Molecular Caches: A caching structure for dynamic creation of application-specific Heterogeneous cache regions

Keshavan Varadarajan, S.K.Nandy, Vishal Sharda, Amrutur Bharadwaj
Indian Institute of Science Bangalore, India
{keshavan@cadl, nandy@serc, vishal@cadl, amrutur@ece}.iisc.ernet.in

Ravi Iyer, Srihari Makineni, Donald Newell
System Technologies Lab, Intel Corporation, Hilsboro, Oregon
{ravishankar.iyer, srihari.makineni, donald.newell}@intel.com

Abstract

CMPs enable simultaneous execution of multiple applications on the same platforms that share cache resources. Diversity in the cache access patterns of these simultaneously executing applications can potentially trigger inter-application interference, leading to cache pollution. Whereas a large cache can ameliorate this problem, the issues of larger power consumption with increasing cache size, amplified at sub-100nm technologies, makes this solution prohibitive.

In this paper, in order to address the issues relating to power-aware performance of caches, we propose a caching structure that addresses the following: 1. Definition of application-specific cache partitions as an aggregation of caching units (molecules). The parameters of each molecule namely size, associativity and line size are chosen so that the power consumed by it and access time are optimal for the given technology. 2. Application-Specific resizing of cache partitions with variable and adaptive associativity per cache line, way size and variable line size. 3. A replacement policy that is transparent to the partition in terms of size, heterogeneity in associativity and line size. Through simulation studies we establish the superiority of molecular cache (caches built as aggregations of molecules) that offers a 29% power advantage over that of an equivalently performing traditional cache.

1. Introduction

Chip Multi-Processors (CMP) have long been considered as the technology of the future and the recent introduction of commercial processors such as the Sun's UltraSparc T1 and Intel Pentium Duo stand to vindicate this. Applying Moore's Law to this phenomenon projects that the number of cores within a processor is bound to grow to 64 or higher in near future. This would mean that one processor would, in future, house as many processing entities as shipped in one cabinet of a HP Integrity Superdome server!

One of the typical CMP configurations as proposed by Olukotun et al [7] is that of multiple cores sharing a common L2 cache. Would such a design be scalable when used with 64 or more cores?

We conducted an experiment to see if the application miss rate varies when run concurrently with other applications. In particular, we considered 4 SPEC benchmarks viz. art, ammp, parser and mcf. These benchmarks were run concurrently in pairs and also with all four running concurrently to study the impact on miss rate for any given benchmark. The benchmarks were run concurrently on a CMP simulator [9]. The simulation was done on a 1MB 4 way shared L2 cache. Table 1 lists the miss rate of the benchmarks when run concurrently with other benchmarks and when run independently. The variations in the miss rate of the benchmarks are evident from Table 1. The miss rate observed by the benchmark (say parser), when run independently is very different from the miss rate when all four benchmarks are run concurrently. Also notice that the miss rate recorded is different for different benchmark pairs. Even though the reason for such variations may not be directly deduced, it does point towards the problem of inter-application interference, which in turn influences the application latency and throughput. Such inter-application interference is projected to be even more pronounced when the number of applications supported on a CMP platform increases with the increase in the number of processing cores. Increasing the cache size is potentially one of the ways of ameliorating this problem, however this comes at the cost of a higher energy bill. The power consumption needs to be contained since the energy density per area on multi-core processors is bound to be high due to enhanced processor

Table 1. Miss rate is dependent on the other benchmarks running concurrently on a CMP with shared cache. The simulations were performed on a shared 1MB 4way L2 cache.

1st app	2nd concurrently executing app	miss rate of app1	miss rate of app2
art	–	0.064	–
mcf	–	0.668	–
ammp	–	0.008	–
parser	–	0.086	–
art	mcf	0.069	0.691
art	ammp	0.065	0.009
art	parser	0.065	0.134
mcf	ammp	0.702	0.012
mcf	parser	0.684	0.247
ammp	parser	0.009	0.091
art	all four	0.734	–
mcf	all four	0.688	–
ammp	all four	0.013	–
parser	all four	0.253	–

activity, which could result in heat dissipation problems.

There is thus a need for an energy controlled technique to solve the problem of inter-application interference through efficient use of cache while still being able to meet application performance requirements (specified as various QoS metrics). Efficient utilization of the cache can then be achieved by defining a *miss rate goal* for each application in a way that could capture QoS requirements[1]. By defining separate caching regions per application, we avoid inter-application interference and ensure that the deviation from the miss rate goal is within tolerable limits. This technique of defining application-specific cache regions is called cache partitioning.

