MEMORY HIERARCHY ISSUES IN PROCESSING LARGE DATA SHEETS

Μεταπτυχιακή Διατριβή

Tziouvaras Athanasios

Μεταπτυχιακή Διατριβή για την απόκτηση του Μεταπτυχιακού διπλώματος Ειδίκευσης «Επιστήμη και Τεχνολογία Υπολογιστών, Τηλεπικοινωνιών και Δικτύων» του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Μεταπτυχιακών Σπουδών του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

……..

Τζιουβάρας Αθανάσιος
Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Ευχαριστίες

Με τη συγγραφή και ολοκλήρωση της διπλωματικής αυτής θέλω να ευχαριστήσω τον καθηγητή Γιώργο Δημητρίου για όλη την βοήθεια που μου προσέφερε και για την υπομονή του κατά την διεκπεραίωση του εγχειρήματος αυτού καθώς η καθοδήγηση και οι γνώσεις του αποδείχτηκαν αναγκαίες για την ολοκλήρωσης της εργασίας αυτής. Επίσης τους καθηγητές Σταμούλη Γεώργιο, Ευμορφόπουλο Νέστορ και Τσομπανοπούλου Παναγιώτα για την στήριξη και την βοήθειά τους στο έργο μας.Επιπλέον θα ήθελα να ευχαριστήσω τους φίλους και τους γονείς μου που στάθηκαν δίπλα μου των οποίων η στήριξη τους ήταν ένας από του λόγους που κατάφερα να ολοκληρώσω τον κύκλο των μεταπτυχιακών μου σπουδών. Τέλος ευχαριστώ και το πανεπιστήμιο Θεσσαλίας που σαν φορέας μου έδωσε τις γνώσεις, τις εμπειρίες και τον τεχνικό εξοπλισμό για τις ανάγκες της διπλωματικής.

Table of contents

Abstract

The evolution of modern computer systems is highly depended on the needs of processing power of the industry. Thus when the current processing power needed is more than the existent one, the processor industry is pushed to deliver the required performance. This relationship works both ways as the higher performance processors support, the better opportunities for the designers to develop more complex and performance hungry applications. This dependence of processing power and emerging applications led to the creation of advanced computer architectures able to achieve high performance on need. Due to this advancement, leading technologies in data processing have advanced to a point that the amount of data processed is more than a few petabytes.

This new type of data processing managed to create a new engineering field of interest know as big data. A more accurate definition of big data would be a large collection of data sets so large and complex that it becomes difficult to process using on-hand data management tools or traditional data processing applications. The challenges include not only the processing of data but also the storage, shorting and visualization of them. Because of this huge amount of existent data, the processing power needed to deliver to expected results is huge. For that reason the processing of such data sets is usually being done in distributed systems which are consisted of many separated processor nodes running in parallel in a network directed by a master node.

In this work we focus on the memory hierarchy of the cache on each processor which is part of a larger network of nodes processing big data. The nature of the algorithms used for distributed processing is largely different than the common algorithms used for a single core or multicore system. What is more the amount of memory needed in such computations is not necessarily larger than any other applications, but the memory access patterns change vastly depending on the specific task. Thus a unique cache memory specifically designed to improve the performance of distributed processing should be more effective than any casual cache memory design for general purpose systems. The basis stated above in conjunction with the fact that the big data problem is rather new to the industry which lacks of an efficient way to handle this amount of data is the contribution of this work.

1. Introduction

1.1 Big data definition

Big data is a term used to describe the exponential growth and availability of data, both structured and unstructured. And big data may be as important to business – and society – as the global web has become as more data may lead to more accurate analyses. Big Data is a notion covering several aspects by one term, ranging from a technology base to a set of economic models. In this study, the following definition of Big Data will be applied: "Big Data" is a term encompassing the use of techniques to capture, process, analyze and visualize potentially large datasets in a reasonable timeframe not accessible to standard industry computer technologies.[3] By extension, the platform, tools and software used for this purpose are collectively called "Big Data technologies". Big Data is not a new concept, and can be seen as a moving target linked to a technology context. The new aspect of Big Data lies within the economic cost of storing and processing such datasets; the unit cost of storage has decreased by many orders of magnitude, amplified by the Cloud business model, significantly lowering the upfront investment costs for all businesses. As a consequence, the "Big Data concerns" have moved from big businesses and state research centers, to a mainstream.[2,3,4]
Today Big Data relates to data creation, storage, retrieval and analysis that are remarkable in terms of volume, velocity, variety, variability and complexity[4]:

- **Volume.** Many factors contribute to the increase in data volume. Transaction-based data stored through the years. Unstructured data streaming in from social media. Increasing amounts of sensor and machine-to-machine data being collected. In the past, excessive data volume was a storage issue. But with decreasing storage costs, other issues emerge, including how to determine relevance within large data volumes and how to use analytics to create value from relevant data.
  Overall volume is the most visible aspect of Big Data, referring to the fact that the amount of generated data has increased tremendously the past years.

- **Velocity.** Data is streaming in at unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time. Reacting quickly enough to deal with data velocity is a challenge for most organizations. Overall velocity is the aspect which captures the growing data production rates as more and more data are produced and must be collected in shorter time frames[11].

- **Variety**. Data today comes in all types of formats. Structured, numeric data in traditional databases. Information created from line-of-business applications. Unstructured text

documents, email, video, audio, stock ticker data and financial transactions. Managing, merging and governing different varieties of data are something many organizations still grapple with. Overall with the multiplication of data sources comes the explosion of data formats, ranging from structured information to free text.

- **Variability.** In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Daily, seasonal and event-triggered peak data loads can be challenging to manage.
- **Complexity.** Today's data comes from multiple sources. And it is still an undertaking to link, match, cleanse and transform data across systems. However, it is necessary to connect and correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out of control.

In this study we explore not only the memory needs of big data applications but also the design options of a cache able to handle the read and write requests with efficiency and low miss rates. This dissertation is organized as follows: In this section we will further analyze the way that both software and hardware address the big data while in the next chapter we are to make an introduction for the reader to the basic concepts of grid computing. In chapter 3 we are describing the organization and hierarchy of cache memories as they have evolved till today and we make an introduction to the basics of adaptive cache. In chapter 4 we present our proposal, a new schema for an adaptive cache specifically design to handle big data traffic and we explain its functionality and architecture. Afterwards, in chapter 5 we show the results of extensive testing and benchmarking of the newly designed cache. The tests were made using various memory access patterns and different memory workloads. Finally in chapter 6 we look into the future of this project discussing on how it could be further enhanced, improved and get more genuine.

1.2 How software addresses big data

In computer engineering terms, when big data is addressed, distributed computing is also involved [12]. Distributed computing describes a closed network of interconnected processors which work together in parallel in order to achieve a specific task such as data processing. Each processor usually is unaware of the work distributed to other processor/nodes but its efficiency is directly influencing the performance of the whole network. Such networks usually achieve high amount of processing power as clustering processing nodes has been proven an efficient way of processing information.
That process is differently addressed by software and hardware designers. Software designers focus their efforts in creating algorithms which are able to distribute the total workload effectively among the existent resources while maintaining stability and fault tolerance. From such efforts the map-reduce programming model has emerged which has prevailed in distributed system processing. A Map Reduce program is composed of a Map procedure that performs

filtering and sorting and a Reduce procedure that performs a summary operation. The "Map Reduce System" orchestrates the processing by marshaling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance. Moreover the Map Reduce framework is able to parallelize the problem across huge datasets using a large number of nodes and is able to take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted.
Although there are many algorithms based on Map Reduce framework, each one of them follows the same basic principle. This principle is a baseline for each algorithm which is used in order to fulfill the criteria of the framework. There are five steps of computation which take place during the execution of an algorithm.

1. **Prepare the Map input** – the "Map Reduce system" designates Map processors, divides the input into smaller parts which are immediately assigned to each processor in a way that the work is almost equally distributed.
2. **Run the user-provided Map code** – the user code is executed in each processor specified during the step 1.
3. **"Shuffle" the Map output to the Reduce processors** – the Map Reduce system designates Reduce processors, which are to collect the results produced by the working processors and assigns to each one of them a part of the code to be executed for the collection.
4. **Run the user-provided Reduce code** – Reduce code is run exactly once for each result to be collected.
5. **Produce the final output** – the Map Reduce system collects all the Reduce output, and sorts it to produce the final outcome.

Usually in Map Reduce system the first step is serialized while the steps 2-5 can be executed in parallel speeding up the information processing. Thus while the preparation of map input is executed by a single node (called mapper), the other steps are equally divided among the existing node of a network (called workers). Many programming languages such as R, and programming frameworks (such as apache hadoop) have been developed according to Map Reduce concept of compartmentalization of work among a network of processing nodes. Those frameworks although new are widely used by developers who wish to processes large data sets.

1.3 How hardware addresses big data

The hardware approach to big data is neither so direct nor as efficient as the software approach. The network consisted of processor nodes which are used for code execution is nothing more

than a cluster of general purpose processors interconnected together with some type of interconnection network. Although general purpose processors are able to deliver the expected performance, the performance scaling of such a network is still an issue. Currently the amount of data processed is large, but not large enough to hinder the code execution time on each processor. As time will pass the need for more and more data to be processed at once will emerge, getting to a point that the required performance cannot be achieved by just adding another processor (or a number of processors) in the cluster. This is the primal concern of computer architects [15,16,17] designing high performance systems as scaling is a completely different issue from performance[12,13,14]. Nowadays we have come to a point where the performance needed is achieved but the scalability of such a network is smaller than expected, making it difficult to maintain the current performance while processing even larger data sets.

