways technique in resizing was more impactful in a single application while selective-sets was across different, these things were simply because of the cache sizes these techniques had to offer. Then, we saw a hybrid-selective-ways selective-sets technique which offered cache sizes at a more granular level and offered better energy reduction. Cache reconfiguration and partitioning were applied to multiprocessor architecture having L1 and L2 cache. Since, L1 cache was private to every processor, reconfiguration was applied on L1 cache and partitioning was applied on L2 cache as it was shared. Energy consumption varied on the best technique applied first at L1 level and then impact reducing partitioning strategy at L2 level because of L1 reconfiguration. At last there were some experimental techniques like vertical partitioning which gave idea of using block memory before accessing cache, and thereby reducing the energy to access cache in case of more spatial locality, horizontal cache partitioning focused on powering each segment separately and gray code addressing reduced number of switches.

6. REFERENCES

[1] Uming Ko, Poras T.,Balsara, K. Nanda, Energy optimization of multilevel cache architectures for RISC and CISC processors, IEEE Transactions on Very Large Scale Integration (VLSI) Systems pp. 299-308,1998

[2] Se-Hyun Yang, Michael D. Powell, Babak Falsafi, T. N. Vijaykumar Exploiting Choice in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay, Proceedings Eighth International Symposium on High Performance Computer Architecture, Feb. 2002.

[3] G. Suh et al., “A new memory monitoring scheme for memory-aware scheduling and partitioning,” HPCA, 2002.

[4] M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” Micro, 2006.

[5] S. Kim et al., “Fair cache sharing and partitioning in a chip multiprocessor architecture,” PACT, 2000

[6]. Ching-Long Su and Alvin M. Despain, Cache Design Trade-offs for Power and Performance Optimization: A Case Study, ISLPED '95 Proceedings of the 1995 international symposium on Low power design pp. 63-68, 1995.

[7]. O. Golubeva, M. Loghi, E. Macii and M. Poncino, "Locality-driven architectural cache sub-banking for leakage energy reduction," *Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED '07)*, Portland, OR, 2007, pp. 274-279.

[8]. S. Chakraborty and H. K. Kapoor, "Static energy reduction by performance linked dynamic cache resizing," *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Tallinn, 2016, pp. 1-6.

[9]. L. Benini L. Macchiarulo A. Macii E. Macii M. Poncino "Layout-Driven Memory Synthesis for Embedded Systems-on-Chip" IEEE Transactions on VLSI Systems vol. 10 no. 2 pp. 96-105 April 2002.

[10]. N. S. Kim, T. M. Austin, D. Blaauw, T. N. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. T. Kandemir, and N. Vijaykrishnan, “Leakage current: Moore’s law meets static power,” IEEE Computer, vol. 36, no. 12, pp. 68–75, 2003.

[11]. F. Angiolini L. Benini A. Caprara "An Efficient Profile-Based Algorithm for Scratchpad Memory Partitioning" IEEE Transactions on Computer-Aided Design vol. 24 no. 11 pp. 1660-1676 Nov 2005.