It is reported in [6] that small caches (8-32KB) are less power hungry than larger caches. In large caches the cache size to power relationship becomes non-linear at sub-100nm technologies. If such small sized caches could be used as building blocks then specific access to these will help contain the dynamic power dissipation in the cache. We call these low power building blocks as *molecules*. In this paper we propose *Molecular Caches*, a new cache architecture, where the cache is an aggregation of molecules (refer Figure 1). Molecular caches support selective enablement of molecules based on the application requesting access in order to reduce the dynamic power dissipation in the cache. We address issues relating to managing such an aggregation

[1]In this paper there is an assumption that the miss rate goal has been provided to us. The derivation of the miss rate goal is outside the scope of this paper and will not be discussed. Alternatively we use some default miss rate goals for purposes of experimentation.

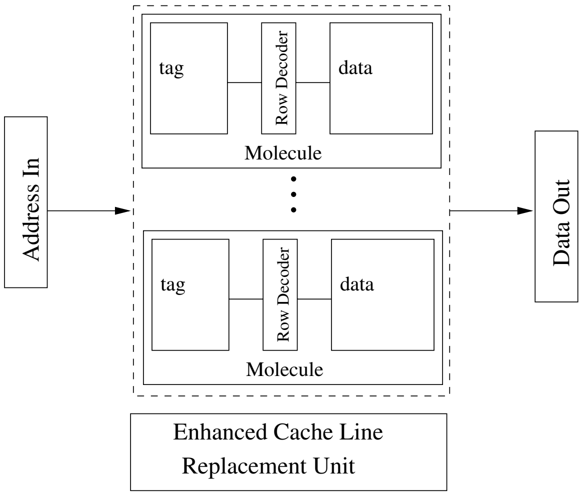


Figure 1. Molecular cache is composed of molecules

of molecules that serve as application-specific cache partitions, for which QoS metrics can be defined.

The paper is organized as follows: The related work section (refer section 2) delves into different types of cache QoS solutions proposed in the past and also takes a look at some of the cache partitioning solutions found in current literature. The details of the working of molecular caches such as handling cache lookup, miss handling etc. are discussed in greater detail in section 3. Section 3 also presents a partition sizing algorithm for molecular caches. Results of the various experiments carried out to establish the superiority of molecular caches are discussed in section 4. Section 5 summarizes the contributions and presents directions for future work.

2. Related Work

This section explains in brief some of the related work that has been carried out in the context of QoS in caches and Cache Partitioning.

Iyer [3] in a recent publication presented a framework for enabling QoS in shared caches. The paper makes a key observation that the current design of caches is more suited towards single application memory access and is not well suited for newer computing paradigms such as CMPs, Simultaneous Multi-Threading (SMT) and Specialized Cores. The paper then proposes a framework for Cache QoS management that includes three aspects namely Priority Classification, Priority Assignment and Priority Enforcement. The Priority Enforcement is of particular interest to us. For priority enforcement three different schemes are proposed. These include static and dynamic partitioning of caches, selective cache allocation and heterogeneous cache regions.

We only elaborate the technique for heterogeneous cache regions in this paper. In this technique, caches are composed of multiple caching units with different organizational structures and policies. The problem is that of assigning the right application to the appropriate cache structure, keeping in mind the priority of the application and its memory access pattern. The results from the paper indicates that heterogeneous cache regions help reduce dedicated cache size.

Cache Partitioning has been greatly researched, even before the arrival of CMPs. Of the several flavours of cache partitioning, multi-application dynamic partitioning techniques are of particular interest to us. Single application partitioning for low power in the context of direct mapped caches is reported in [8]. In this section we discuss examples of both the hardware controlled and the software controlled variants. Several solutions have been proposed that address the issue of cache pollution through the use of NUCA architectures. Techniques such as Victim Replication [14], NuRAPID [1], CMP-NuRAPID[2] address several different aspects of these. However none of these use application QoS parameters as the guiding principle. They attempt to reduce the miss rate experienced by a processor.

Suh et al [11] propose the use of two different techniques for implementing partitioned caches namely Modified LRU and Column Caching. In Modified LRU scheme, the replacement decision depends on the number of cache blocks already allocated to a particular process. If the process has not exceeded its predefined space threshold, a global replacement is performed, else a local replacement is performed. The column caching approach restricts some processes to place data in some 'columns' (i.e. ways) of a multi-way associative cache. Their results indicate that for a cache of size 1-2MB the hit rate improves by 40% as compared to standard LRU for a time quantum of 200,000 memory references and weighting factor of 0.5 (which is used to give preference to recent measurements as opposed to older ones). Compared with the standard LRU, relative IPC improvements of 13.29% for 2 processes on a 2MB cache and 14.21% for 4 processes on 2 MB cache are recorded. Kim et al [5] propose a variant of the Modified LRU technique where fairness is taken as a measure of cache allocation. As indicated in [5], cache pollution violates operating system expectation that all scheduled entities get an equal chance to utilize the processor, hence fairness of allocation is required. We take this argument forward, since fairness weighted by application's priorities is far more important. Yet another technique that takes fairness into consideration was proposed by Yeh et al [13]. The approach used in this technique is very similar to our approach. The authors propose a phase-based approach for cache partition resizing.