Although the problem is visible, not many research results have been shown to dissolve it. While most researchers focus on developing hardware accelerators for the Map Reduce algorithms [1,5,7,8,9,10]which are to deliver better and more scalable performance, we show the problem from a different angle. As each processor on the network usually is fed with different inputs at any rate, its cache memory will not be performing at its maximum efficiency. This happens because when big data is involved, memory access patterns are getting unpredictable, filling the cache with data which will never be used again and will be evicted for the next data to arrive. The approach we made on this problem was the creation of a cache simulator which simulates an adaptive cache hierarchy, specifically designed for adaptation to different memory access patterns that fit in the big data problem.

While most researchers are focus on computer architectures able to support the processing of large data sets, we manage to approach the same problem in a different way by looking at a very crucial part of modern processors, the cache memory. An adaptive memory would not only increase the gained performance but also make the whole cluster of processors more scalable, as it is explained in next sections of this dissertation[22].

2. Grid computing

The term grid is referring to a distributed computing infrastructure, able to provide resources based on the needs of each client. Grid technology can largely enhance productivity and efficiency of virtual organizations, which must face the challenges by optimizing processes and resources and by sharing their networking and collaboration. Grid computing technology is a set of techniques and methods applied for the coordinated use of multiple servers. These servers are specialized and works as a single, logic integrated system. The grid has developed as a computing technology that connects machines and resources geographically dispersed in order to create s virtual supercomputer. A virtual system like this is perceived like it has all the computing resources, even if they are distributed and has a computing capacity to execute tasks that different machines cannot execute individually. In the past few years grid computing is defined as a technology that allows strengthening, accessing and managing IT resources in a distributed computing environment. Being an advanced distributed technology, grid computing brings into a single system: servers, databases and applications, using specialized software. In terms of partnership between organizations, grid technology may include the same enterprise organizations as well as external organizations. Therefore, grids may involve both internal and external partners, as well as only internal ones. The complexity of the environment in which the grid will be developed and the requirements that have to be applied depend on the environmental impact of trade, defining the relationship of trust, security considerations, globalization and integration period, of the company involved, in the market [20].

Grid computing has recently enjoyed an increase in popularity as a distributed computing architecture. Grid computing got its name because it strives for an ideal scenario in which the CPU cycles and storage of millions of systems across a worldwide network function as a flexible, readily accessible pool that could be harnessed by anyone who needs it, similar to the way power companies and their users share the electricity grid. They cover multiple administrative domains and enable virtual organizations. Such organizations can share their

resources collectively to create an even larger grid. Grid Computing has proved beneficial in many ways, some of these benefits are [21]:

1) *Exploitation of Under-Utilized Resources:* Exploitation of under-utilized resources by running an existing application on different machines, exploiting idle times on other machines and aggregating unused disk drive capacity.

2) *Reduces Computational Time:* Computational time is reduced for complex numerical and data analysis problems.

3) *Provide Information Access:* Information accessibility to maximize the exploitation of existing data assets by providing unified data access during the querying process of non-standard data formats

4) *Reduces cost by optimizing existing IT infrastructure:* The grid facilitate reduction of costs by optimizing the use of existing IT infrastructure investments and by enabling data sharing and distributed workflow across partners, and therefore enabling faster design processes.
5) *Providing access to parallel CPU capacity:* Grid computing offers potential access for large-scale parallel computation to enhance performance in computationally intensive applications.

6) *Offers improved reliability:* Grid technology offers alternate approach to achieving improved reliability. Parallelization can boost reliability by having multiple copies of important jobs run concurrently on separate machines on the grid. Their results can be checked for any kind of inconsistency, such as failures, data corruption and tempering.

7) *Provision of resource balancing:* The grid offers good resource balancing measures that can handle occasional peak loads, job prioritization, and job scheduling.

8) *Effective management of resources:* With grid technology, management of organization can easily visualize resource capacity and utilization to effectively control expenditures for computing resources over a larger organization.

9) *Interoperability of virtual organizations:* The grid offers collaboration facilities and interoperability of different virtual organization by allowing the sharing and interoperation of the heterogeneous resources available.

10) *Access to additional resources:* The grid offers access to other specialized devices such as the cameras and embedded systems.

11) *Harnessing heterogeneous systems together:* Grid computing can be used to harness heterogeneous systems together into a mega computer by applying greater computational power to a task.

12) *Grid virtualization:* Grid computing offers grid virtualization, thereby making a single, local computer to undertake powerful applications.

Although grid computing has been developed in research laboratories, manufactures and companies have also started to adopt this technology on two of the most important benefits that grids bring: economy and performance. Grid computing aims to bring together operating systems and different hardware platforms into a single virtualized entity whose performances is higher on average then parts. Grid computing saves financial resources both in capital and operating costs. This positive aspect is achieved by using all the computing resources of all components of the grid. The second benefit of grid computing technology is performance in processing data integrity. By increasing the processing power, applications run faster the computing tasks and provide faster results. Even if the benefits of grids are real, this is still a specialized technology. Grid computing is suitable for organizations that already use in a way or another the high performance computing or are already oriented in some form towards distributed computing. Although there isn't a direct barrier to the passage of an organization to use a grid system, there are still issues to be taken into consideration before such a step. One of the reasons this happens is determined by the applications, which must provide the possibility of breakdown into smaller computing tasks in order to take advantage of parallel computing. If the operation currently executed depends on the previous tasks, then processing cannot be done in parallel on distributed resources so that the application cannot benefit from the grid advantages in terms of high performance computing [18]. Another issue that emerged over the time is the one that occurs in administration and accounting resources. This problem is more obvious as a solution involving a grid system involves multiple departments of the same organizations and each department require proportionality between the grids contribution, through its own resources, and the gain obtained from it. Besides resources accounting there's also a problem with software licensing. This is particularly important because licensing costs may eventually cancel the savings obtained from the use of grids. Securing the servers and grid administration resource is an issue that should not be neglected. In many cases servers are public addressable because of data delivering and receiving from agents that are anywhere, geographically speaking. So, these servers are exposed to attacks and unauthorized access, even DoS – Deny of Service. Experts recommends that, in this case, the companies must eliminate all unnecessary services and to carefully monitor these machines. However, this brings an extra complexity problem. One of the benefits of grid environments is that it allows the development of specific and on demand environments for various commercial environments. An important feature of on demand environments is the ability to respond to rapid changes in the market while reducing operational costs. Grid computing helps distributing and sharing data, which enables a collaborative improvement at

both inside and outside the enterprise. Thus, companies can reduce time spent placing products on the market, can quickly solve specific problems and can address immediately to the customers' requests. One of the troubles encountered when distributing the workload over several machines is the difficulty to trace the distribution process, which can lead to bandwidth problems. Is not only the bandwidth performance criterion but also the performances of the grid, of the way tasks are distributed and how they handle the distribution server. Bandwidth management is a serious problem that must be taken into account. A grid network bandwidth covers two aspects of any grid:

• *CPU utilization:* CPU bandwidth in a grid represents the maximum rate at which the grid can operate.
• *Using the network:* Available bandwidth of the grid and the one used internally by the grid.

These two elements affect a number of different systems and it's important to balance these elements in order to avoid the risk of having an inefficient grid, a grid that is unable to manage the computing tasks, or customers to start overburden the resources that they have at their disposal. Although there are several technologies that can be integrated into the grid or pattern which can form the basis of the mechanism of resource management and workload distribution and planning, however, they must be adapted to the needs of grids. We cannot avoid the artificial intelligence solutions to solve these types of problems, but must consider the complexity of the system and the additional resources that they are bringing.
Categories of grid computing fit into several areas of research communities. These may include areas oriented in intensive computational calculus, peer to peer, utility, data and applications and collaborations. Each of these communities can use a different model on the adoption of grid technology. Grid computing is a new concept for commercial industry and a large area of interest has developed around infrastructure virtualization through the manipulation of resources as utilities. Adopting a grid solution in a commercial activity depends on the ability of technology to meet the needs of improving turnover. This involves the adoption of the grid models key factors, such as adaptation of existing resources, reducing operational costs, creating a flexible and scalable infrastructure, while accelerating development, reducing the period of development and marketing of products and increase customer satisfaction and business productivity. An important issue in grid adoption models is the complexity of IT infrastructure needed to implement a grid system. The integration complexity of heterogeneous environments is a challenge and it must be taken into account factors such as activation of grid resources in homogeneous and heterogeneous environments, enabling resources in the form of services to external participants and porting applications to grid applications. These features allow a classification of grids into the following categories, with varying degrees of complexity at the level of integration[19]:

• *Grids developed to optimize computing infrastructure.*

• *Computing grids, with the virtualization of processing resources.*
• *Data grids with virtualization and data storage resources.*
• *Service grids with virtualized services for easy integration.*
• *Virtualized applications through the composition of resources, through service interfaces, applications from various partners.*

Architecture and technology standards developed for grid computing presents a crucial role in adopting grids commercially. Because these standards are constantly evolving and are not mature enough to support subsequent stages, which can evolve into after the adoption of grids, the speed at which these stages will be achieved is reduced. Grid computing solutions have been adopted in several key areas (finance, education, telecommunication, research) and answer the implementation and evolution of environmental performance requirements that are integrated. From this we can conclude that the success of grid computing depends on the integration, service orientation and ability to break down skills and applications and then expose them through the services. Although grid computing is an important step in information technology it does not involve the disappearance of existing resources and advanced equipment such as supercomputers, mainframes, clusters etc. The grid has no purpose in replacing the current technology resources, performance management, but the efficiency in extracting optimum benefits from investments made in such resources. There are still many areas (particle physics, simulations, computational intensity processing) in which execution of tasks involves only supercomputer (or dedicated resources in general performance), only areas where the grid cannot cope, or rather does not provide the necessary desired performance in such cases. Growth of technology and applications on this area, in the last period, may lead to a future in areas indispensability of enterprise grid computing. Such a critique must always be approached in the case of adoption of this technology to enable a more coherent adaptation to the needs for which a solution is desired grid.

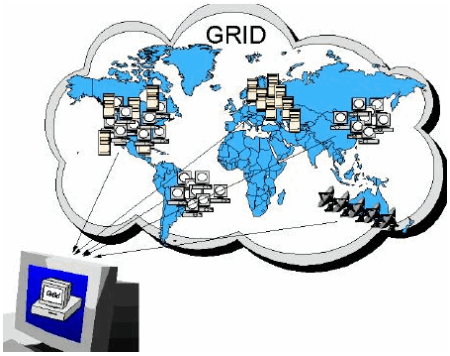

Fig 1. Grid computing is about exploiting distributed resources.

3. Cache memory

3.1 Memory Basics

Processors are generally able to perform operations on operands faster than the access time of large capacity main memory. Though semiconductor memory which can operate at speeds comparable with the operation of the processor exists, it is not economical to provide all the main memory with very high speed semiconductor memory. The problem can be alleviated by introducing a small block of high speed memory called a cache between the main memory and the processor [25].

The idea of cache memories is similar to virtual memory in that some active portion of a low-speed memory is stored in duplicate in a higher-speed cache memory. When a memory request is generated, the request is first presented to the cache memory, and if the cache cannot respond, the request is then presented to main memory.

The difference between cache and virtual memory is a matter of implementation; the two notions are conceptually the same because they both rely on the correlation properties observed in sequences of address references. Cache implementations are totally different from virtual memory implementation because of the speed requirements of cache.

We define a cache miss to be a reference to an item that is not resident in cache, but is resident in main memory. The corresponding concept for cache memories is page fault, which is defined to be a reference to a page in virtual memory that is not resident in main memory. For cache misses, the fast memory is cache and the slow memory is main memory. For page faults the fast memory is main memory, and the slow memory is auxiliary memory.

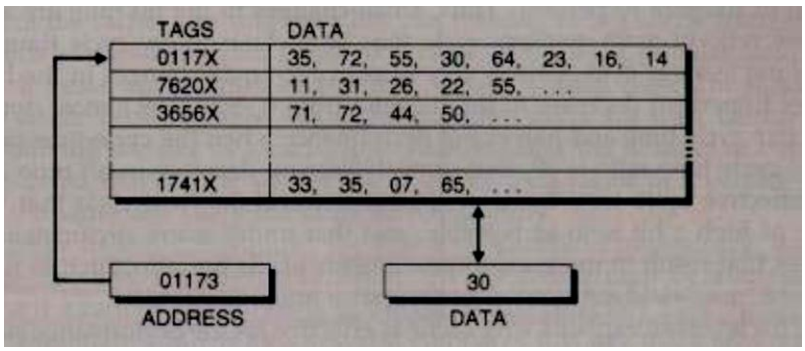

Fig 2. A cache-memory reference. The tag 0117X matches address 01173, so the cache returns the item in the position X=3 of the matched block.

Figure 2 shows the structure of a typical cache memory. Each reference to a cell in memory is presented to the cache. The cache searches its directory of address tags shown in the figure to see if the item is in the cache. If the item is not in the cache, a miss occurs.

For READ operations that cause a cache miss, the item is retrieved from main memory and copied into the cache. During the short period available before the main-memory operation is complete, some other item in cache is removed from the cache to make room for the new item.
The cache-replacement decision is critical; a good replacement algorithm can yield somewhat higher performance than can a bad replacement algorithm. The effective cycle-time of a cache memory (t_{eff}) is the average of cache-memory cycle time (t_{cache}) and main-memory cycle time (t_{main}), where the probabilities in the averaging process are the probabilities of hits and misses.
If we consider only READ operations, then a formula for the average cycle-time is:

$$t_{eff} = t_{cache} + (1 - h) t_{main}$$

Where h is the probability of a cache hit (sometimes called the hit rate), the quantity (1 - h), which is the probability of a miss, is known as the miss rate.

In Fig.2 we show an item in the cache surrounded by nearby items, all of which are moved into and out of the cache together. We call such a group of data a block of the cache.

3.2 Memory organization and performance

Although cache memories present a variety of architectural designs, there are a few common constrains and design patterns which are common in all cache memory types. Firstly, each modern cache is consisted of three levels (L1, L2 and L3) which represent the hierarchy in which data is being searched and stored. The first level is smaller, faster while the third is relatively bigger and slower than the first. This type of memory is consisted of transistor elements which lose their current which the cache is shut down, therefore preventing permanent storage of data. What is more, the performance gain of a cache is not only depended on low latency but also comes from data locality. Data locality is a phenomenon describing the same value, or related storage locations, being frequently accessed. There are two basic types of locality, temporal and spatial locality. Temporal locality refers to the reuse of specific data, within relatively small time duration. Spatial locality refers to the use of data elements within relatively close storage locations. In general purpose processors, both spatial and temporal locality is being exploited by cache memories as it is a common phenomenon which frequently appears in code execution [26].

Except for their hierarchical design, processor caches are also distinguished by the appliance of different implementation techniques, some of which are provided below [28].

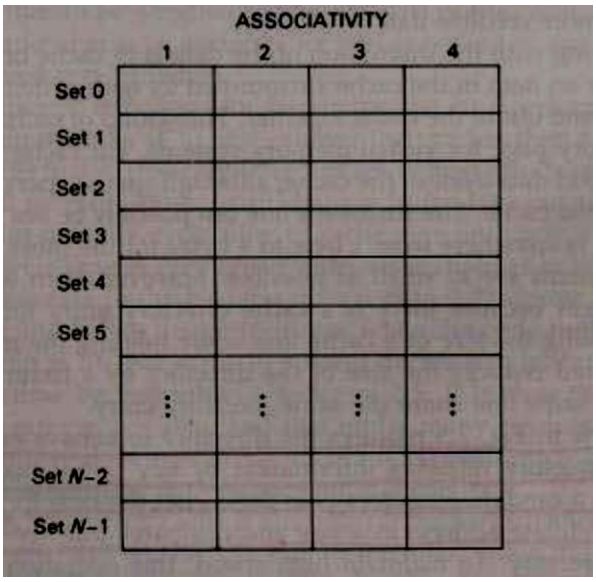

Fig 3. The logical organization of a four-way set-associate cache.

Fig.3 shows a conceptual implementation of a cache memory. This system is called set associative because the cache is partitioned into distinct sets of blocks, and each set contains a small fixed number of blocks. The sets are represented by the rows in the figure. In this case, the cache has N sets, and each set contains four blocks. When an access occurs to this cache, the cache controller does not search the entire cache looking for a match. Instead, the controller maps the address to a particular set of the cache and searches only the set for a match.

If the block is in the cache, it is guaranteed to be in the set that is searched. Hence, if the block is not in that set, the block is not present in the cache, and the cache controller searches no further. Because the search is conducted over four blocks, the cache is said to be four-way set associative or, equivalently, to have an associativity of four.

Fig.3 is only one example, there are various ways that a cache can be arranged internally to store the cached data. In all cases, the processor references the cache with the main memory address of the data it wants. Hence each cache organization must use this address to find the data in the cache if it is stored there, or to indicate to the processor when a miss has occurred. The problem of mapping the information held in the main memory into the cache must be totally implemented in hardware to achieve improvements in the system operation. Various strategies are possible[26].

1. **Fully associative mapping.**

Perhaps the most obvious way of relating cached data to the main memory address is to store both memory address and data together in the cache. This the fully associative mapping

approach. A fully associative cache requires the cache to be composed of associative memory holding both the memory address and the data for each cached line. The incoming memory address is simultaneously compared with all stored addresses using the internal logic of the associative memory, as shown in Fig.4. If a match is fund, the corresponding data is read out. Single words form anywhere within the main memory could be held in the cache, if the associative part of the cache is capable of holding a full address

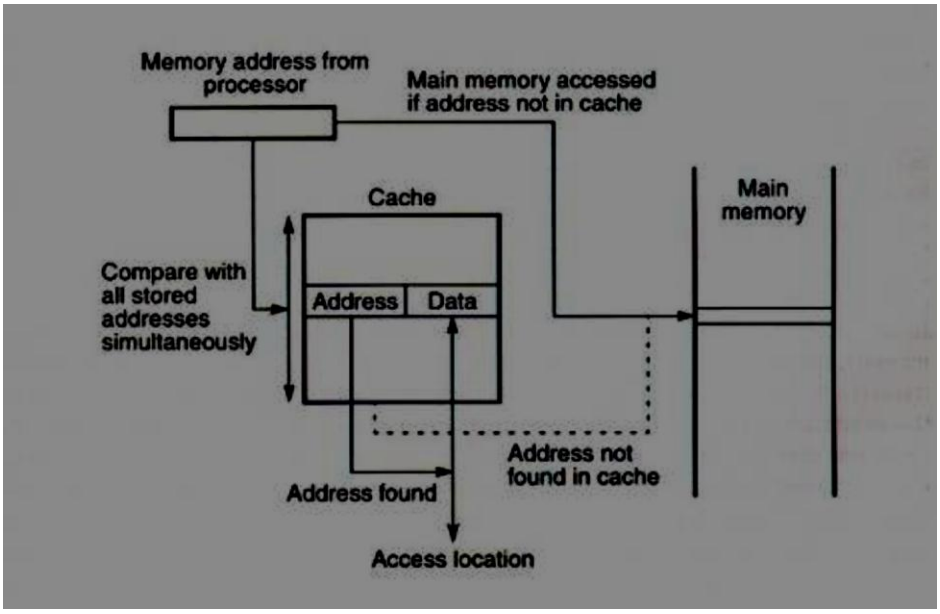

Fig 4.Cache with fully associative mapping.