Kim et al [4] proposed a partitioning technique in which different banks of a multi-banked cache are allocated to different processes. First a search of the 'home' bank is performed, failing which the entire cache is searched in a set-associative manner. Called Process Ownership-based Cache Architecture (POCA), this cache contains four components viz. Current Set Number, Set Assignment Table, Data Enable Controller and the Victim Process Table. The authors [4] used ATUM traces to evaluate the cache. Simulation performed on cache of size from 8KB-1MB with 4 banks indicate that the hit rate of the cache is higher than a direct mapped cache of equal size and is either as good or better than a 4 way associative cache of equal size. The cache access time of POCA is 10% better than an equivalent associative cache.

Is there a need for something better than the state of the art? Suh et al's [11] proposed cache partitioning solution does not look into the dimension of heterogeneous cache regions that can potentially improve the efficiency of the cache usage. A major drawback of their cache architecture is the reliance on multi-way associative caches. Multi-way associative caches tend to consume a lot of power and have longer access time. Hence associativity cannot be increased beyond a certain value, since the power consumption increases exponentially with increase in associativity (refer [6]). Kim et al's [4] work is a software based resizing solution and relies on operating system based information for determining the victim sets. Such a scheme would not be practically realizable due to the non-standard ways of obtaining the required information from the large variety of operating systems available today.

The work on heterogeneous cache regions by Iyer [3] is based on the use of multiple caches each with different cache parameters and the optimal assignment of applications to appropriate caches with suitable cache parameters. This technique would only enable some applications to run efficiently. We extend this work by marrying the ideas of dynamic partitioning and heterogeneous cache regions to create more effective dynamic partitions that are dynamically customized to the application's needs. We create caching structures that allow application dependent dynamic creation and reconfiguration of heterogeneous cache regions with different associativities and line sizes. The capacity of these regions too can be modified. The cache regions are created from homogeneous building blocks that are direct mapped. These cache regions also support non-uniform line associativity.

3. Molecular Caches

Molecules are small sized (8-32KB) direct mapped caching units that have 64 byte wide lines. This choice of size and associativity enables reduction in energy dissipation as reported by Mamidipaka et al [6]. Larger structures are created from these building blocks by forming aggre-

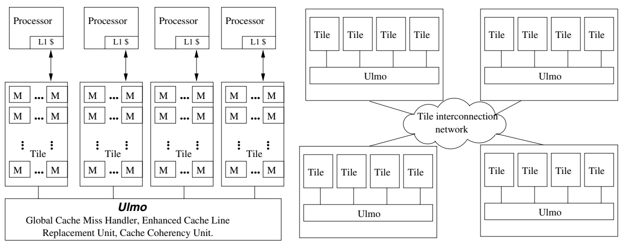

Processor
L1 $
M
Tile
Ulmo
Global Cache Miss Handler, Enhanced Cache Line Replacement Unit, Cache Coherency Unit.
Tile interconnection network

Figure 2. *Tiles* - A physical organization of molecules. Here 'M' is used to indicate a molecule. Tiles with physical proximity are grouped to form clusters. Each cluster has one *Ulmo*. The interconnection network is shown as a cloud since there is no assumption made on the topology of the interconnections.

gations. 32-256 molecules are grouped together with one read/write port and this group is called a *tile* (Figure 2). Every processor is statically assigned to a tile and all requests from that processor are forwarded to that tile. The processor-tile assignment can be made non-static by allowing the processor-tile mapping to be changed during a context-switch. 4-8 tiles are grouped together into a *tile cluster* (Figure 2). Every tile cluster is associated with one tile controller called *Ulmo*[2]. *Ulmo* handles tile-misses and the coherence traffic between the tile clusters. The tile clusters are connected through an interconnection network to enable coherence transactions. A subset of the molecules of the molecular cache are used to form a cache region/partition. The cache regions are assigned to applications for exclusive use.

3.1. Creating Cache Partitions

The cache regions that have been created need to be assigned to applications for exclusive use. To tie a region to a specific application, each molecule of the region is configured with the Application Space Identifier (ASID). The ASID uniquely identifies a running application. Before any cache operation is performed on the molecules, an ASID match is performed to see if the molecule is eligible to perform the operation (refer Figure 3). An additional stage is introduced in the address decoding logic of the molecule to compare the ASID of the requestor with the configured ASID of the molecule. Only comparisons resulting in a TRUE condition allow the molecule to proceed to the subsequent stages of data retrieval. The introduction of the ASID comparison would increase the number of cycles consumed by an additional cycle.

Addition of molecules to a cache region is done by configuring the ASID of that molecule to match the ASID of the application that wields exclusive control over that cache region. If the shared bit is set (refer Figure 3) then the molecule is accessed for all requests on that tile, irrespective of the ASID of the running application.