In all organizations, the data can be more than one word, i.e., a block of consecutive locations to take advantage of spatial locality. In Fig.5 a line constitutes four words, each word being 4 bytes. The least significant part of the address selects the particular byte, the next part selects the word, and the remaining bits form the address compared to the address in the cache. The whole line can be transferred to and from the cache in one transaction if there are sufficient data paths between the main memory and the cache. With only one data word path, the words of the line have to be transferred in separate transactions.

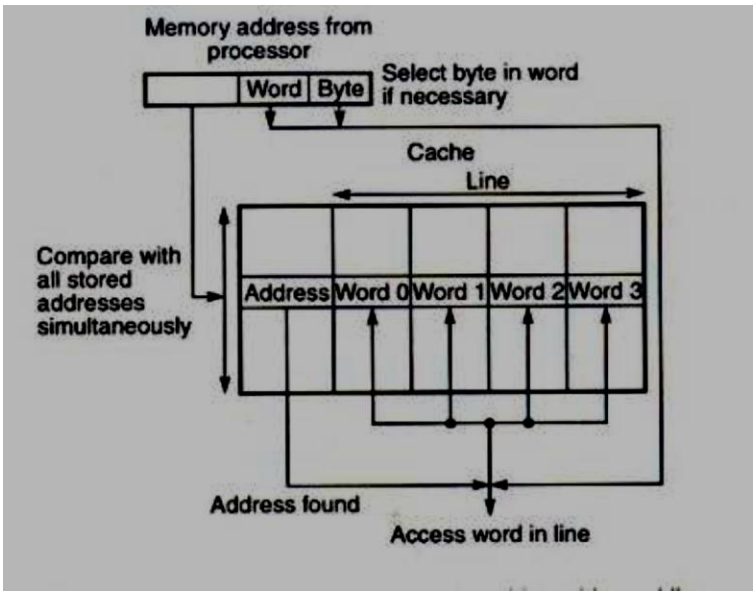

Fig 5.Fully associative mapped cache with multi-word lines.

The fully associate mapping cache gives the greatest flexibility of holding combinations of blocks in the cache and minimum conflict for a given sized cache, but is also the most expensive, due to the cost of the associative memory. It requires a replacement algorithm to select a block to remove upon a miss and the algorithm must be implemented in hardware to maintain a high speed of operation. The fully associative cache can only be formed economically with a moderate size capacity. Microprocessors with small internal caches often employ the fully associative mechanism.

2. Direct mapping

The fully associative cache is expensive to implement because of requiring a comparator with each cache location, effectively a special type of memory. In direct mapping, the cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address. This enables the block to be selected directly from the lower significant bits of the memory address. The remaining higher significant bits of the address are stored in the cache with the data to complete the identification of the cached data.

In Fig.6 ,the address from the processor is divided into two fields, a tag and an index. The tag consists of the higher significant bits of the address, which are stored with the data. The index is the lower significant bits of the address used to address the cache.

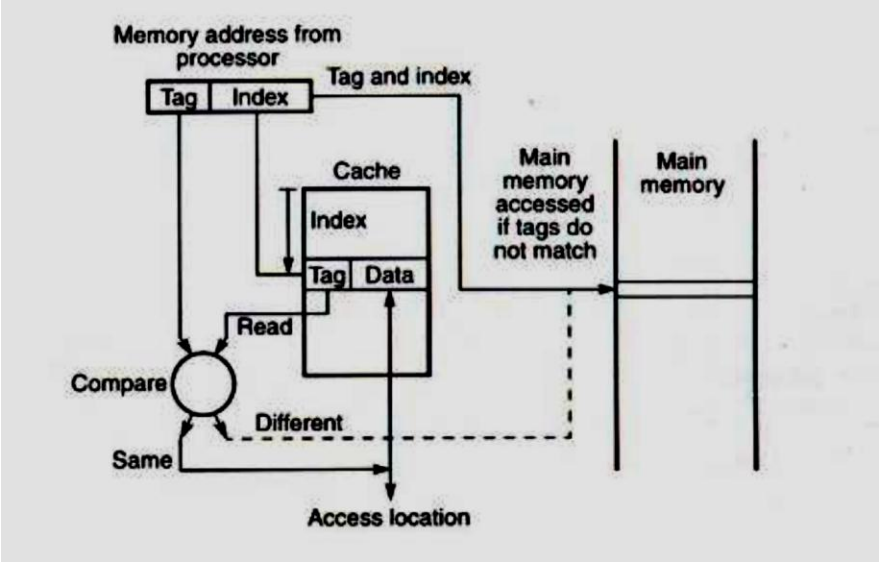

Fig 6.Cache with direct mapping.

When the memory is referenced, the index is first used to access a word in the cache. Then the tag stored in the accessed word is read and compared with the tag in the address. If the two tags are the same, indicating that the word is the one required, access is made to the addressed cache word. However, if the tags are not the same, indicating that the required word is not in the cache, reference is made to the main memory to find it. For a memory read operation, the word is then transferred into the cache where it is accessed. It is possible to pass the information to the cache and the processor simultaneously, i.e., to read-through the cache, on a miss. The cache location is altered for a write operation. The main memory may be altered at the same time (write-through) or later.

Fig.7. shows the direct mapped cache with a line consisting of more than one word. The main memory address is composed of a tag, an index, and a word within a line. All the words within a line in the cache have the same stored tag. The index part to the address is used to access the cache and the stored tag is compared with required tag address. For a read operation, if the tags are the same the word within the block is selected for transfer to the processor. If the tags are not the same, the block containing the required word is first transferred to the cache.

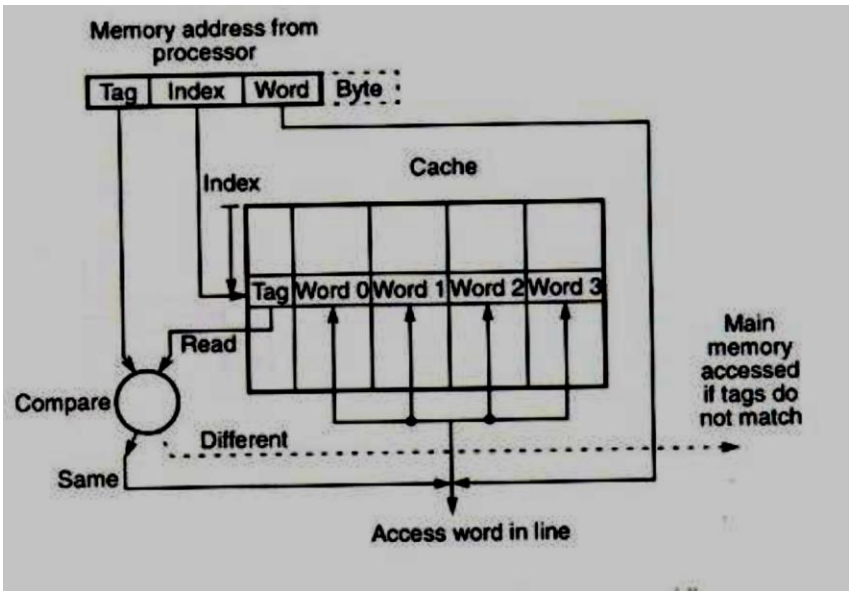

Fig 7.Direct mapped cache with a multi-word block.

In direct mapping, the corresponding blocks with the same index in the main memory will map into the same block in the cache, and hence only blocks with different indices can be in the cache at the same time. A replacement algorithm is unnecessary, since there is only one allowable location for each incoming block. Efficient replacement relies on the low probability of lines with the same index being required. However there are such occurrences, for example, when two data vectors are stored starting at the same index and pairs of elements need to processed together. To gain the greatest performance, data arrays and vectors need to be stored in a manner which minimizes the conflicts in processing pairs of elements. Fig.7 shows the lower bits of the processor address used to address the cache location directly. It is possible to introduce a mapping function between the address index and the cache index so that they are not the same.

3. Set-associative mapping

In the direct scheme, all words stored in the cache must have different indices. The tags may be the same or different. In the fully associative scheme, blocks can displace any other block and can be placed anywhere, but the cost of the fully associative memories operate relatively slowly.
Set-associative mapping allows a limited number of blocks, with the same index and different tags, in the cache and can therefore be considered as a compromise between a fully associative cache and a direct mapped cache. The organization is shown in Fig.8. The cache is divided into "sets" of blocks. A four-way set associative cache would have four blocks in each set. The number of blocks in a set is known as the associativity or set size. Each block in each set has a stored tag which, together with the index, completes the identification of the block. First, the index of the address from the processor is used to access the set. Then, comparators are used to

compare all tags of the selected set with the incoming tag. If a match is found, the corresponding location is accessed, otherwise, as before; an access to the main memory is made.