[2]Unlimited Molecules

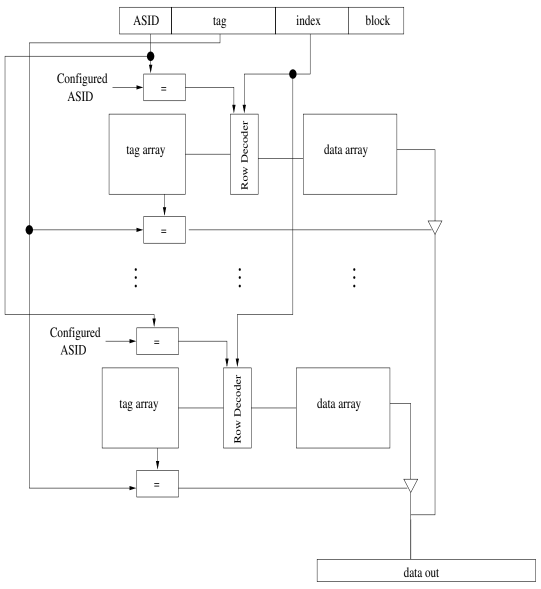
3.2. Varying the Line Size

Increasing the line size helps in reducing the cache miss rate in case of high spatial locality. Let the line size of a molecule be l. If a line of size $2 * l$ is desired, the cache retrieves 2 lines whenever a cache miss occurs. Two lines are treated as a single unit of replacement. Both the lines are stored in consecutive lines of the same molecule. All cache hits still use l as the operating cache line size. Cache misses trigger retrieval of more than one cache line. A cache region may support only one line size. The line size of the region may be modified only at the time of creation.

3.3. Replacement and Lookup

There are three basic ways of performing line replacement namely FIFO, Random and LRU. Among these, LRU is known to be the best followed by the Random replacement policy. A similar scheme can be used to select a molecule in which the new line is placed. Implementation of an approximate LRU scheme would be very expensive as it needs to keep track of which molecules are in use and their corresponding order of usage. Instead, we adopt Random replacement as one of the preferred scheme for replacement in Molecular Cache. The ability of the random replacement algorithm to distribute the load equally across all molecules is highly dependent on the entropy of the random number generator implemented in hardware.

The primary purpose of associativity in a traditional cache is to reduce the conflict misses. However the traditional cache provides a common associativity for all cache lines. But a common associativity may not be the best suited for all the applications. The associativity requirements of a video streaming application performing computations on a number of macro blocks is very different from the associativity requirements for a sparse matrix based linear solver. In traditional caches different associativities cannot be supported for different lines because of complexities involved in the output selection logic, during cache access. The use of identical structures for cache access and cache replacement complicates supporting different associativities along different lines. In order to be able to support different associativities, in molecular caches, across different lines the access view of the cache is separated from the replacement view of the cache. The access view of the cache is the same as the physical organization of the cache (refer Figure 4). The replacement logic views the molecules as a 2-D sparse matrix (Figure 4). Every row of the matrix must contain



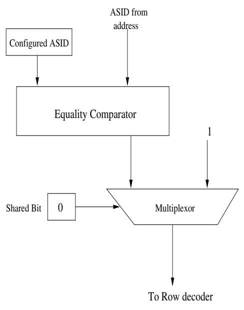


Figure 3. Figure on the left shows the different steps in cache access in the molecular cache and the figure on the right is the ASID comparator block diagram.When the shared bit is unset the multiplexor selects the output of the comparator. When the shared bit is set then the multiplexor always return 1, thus enabling the row decoder to proceed with the lookup operation.

atleast one molecule. The number of molecules per row need not be uniform. The number of rows and columns can be varied dynamically. Such a view leads us to a new molecule selection policy, which we call, *Randy* replacement algorithm. In the Randy replacement scheme, choosing a molecule for replacement is accomplished by:

1. determining the row to which the replacement is performed using the expression:

   $$row = \lfloor \frac{address}{size_{molecule}} \rfloor mod\ row_{max}$$

   where $address$ is the physical address, $size_{molecule}$ is the size of each molecule and row_{max} is the maximum number of rows in the replacement view (refer Figure 4). The maximum way size (i.e. row_{max}) is found along the first column.

2. One molecule in the row is chosen at random to perform the replacement.

The Randy replacement scheme reduces the reliance on random numbers, at the same time provides an opportunity to the resizing algorithm to add molecules only along those rows where the miss count[3] is higher. The comparison between the Random and Randy replacement algorithms is presented in the results section (section 4).