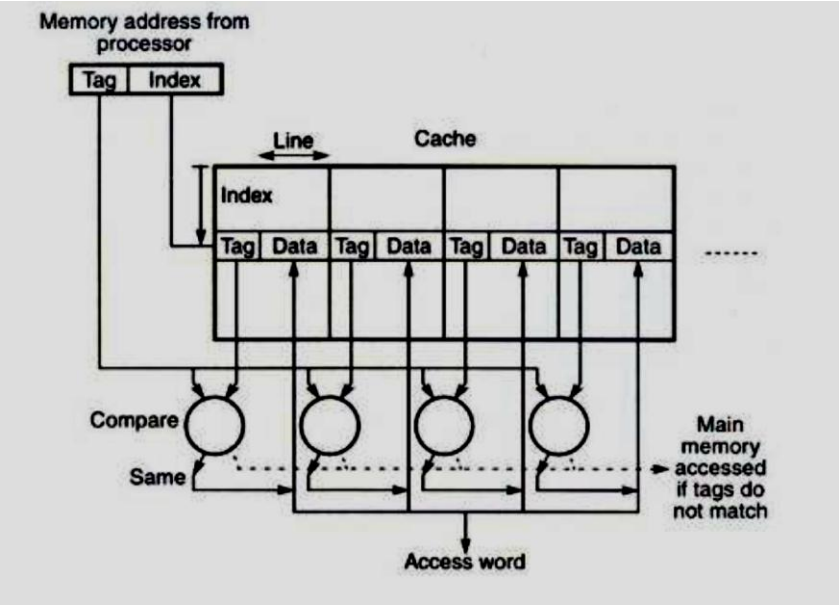

Fig 8.Cache with set-associative mapping.

The tag address bits are always chosen to be the most significant bits of the full address, the block address bits are the next significant bits and the word/byte address bits form the least significant bits as this spreads out consecutive man memory blocks throughout consecutive sets in the cache. This addressing format is known as bit selection and is used by all known systems. In a set-associative cache it would be possible to have the set address bits as the most significant bits of the address and the block address bits as the next significant, with the word within the block as the least significant bits, or with the block address bits as the least significant bits and the word within the block as the middle bits.

Notice that the association between the stored tags and the incoming tag is done using comparators and can be shared for each associative search, and all the information, tags and data, can be stored in ordinary random access memory. The number of comparators required in the set-associative cache is given by the number of blocks in a set, not the number of blocks in all, as in a fully associative memory. The set can be selected quickly and all the blocks of the set can be read out simultaneously with the tags before waiting for the tag comparisons to be made. After a tag has been identified, the corresponding block can be selected.

The replacement algorithm for set-associative mapping need only consider the lines in one set, as the choice of set is predetermined by the index in the address. Hence, with two blocks in each set, for example, only one additional bit is necessary in each set to identify the block to replace.

4. **Sector mapping**

In sector mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks. Any sector in the main memory can map into any sector in the cache and a tag is stored with each sector in the cache to identify the main memory sector address. However, a complete sector is not transferred to the cache or back to the main memory as one unit. Instead, individual blocks are transferred as required. On cache sector miss, the required block of the sector is transferred into a specific location within one sector. The sector location in the cache is selected and all the other existing blocks in the sector in the cache are from a previous sector.

Sector mapping might be regarded as a fully associative mapping scheme with valid bits, as in some microprocessor caches. Each block in the fully associative mapped cache corresponds to a sector, and each byte corresponds to a "sector block".

The performance of a cache can be quantified in terms of the hit and miss rates, the cost of a hit, and the miss penalty, where a cache hit is a memory access that finds data in the cache and a cache miss is one that does not. When reading, the cost of a cache hit is roughly the time to access an entry in the cache. The miss penalty is the additional cost of replacing a cache line with one containing the desired data.

(Access time) = (hit cost) + (miss rate)*(miss penalty) =>
(Fast memory access time) + (miss rate)*(slow memory access time)

Note that the approximation is an underestimate - control costs have been left out. Also note that only one word is being loaded from the faster memory while a whole cache block's worth of data is being loaded from the slower memory.

Since the speeds of the actual memory used will be improving ``independently", most effort in cache design is spent on fast control and decreasing the miss rates. We can classify misses into three categories, compulsory misses, capacity misses and conflict misses. Compulsory misses are when data is loaded into the cache for the first time (e.g. program startup) and are unavoidable. Capacity misses are when data is reloaded because the cache is not large enough to hold all the data no matter how we organize the data (i.e. even if we changed the hash function and made it omniscient). All other misses are conflict misses - there is theoretically enough space in the cache to avoid the miss but our fast hash function caused a miss anyway.

3.3 Memory access patterns

A memory access is triggered by specific instructions issued by the processor. These instructions usually contain actions which require memory store or load operations in order to be accomplished. In such cases a piece of data is either required to be fetched from the memory or stored in it [28,29].

When a memory operation is initiated, the processor needs the location of a specific memory address for loading or storing data. In order to obtain it, an address retrieve protocol is put into action which utilizes the current memory hierarchy for the search to be more efficient. At the beginning the requested address is searched in lower cache level which is closer to the processor. That cache is called level one (L1) cache. If the information is found, it is sent back to the processor so as to continue its function. If not, the search goes deeper in cache levels, specifically in level 2 (L2) and level 3 (L3) cache correspondingly. At any time during this search the information is retrieved, further search is prohibited and the processor proceeds in using the data fetch. If the cache memory fails to deliver the requested address then the search takes place in the main memory which is located further away from the processor. The main memory, also known as random access memory (RAM), is significantly slower than cache memory. If the search also fails and the requested address is not retrieved in RAM, then it has to be search on the hard disc or solid state drive of the system. This comes with performance heavy cost as the above devices have vast latency timings compared to a cache memory.

The smallest cell in a cache is known as block. Blocks can contain from 1 to even 256 words. Each piece of data is stored in a specific way according to current compiler optimization options. Although each compiler tends to introduce or change certain data storage options, there are global storage patterns which remain unchanged due to their effectiveness and ease of usage. The most common of such patterns is the way arrays and matrixes. Each matrix or vector element is always stored sequentially meaning that each element is stored next to its previous one. This storage pattern is widely exploited by cache memories as it is the main reason that data locality is a widely known phenomenon.

3.4 Adaptive caches

An adaptive cache is a cache which is able to resize itself on the fly in order to increase hit rate for the current workload [23]. Resizable caches can trade-off capacity for access speed to dynamically match the needs of the currently executed code. To do this, the adaptive cache needs to maintain some more information than a traditional cache. This information is translated into more complex hardware design and comes with a considerable area cost.

In general there are two types of adaptive cache memories, one which is resizable and other which can change its replacement policies. In this study we focus on a cache which can change it size rather than one changing its replacement policy. This cache can be configured so that either its block size or associativity changes but not it total size, as it is restricted by physical characteristics. Although relatively new to the industry, the concept of adaptive cache is still under research as experimental prototypes are underway [23,24].

4. New cache design organization

In today's microprocessors, the cache architecture is highly optimized for one particular design and cannot be changed after fabrication. While allowing efficient implementations in dedicated logic, this inflexibility also implies that new techniques cannot be deployed in the field. The best cache configuration depends on the application characteristics and design constraints, like performance, power consumption or area, which has promoted the diversity of cache architectures found in different processors. Since no cache organization satisfies the requirements of all applications, a promising approach to this problem is to add reconfiguration capabilities to the cache memory.

In this thesis, we propose one method for dynamic reconfiguration of a properly designed cache memory. The reconfigurable cache memory, designed and simulated in C code, bases its functionality on our existing cache designs but the new design has 3 different cache modes enhancing the previous designs for obtaining resource usage and speed improvement.

4.1 Cache operation

The cache organization resembles that of standard cache memory as it contains sets, blocks and word of data. The very function which diversifies our design from a standard industrial cache is the reconfiguration capabilities it presents. Below the data flow of the new design is described during a series of cache memory requests from the processing unit.

In this scenario let us assume that the processor is to execute a load/store instruction which has to send or retrieve data from the cache memory. The transaction is fulfilled as normal except for a specifically design circuit for snooping the cache instruction and data flow. That circuit collects data correlated with the number of hits or misses and the number of blocks evicted from a set due to block overcrowding and cache replacing policy. When the number of blocks evicted or the number of misses overcome a predefined threshold, the control circuit sends a reconfiguration signal to the cache memory.

Reconfiguration may occur infrequently (e.g., just once at the start of the application) or frequently (e.g., at the beginning of certain loops), depending on the characteristics of the application and the processor activities for which the adaptive is used. The mechanism to detect when to reconfigure can be software or hardware controlled. A software-controlled approach can expose the cache status register to the code-generator (user or compiler) which can use information about the program behavior to invoke appropriate reconfiguration at the appropriate points in the program. Alternatively, a hardware approach could use hardware performance monitoring support to automatically decide when and how to change the partitions.

The event monitor would be used to send a message to the control circuit fabric and the control circuit controller on cache events that are configured to be monitored. These messages contain the event type such as read hit, read miss, write hit and write miss, along with additional information on the details of the event. In our current design, the detailed information includes the instruction address (PC), the memory address of the access, the data value (only for a write

operation) and the cache set number in a set-associative cache. In most cases, the monitoring messages do not affect the main cache controller operation.

The final issue with the reconfigurable cache organization is the level that is reconfigurable in a multi-level cache hierarchy. The cache organization described above does not preclude its application to any level. Tradeoffs in terms of the size, granularity, access time, and usage of the partitions will determine the level to partition. In figure 9 a schematic shows the functionality of the new cache design pointing its core modules and the communication occurring between them.