[3]Explained in greater detail in section 3.4

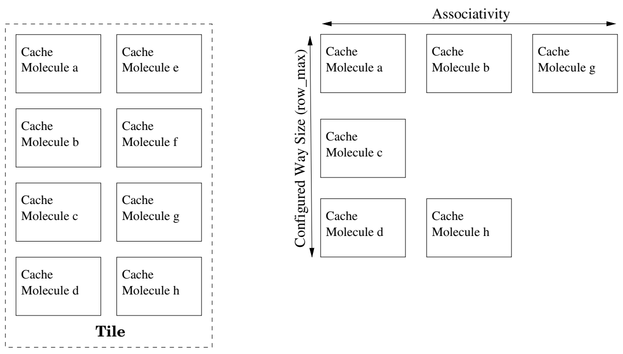


Figure 4. The figure on the left shows the Physical organization of the molecules contained in a cache region. The molecules a to h are contained in the tile. The figure on the right shows the replacement view of a cache region. Heterogeneity in different regions is achieved through varying number of molecules and their arrangements. Note that the physical arrangement of the molecules a to h has no bearing on the replacement view of the same.

The use of adaptive associativity and variable sized partitions complicates the cache lookup operation, since the data may be found in any molecule belonging to the cache region owned by the application. This requires that all the molecules of the region are searched for the data. In order to minimize the energy dissipated on account of an all molecule lookup, a hierarchical search is performed. The tile associated with the requesting processor is searched first. Only those molecules whose ASID matches the requesting application's ASID are accessed. If the data is not found in the tile then other tiles in the tile-cluster need to be searched. This task is performed by *Ulmo*. *Ulmo* searches only those tiles whose molecules have contributed to the application's cache region.

3.4. Dynamically Resizing Cache

The primary aim of cache partitioning is to be able to eliminate the inter-application interference, while still being able to meet the miss rate goal of the individual partitions. Such a *goal based* scheme would prevent some applications from occupying large portions of the cache at the cost of performance of other applications. The miss rate goal is defined per application. The miss rate goal can be different for different applications based on relative priority of applications. Alternatively, a common miss rate goal can be defined for all applications.

Ground Zero The choice of size of initial partition can be either a very small number of molecules (say 2 molecules) or a large number of molecules (say 32 molecules). User-driven/Profile-driven directives such as "small", "typical" and "large" cache usage patterns can be used to suitably modify the initial allocation. Periodic resizing is performed with the objective of meeting the miss rate goals per application. The process of resizing can either increase or decrease the size of the partition. Our experiments indicate that when small initial partition size is used frequent repartitions are required during the initial phases in order to reduce the application miss rate. Frequent resizing is not favored since it would incur the cost of computation of the new configuration. In the current scheme each partition is provided with half the number of molecules contained in a tile in the beginning and then periodically the molecules are removed or added based on the prevailing miss rate and the miss rate goal.

When to add? The resizing period is specified in terms of the number of addresses serviced by the cache. The periodicity of resizing can be either constant or adaptive. In the constant address scheme, resizing is performed once the user specified address count is met. In the adaptive scheme the first resize is performed at the user specified address count. The next point at which resizing is triggered is determined based on the prevailing miss count and the miss rate goal. In the case that the miss rate is higher than expected then the address count for resizing is brought down to 10% of the original value. If the miss rate is well within acceptable limits then the resize trigger count is doubled. If the resize trigger count is determined by the overall miss rate of the cache then it is called a global adaptive scheme and if it is triggered by the application miss rate it is called a per-application adaptive scheme. Our experiments indicate that a constant address count resizing does not aid in bringing down the miss rate. Adaptive schemes perform better than constant address schemes. The observations from actual experiments indicate that global adaptive scheme performs well with small-sized tiles and per-application adaptive scheme works better with larger tile sizes ($>=$ 2MB). The global adaptive scheme aids better in maintaining cache miss goals, as opposed to the per-application adaptive scheme in small caches, since resizing happens whenever the global miss rate increases.

Where to add? An effective resizing algorithm would

- Remove only those molecules from the cache that would have least impact on the hit rate of the partition. In order to implement such a scheme we associate counters with a molecule or a group of molecules. The counter records the number of misses that lead to line replacements in the molecule or group of molecules. The molecule that has the least count is withdrawn, since it holds the least number of addresses. Other effective schemes such as LRU stack, counters with cold miss compensation etc. can be used. The actual evaluation of the resize algorithms based on these techniques is outside the scope of this paper. In the context of this presentation we adopt the following:
  - When Random replacement scheme is used per-molecule counters are employed.
  - When Randy replacement scheme is used counters per row are employed.
- Add molecules in a way such that the miss rate is decreased substantially using the least possible number of molecules. Once again the miss counters indicate where the highest activity has been seen. The counter values can be used to increase the associativity for that group of molecules with the highest recorded miss counts.
  - When Random replacement algorithm is used all molecules can be visualized as placed one behind the other (i.e. in a single row). Any new addition of molecules simply increases the associativity of the arrangement.