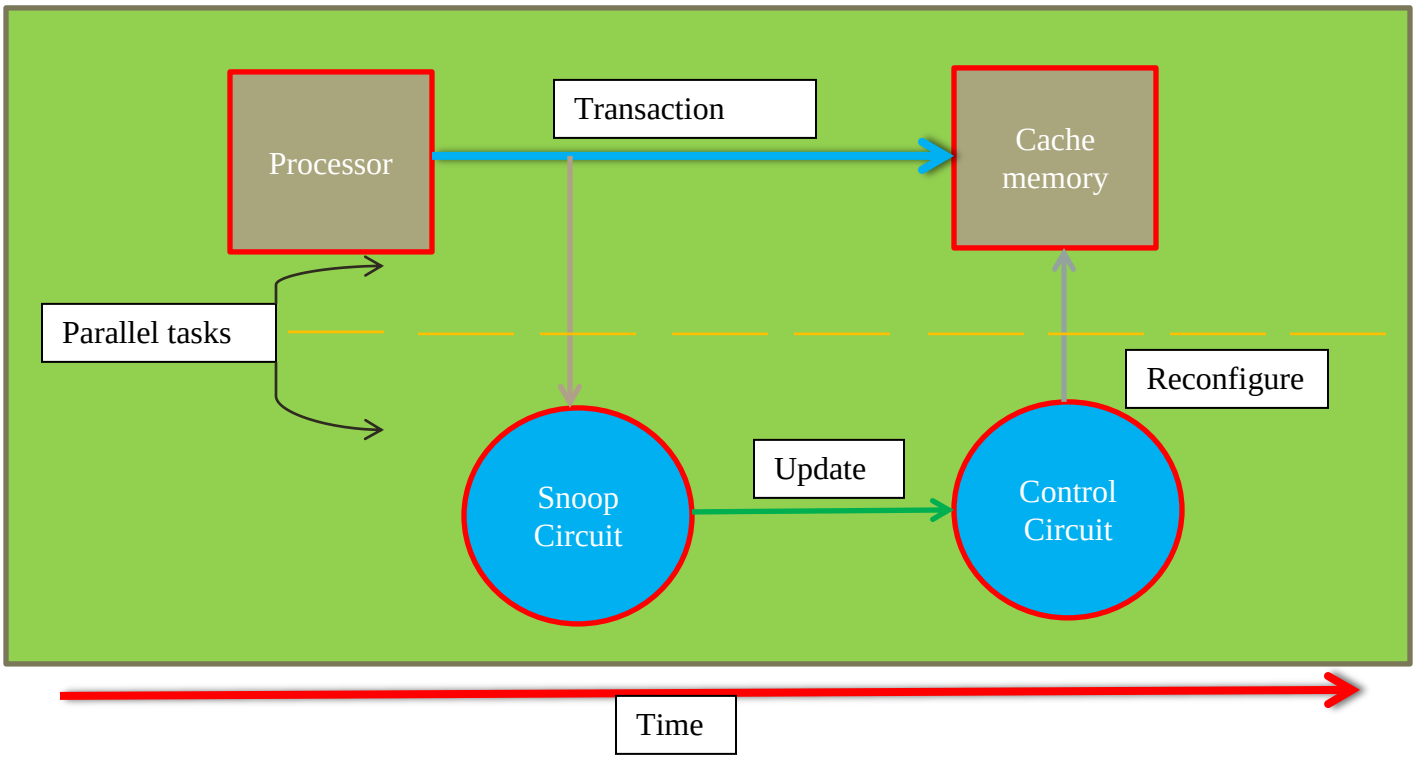

Fig 9. Cache organization

4.2 Reconfiguration options

The simulator developed supports a wide range of reconfigurable ache memories sorted by their size and associativity. As the user explicitly configures the cache's initial size, associativity and block size, the system saves the data collected and uses them in order to complete the simulation. During the runtime the cache is reconfigured any number of times depending on the application requirements. Specifically, the cache can be reconfigured as far as associativity and block size is concerned. The more cache misses occur the more the simulator tends to reconfigure the cache's associativity, raising the number of blocks contained in a single set, thus lowering the total number of sets. On the other hand the more block evictions (due to cache replacement policy and block overcrowding) occur, the more the likelihood the simulator will increase the number of existing blocks in a single set. In that way, the cache is reconfigurable in two ways: In the number of sets and the number of blocks.

4.3 Distributed computing with adaptive cache

Although the new cache design we proposed works great in a single processor system whether this processor supports multithreading or not it is initially meant to exist in a cluster of processors called grid. In this cluster, each core of the participating processor utilizes a level one cache similar to our design. This would greatly improve the hit rate of the corresponding processor resulting in greater system performance as the whole cluster would suffer a cache miss less often. As we mentioned in the chapter 1, the most widely used algorithm for task compartmentalization in such systems is called MapReduce. We noticed that in the first stages during the execution of Mapreduce, when the major tasks are broken into smaller and simpler ones, the compiler has complete awareness of the code to be executed in different terminals. Exploiting its awareness, the compiler can predict the memory access patterns of the application, but not the exact memory addresses that will be used. In this case the compiler can send a signal to the cache, informing it whether more sets or more blocks per set are needed. For example, if the compiler identifies a matrix or vector multiplication, more sets will be needed soon. On the other hand, if a tridiagonal matrix access is identified, then the hardware will be in need of more blocks per set. This type of information is easily extracted by the compiled code and can be used to further increase the efficiency of our cache.

We use a signal called "spatial locality" to depict such information flown by the compiler. While this signal has zero value, we assume that no predictions are made and the cache works as intended. If "spatial locality" has the value of "1" then we assume that the compiler has predicted some memory access patterns and we use them to our advantage.

Depending on the compiler prediction, we initially configure the cache (before the code execution begins) either with high associativity or with a relative high number of blocks in each set. Configuring the cache with initial high blocks size, enables it to achieve high hit rates when data structures with high locality are used (such are arrays and matrices). On the other hand, initially configuring the cache with high associativity we increase the hit rate of cache when data

structures with more relaxed access patterns (such as lists) are involved. Either way this is an initial configuration only as the cache organization can (and probably will) change during the code execution. This initial configuration is a decision based on the compiler's prediction. What is more we can also reconfigure each cluster node's cache exclusively, according to the task that this node is about to execute, further improving the whole system's performance. Above we present such an example of Mapreduce and how each node's cache is configured. In this scenario we assume a cluster of 4 nodes, one node responsible for mapping and three nodes responsible for the reduce operation.

Time unit	Node 0	Node 1	Node 1 cache	Node 2	Node 2 cache	Node 3	Node 3 cache
0	Mapping	Idle	Idle	Idle	Idle	Idle	Idle
1	Predicting	Idle	Default init.	Idle	Assoc. init.	Idle	Block init.
2	Idle	Executing	Working	Executing	Working	Executing	Working
3	Idle	Executing	Reconfiguring assoc.	Executing	Working	Executing	Reconfiguring block
4	Idle	Executing	Reconfiguring block	Executing	Reconfiguring assoc.	Executing	working
5	Executing	Reducing	Working	Reducing	Working	Reducing	working.

In the table above we assume that we have a system of 4 nodes. The node 0 is responsible for the map of resources while the nodes 1-3 are responsible for code execution and reduction. Moreover we are monitoring the cache behavior of each node except for the node 0.

Time unit 0: During this period of time the central node (node 0) is compiling and compartmentalizing the code of the application, distributing it to the other 3 nodes for execution. So, during this period the nodes 1-3 are idle waiting for the node 0 to finish its task.

Time unit 1: During this time period the central node is making predictions for the memory access patterns that will be encountered. At the end of this period each node is informed through the "spatial locality" signal and starts configuring its level one cache according to the predictions made by the compiler. In this particular case node 1 does not make any initial configurations while node 2 increases the associativity of its cache and node 3 increases the number of blocks in each set.

Time unit 2: During this time window the central node is inactive waiting for other nodes to execute the distributed code. The other nodes are all in code execution phase while their caches work separately from each other, each one gathering data that will enable it reconfigure itself in the future.

Time unit 3: While node 0is still idle, the other nodes continue code execution. Here we can see the first cache reconfigurations take place depending on the needs of each code segment. Specifically, node one cache increases its associativity and node 3 cache further increases the number of block in each set while node 2 cache does not reconfigure itself as its miss threshold is not yet surpassed.

Time unit 4: During this period of time node 0 remains idle as the other nodes have not yet finished code execution. Moreover node one cache increases the number of blocks in each set while node 2 cache is increasing its associativity. Node 3 cache remains as it is as the total misses are still lower than the established threshold.

Time unit 5: In the last time window the nodes 1-3 have executed the distributed code are on the reduce phase where all the results are collected and sent back to the central node 0. From this point node 0 starts collecting all the results and combining them into one consistent format. After the completion of this task, the code is executed and all the results are available.

5. **Performance**

The designed cache memory was evaluated using different types of benchmarks. We developed 6 different types of benchmark programs which present similar memory access pattern with big data applications. Those patterns include vector multiplication, diagonal matrix access, list element access, column based matrix access, matrix multiplication and random access patterns. Each benchmark generates 10.000 memory addresses based on its specific pattern and uses them to gain access to the designed cache. The total misses and hits are counted in order to correctly calculate the delivered performance.

We provide analytical results based on the hit rates achieved by our new cache design compared to those of a same size cache which is not adaptive. Moreover, we also present the performance results while the flag "spatial locality" is active.

Vector multiplication.

In this benchmark we consider two vectors as inputs and one vector as a output. In that way 10.000 load and store instructions are generated in order to read the elements of the two first vectors and to store the result on a third vector which is created. In this scenario an common access pattern is implemented, that which each element of a specific vector is the word following the previous element, a common technique for storing vectors in the cache memory. Specifically the table below depicts the exact pattern by which the addresses are generated.