- When the Randy replacement algorithm is used then the molecules are added along the rows with the highest miss count. The rows that handle a larger amount of misses need more associativity in order to prevent conflict misses, thereby effecting variable way sizes within a given region.

The additional molecules required for increasing the size of the partition can be either obtained from the tile in which the cache region is being currently hosted or from other tiles in the tile-cluster.

How much to add? The number of molecules allocated during resize can be either one molecule or a large chunk required to meet the miss rate goal. Single molecule increments are less effective and require frequent resizing to be performed. Refer Algorithm 1 for details of how the increments and decrements are computed.

Who does the computation? Instead of employing a separate hardware for computation required to resize a partition, the resizing function can be scheduled periodically on one of the processors through an OS level daemon. The daemon periodically reads the information related to the partition and then computes the new configuration of the partition. The $resize()$ function takes about 1500 cycles per application. Given a recomputation count of about 25,000 memory references[4], the data for processing could be picked up at the end of 25,000 references and size changes effected once the computation is complete. The computation needs to be distributed across several processors for different tile clusters.

4. Results

We evaluate the performance of molecular caches using SESC [9], a CMP simulator, to run the various workloads. The L1-Data misses were recorded and the traces were used as input to a modified version of Dinero, to simulate the behavior of molecular caches. The Power results were obtained using CACTI [12].

In order to evaluate the performance the average deviation from the miss rate goal is measured. Our initial workload was a SPEC based workload consisting of 4 benchmarks namely ammp, parser, art and mcf. The choice of benchmarks was based on experimental data that confirmed their sensitivity to L2 parameters, viz. cache size and associativity, for performance. The simulations were performed on a trace containing about 3.9 million references. The change in average deviation from miss rate goal with a

[4]This was determined through experimentation as being a large enough count at which the miss rate remains acceptable.

```
for every application partition do
    compute partition miss rate;
    if miss rate > 50% then
        if max allocation > last allocation then
            max allocation = last allocation;
        end
        resize (max allocation);
    else
        if miss rate < miss rate goal then
            /* Withdraw molecules more
               slowly than you add -
               Conservative                 */
            temp = sqrt((current molecules * miss rate)
            / miss rate goal);
            withdraw (temp, current partition);
        else
            if miss rate < last miss rate then
                /* Using a Linear
                   relationship between
                   Cache Size and Miss
                   Rate.  Simplifies
                   Computation!             */
                . temp = (current molecules * miss
                rate) / miss rate goal;
                /* Do not allocate more
                   than the maximum
                   allowed in one chunk     */
                if temp > max allocation then
                    temp = max allocation;
                end
                resize (temp, current partition);
            end
        end
    end
end
if current overall miss rate < miss rate goal then
    resize period = 2 * resize period;
else
    resize period = 0.1 * resize period;
end
```

Algorithm 1: The Function used to determine the number of molecules by which the partition should be expanded or shrunk

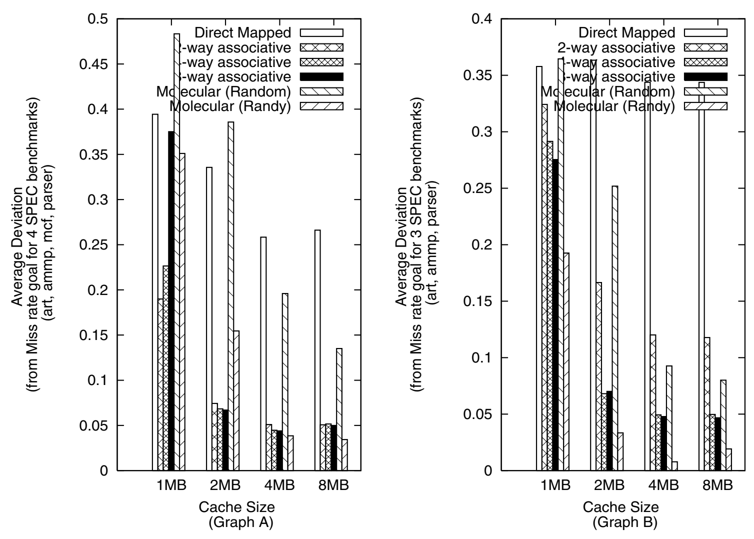


Figure 5. Graph indicating the average deviation from the miss rate goal for the SPEC workload. In Graph A the miss rate goal was set to 10% for the four benchmarks. In graph B the miss rate goal was set to 10% for 3 out of the 4 benchmarks.

change in cache size and associativity are captured in Figure 5. The plots indicate the average deviation from the miss rate goal for various associativities, for different cache sizes. The graphs have been plotted for Direct Mapped, 2 Way, 4 Way and 8 Way associative caches of sizes 1MB, 2MB, 4MB and 8MB. The results have also been plotted for Molecular cache with 4 tiles for the cache sizes 1MB (tile size 256KB), 2MB (tile size 512KB), 4MB (tile size 1MB) and 8MB (tile size 2MB). The graph shows the results plotted for both Random and Randy replacement algorithms. Graph A plots use a miss rate goal of 10% for all the four benchmarks. Graph B plots use a miss rate goal of 10% for the ammp, parser and art only.

For traditional caches, the graphs indicate that the average deviation decreases with the increase in cache size and associativity. The average deviation of the molecular cache falls sharply at 4MB in graph A and at 2MB for graph B. For a specific miss rate goal, molecular caches have a threshold size at which they are effective. The thresholds for graphs A and B are 4MB and 2MB respectively. The sudden drop in deviation, at these thresholds, is because of an increase in molecule availability at higher cache sizes. Since all the applications are cache intensive there are phases during which no free molecules are available and no resizing takes place.

In another experiment we used a mix of 12 benchmarks from SPEC, Netbench and Media bench. The benchmarks include crafty, gcc, gzip, parser, twolf, CRC, DRR, NAT, CJPEG, decode and epic. The molecular cache simulation was done using 3 tile clusters, each with 4 tiles. Each tile cluster was 2MB in size (i.e. tile size of 512KB). The miss rate goal was set at 25%. For the molecular cache simulation, the applications were divided into three groups, without giving consideration to the nature of the mix (since real world applications could have any combination of applications running simultaneously). Each group was assigned a tile cluster. The performance of the 6MB molecular cache (2MB per cluster) was compared with the performance of 4MB 4 way, 4MB 8way, 8MB 4way and 8MB 8way caches. Table 2 indicates the average deviation from the miss rate goal for all the caches. The results indicate that the 6MB molecular cache performs even better than the 8MB 8way cache. The reason for this better performance is the two level isolation of the applications. One, each partition isolates the address space of various applications. Two, the separation into different tile clusters reduces the contention for molecules among the different applications (since all molecules in a tile cluster can be used by all application

Table 2. Table of Average Deviation from the Miss Rate Goal

Cache Type	Average Deviation
4MB 4way	0.313261
4MB 8way	0.309515
8MB 4way	0.246843
8MB 8way	0.243161
6MB Molecular Randy	0.222075
6MB Molecular Random	0.356923

Table 3. The cache configurations for which power consumption was evaluated.

Parameter	Molecular Cache	Traditional Cache
Total Cache Size	8MB	8MB
Molecule Size	8KB	–
Tile Size	512KB	–
No. of tile-clusters	4	–
No. of tiles per cluster	4	–
No. of Read-Write ports	1 per tile cluster	4
Associativity	adaptive	DM, 2, 4, 8

Table 4. Results of CACTI simulations. Cacti executions were performed for 0.07μ technology.

Cache type	Freq (MHz)	Power (W)	Size of mol. cache	mol. power worst case	mol. power (average mixed workload)
8MB DM	199	4.93	8MB	5.29	4.85
8MB 2way	205	5.95	8MB	5.45	4.99
8MB 4way	206	7.66	8MB	5.46	5.0
8MB 8way	96	3.58	8MB	2.55	2.34

running on that cluster).

CACTI [12] was used for power simulations, since we are only interested in the dynamic power consumption. The leakage power consumption remains unaffected in molecular cache. The power results reported for the molecular cache is the worst case power consumption. The worst case/highest power consumption happens when all the molecules of a tile are enabled (based on the ASID match). An application normally uses a subset of the molecules of the tile, hence would consume lower power in practice. The power computation for the molecular cache is approximated as the power consumed by all the molecules of a tile.

The configurations of the molecular and traditional caches for which power evaluations were performed are listed in Table 3. Table 4 compares the power consumed by the traditional cache and the molecular cache at the frequency of operation of the traditional cache. The power and the frequency values are derived from the energy and cycle time reported by CACTI. The power consumed by a molecular cache is computed using the energy reported for a molecular cache at the frequency of the traditional cache to which the molecular cache is being compared. The power consumed by the 8MB molecular cache is slightly higher than the power consumed by an 8MB traditional direct mapped cache with four ports. The selective access and hierarchical search space of the molecular cache enables reduction in the dynamic power consumption in the cache.

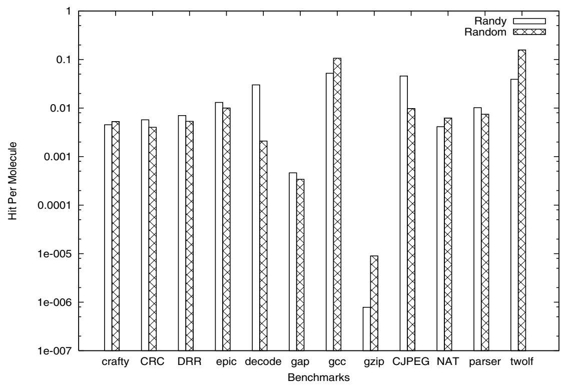


Figure 6. A graph comparing the hit rate contributions per molecule for Random and Randy replacement algorithms.