Vector A	Vector B	Vector C (Result)
Load 0xAF000000	Load 0xAFFF0000	Store 0xAFDD0000
Load 0xAF000004	Load 0xAFFF0004	Store 0xAFDD0004
Load 0xAF000008	Load 0xAFFF0008	Store 0xAFDD0008
Load 0xAF00000C	Load 0xAFFF000C	Store 0xAFDD000C
Load 0xAF000010	Load 0xAFFF0010	Store 0xAFDD0010

This is a very common case in any industrial program, as vector multiplication is an arithmetic operation which is commonly used not only in the majority of applications but also in big data industry. The results in hit rate per cent are shown in the figure below.

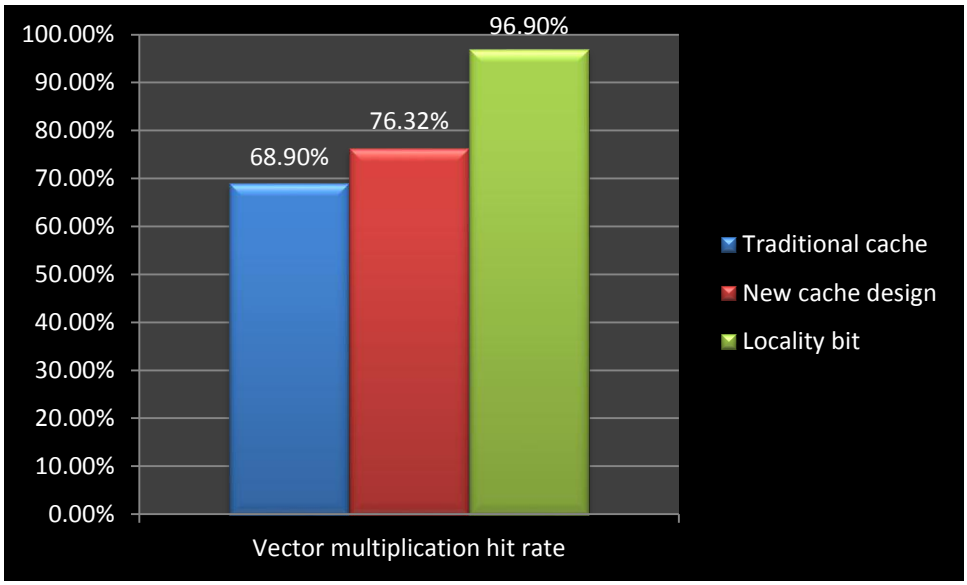

Fig 10.Vector multiplication hit rates

List access.

A list is a data structure widely used in any programming language. It is consisted of several nodes stored in different memory locations, but all stored in the data segment of the memory. Thus the memory addresses generated are random with an upper and lower threshold which cannot be surpassed. Finally we keep in mind that each struct's size is proportional to the amount of elements it contains, so as not to surpass the designated generated memory addresses. In the table below a typical address generation for a list is depicted. For this paradigm we presume that each struct has 64 bits of data. The results we obtained are shown in the figure 10.

Generated address
0xAFB00000
0xAFB00004
0xAFB00008
0xAFBAA000
0xAFBAA004
0xAFBAA008
0xAFBCB000
0xAFBDD000
0xAFBCB004
0xAFBDD008

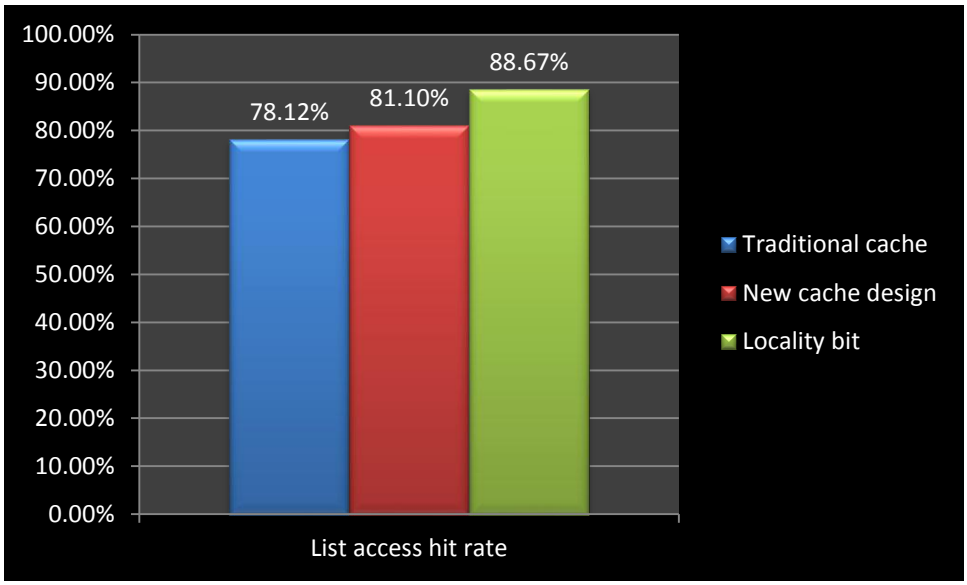

Fig 11.List access hit rate

Tridiagonal matrix access.

In linear algebra, a tridiagonal matrix is a matrix that has nonzero elements only on the main diagonal, the first diagonal below this, and the first diagonal above the main diagonal. Tridiagonal matrixes are widely used in memory intensive applications because sparse memory libraries (like cs sparse) make many uses of them. Figure 11 shows the hit rates we obtained by accessing such a matrix.

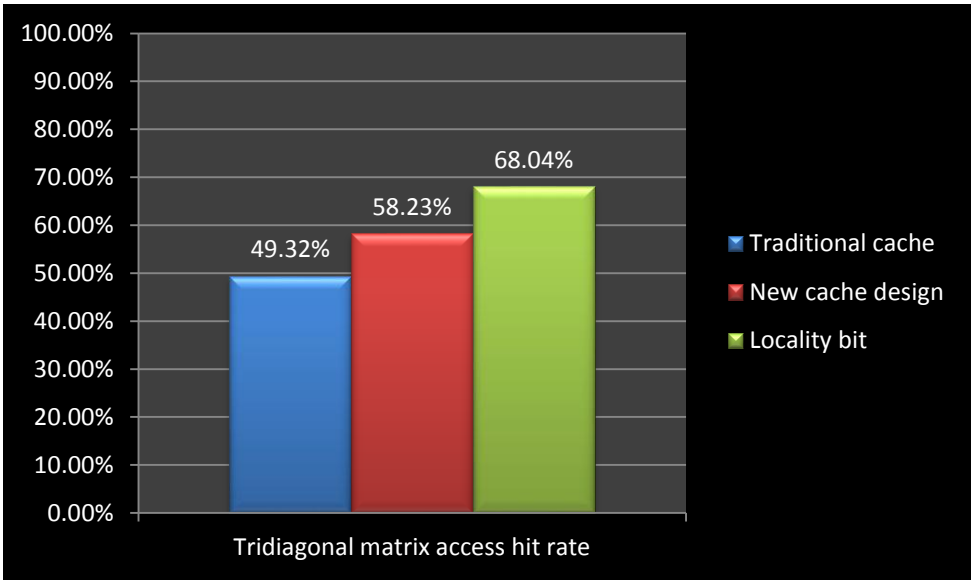

Fig 12. Tridiagonal matrix access hit rate.

Column based matrix access.

In this benchmark we study another, not so common access pattern for matrixes. Although consecutive elements are stored in sequential order in a matrix, in some cases developers choose to access such elements in a different order than the stored one. This access type is ineffective and expensive as far as performance is concerned but in some cases the only solution. Figure 12 shows the effective performance of such an access pattern.

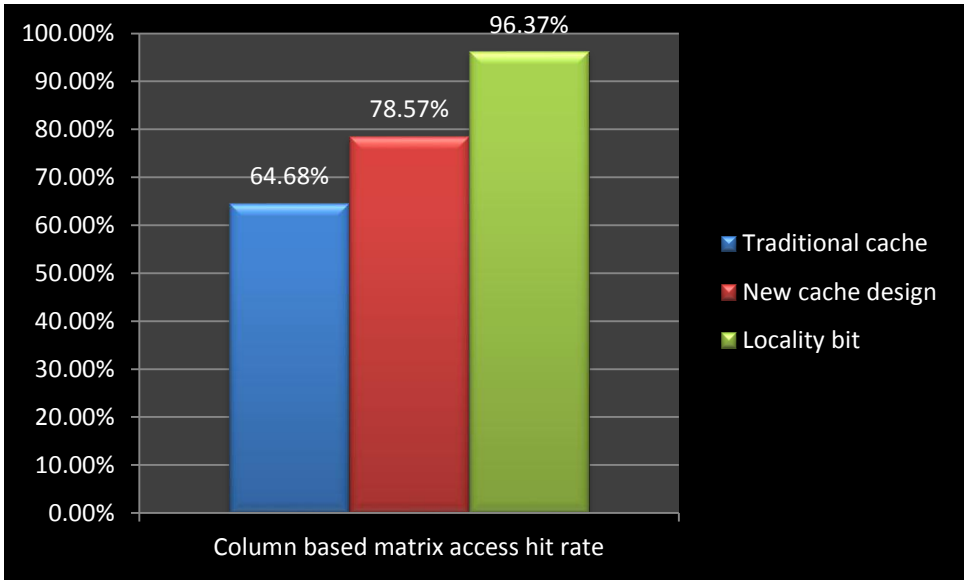

Fig 13.Column based matrix access hit rate.

Matrix multiplication.