The replacement algorithms namely Random and Randy were compared. The most obvious criterion for evaluating the replacement algorithm is the miss rate. The other criterion that needs to be used is the number of molecules employed to achieve the given hit rate. The replacement scheme that achieves a lower miss rate with a lesser number of molecules is more effective than a scheme that uses more molecules to achieve the same miss rate. To evaluate the effectiveness of each of the Random and Randy schemes, we compute the hit rate per molecule (HPM). The graph in Figure 6 compares the HPM for the replacement algorithms.

The graph is a plot of HPM recorded for different applications of the mixed workload. Note that the Y-axis is on a logarithmic scale. Barring four applications, namely gcc, gzip, NAT, and twolf, the HPM of Randy replacement algorithm is higher than the HPM for Random replacement algorithm. This anomaly needs to be interpreted in light of the overall miss rate and overall molecule usage. The overall miss rate of Randy replacement algorithm is 9% lower than the miss rate of Random replacement algorithm, while the Randy replacement algorithm uses 5% more molecules when compared with Random replacement algorithm. This 5% increase in molecular assignment is due to an attempt

Table 5. The power-deviation product for each of the caches for which performance results were reported in Table 2. The power for the 6MB molecular cache is computed for the same frequencies as listed in Table 4. The power-deviation table for molecular caches is only reported for the Randy replacement algorithm.

Cache Type	Power-Deviation Product	Power-Deviation Product of Mol. cache
8MB 4way	1.890	0.909
8MB 8way	0.870	0.425

to further bring down the miss rate. The increase in allocation does not lead to a commensurate drop in miss rate (as is characteristic of the cache miss rate behavior in convex regions [10]). This increase in the number of molecules for a small gain in hit rate causes the anomaly observed in the graphs.

We have defined a new metric to measure the QoS in conjunction with the traditional parameters. We use the **power-deviation product** to measure the effectiveness of the cache in meeting the QoS while still being able to keep the cache power consumption in check. Table 5 presents the power-deviation product for the mixed workload. The table indicates that the Molecular Cache using Randy replacement algorithm performs consistently better than the traditional cache.

5. Future Work and Conclusion

In this paper we presented a new cache architecture called Molecular Cache. Molecular Caches solves the problem of Cache pollution in a power efficient manner through the use of Application-specific regions with adaptive associativity, variable line size and adaptive partition sizes. Selective cache access and hierarchical lookup enable reduction in dynamic power dissipation.

The work presented in this paper needs to address the following issues related to its practical realization: The resizing algorithm needs to be further refined through better techniques for identifying locations for addition and withdrawal of molecules. A different scheme for replacements such as an LRU-Direct scheme needs to be evaluated. The metric for QoS based cache evaluation - power-deviation product - needs to be further refined.

Acknowledgments The authors would like to express their gratitude to Prof. V.Rajaraman of Supercomputer Education and Research Centre, IISc for all his valuable inputs. The authors also thank Ranjani Narayan for her useful suggestions.

References

[1] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 55, Washington, DC, USA, 2003. IEEE Computer Society.

[2] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in cmps. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 357–368, Washington, DC, USA, 2005. IEEE Computer Society.

[3] R. Iyer. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 257–266, New York, NY, USA, 2004. ACM Press.

[4] D. Kim, J. Lee, and S. K. Park. A partitioned on-chip virtual cache for fast processors. *J. Syst. Archit.*, 43(8):519–531, 1997.

[5] S. Kim, D. Chandra, and Y. Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *PACT '04: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 111–122, Washington, DC, USA, 2004. IEEE Computer Society.

[6] M. Mamidipaka and N. Dutt. ecacti: An enhanced power estimation model for on-chip caches. Technical Report TR-04-28, Center for Embedded Computer Systems, University of California, Irvine, September 2004.

[7] K. Olukotun and L. Hammond. The future of microprocessors. *Queue*, 3(7):26–29, 2005.

[8] P. Petrov and A. Orailoglu. Towards effective embedded processors in codesigns: customizable partitioned caches. In *CODES*, pages 79–84, 2001.

[9] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC simulator, January 2005. http://sesc.sourceforge.net.

[10] A. J. Smith. Cache memories. In *ACM Computing Surveys, vol. 14, no. 3*, pages 473–530, 1982.

[11] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic partitioning of shared cache memory. *J. Supercomput.*, 28(1):7–26, 2004.

[12] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model, 1996.

[13] T. Y. Yeh and G. Reinman. Fast and fair: data-stream quality of service. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 237–248, New York, NY, USA, 2005. ACM Press.

[14] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 336–345, Washington, DC, USA, 2005. IEEE Computer Society.