When matrix multiplication code is executed, two (or more) matrices are accessed element by element and the product is store to a third matrix. The memory access pattern is very similar to the vector multiplication process. Specifically, in the table below we present a sample memory address generation during a multiplication of two 2x2 matrices.

Matrix A	Matrix B	Result (matrix C)
0xAFB00000	0xAFBAA000	0xAFBAAB00
0xAFB00004	0xAFBAA004	0xAFBAAB04
0xAFB00008	0xAFBAA008	0xAFBAAB08
0xAFB0000C	0xAFBAA00C	0xAFBAAB0C

In the figure 14 we present the performance results in such operation.

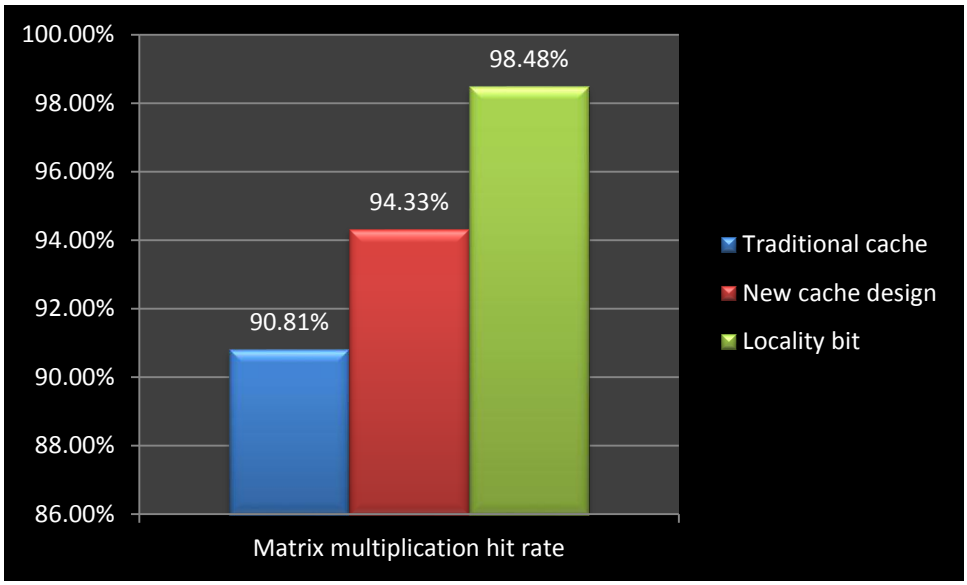

Fig 14.Matrix multiplication hit rate.

Random access.

We also created a random address generation in order to test real time memory accesses after a context switch. After a context switch, the new memory access addresses should not correlate with previous addresses generated by the previous application and should be nearly random. Thus a random generation could closely simulate such a state. The performance results we obtained are shown in the figure 15.

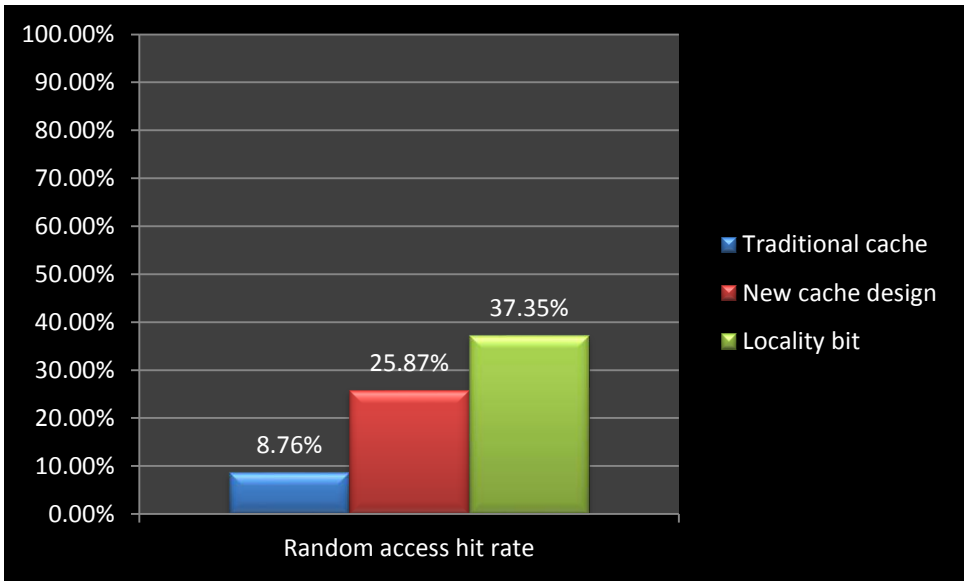

Fig 15.Random access hit rate

Overall hit rate

The figure 16 show the overall performance we obtained from the benchmarking process of our new cache design. The benchmarks were written in correspondence with the most common memory access patterns of big data applications, having in mind the vast volume of the data being processed. The figure 16 is the proof we need in order to assume that our cache delivered the required performance, while maintaining the functionality needed.

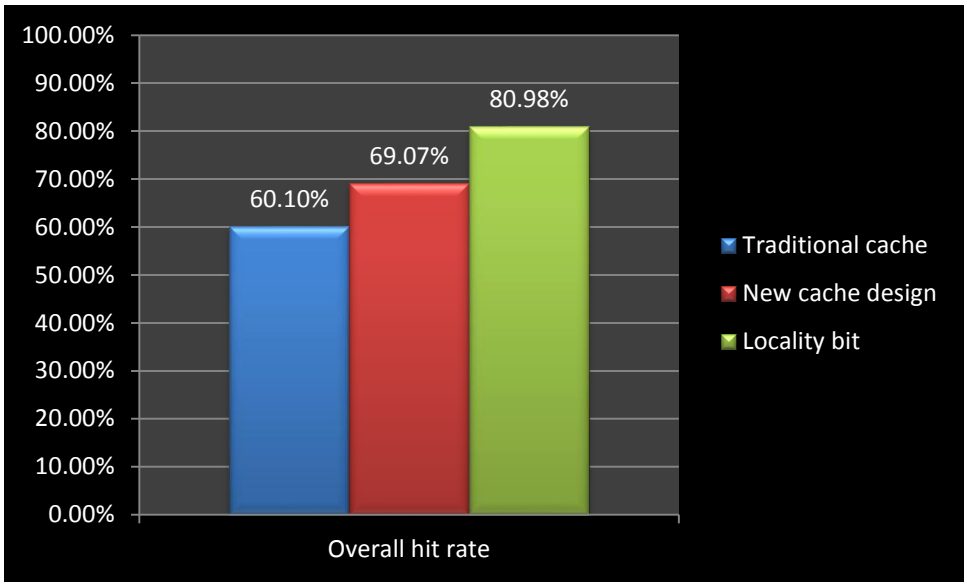

Fig 16.Overall hit rate.

6. Future work

Although this new scheme presented delivers noticeable better performance in big data applications, compared to standard static cache memories, there are a lot of research concerns and ideas which can be implemented in order to further improve its performance and lower its complexity. First of all, this configurable cache is restrained to level one only cache memories. This restrain, although tolerable in this project, is not a fully complete solution for a fully functional cache memory. Thus, an extension of this simulator can be programmed in order to include level 2 and 3 caches. Moreover there could be another extension concerning power dissipation. The number of blocks and sets is (according to this implementation) changing in conjunction with the performance needed. This can change to include power consumption criteria in order to reduce power consumption. What is more, reconfiguration could also be achieved in the cache block replace policy, enabling it to change it according to the requirements of the executed application. On the whole we can deduce that numerous changes can be done in order to improve the existing schema in the direction the researcher wishes.

References

[1] Jeffrey Dean and Sanjay Ghemawat. “Map Reduce: Simplified Data Processing on Large Clusters”. *In Proc. of the Sixth Symposium on Operating System Design and Implementation San Francisco, CA, December, 2004.*

[2] Community white paper. ”Challenges and Opportunities with Big Data”.2012

[3] NESSI white paper. ”Big Data A New World of Opportunities”.2013

[4] Oracle white paper. ”Oracle: Big data for the enterprise”.2013

[5] T.P. Shabeera, S.D. Madhu Kumar.”Optimizing virtual machine allocation in MapReduce cloud for improved data locality». In Int. J. of Big Data Intelligence, 2015 Vol.2, No.1, pp.2 – 8

[6] G.K. Karthikeyan, Prassanna Jayachandran et al.” Energy aware network scheduling for a data centre”. In Int. J. of Big Data Intelligence, 2015 Vol.2, No.1, pp.37 – 44.

[7] Jiadong Wu,Bo Hong.” Multi-source streaming-based data accesses for MapReduce systems”. In Int. J. of Big Data Intelligence, 2014 Vol.1, No.1/2, pp.36 – 49.

[8] Honjo, T. , Oikawa, K.” Hardware acceleration of Hadoop MapReduce”.In IEEE International Conference on Big Data, pages 118 - 124 , Oct. 2013.

[9] Mian Lu , Lei Zhang , Huynh Phung Huynh et al.” Optimizing the MapReduce framework on Intel Xeon Phi coprocessor”. In IEEE International Conference on Big Data, pages 125-130 , Oct. 2013.

[10] Nadungodage, C.H. ,PYuni Xia ,Lee, J.J.” GPU accelerated item-based collaborative filtering for big-data applications” .In IEEE International Conference on Big Data, pages 175-180 , Oct. 2013.

[11] Chardonnens, T. ,Cudre-Mauroux, P. ,Grund, M.” Big data analytics on high Velocity streams: A case study”. In IEEE International Conference on Big Data, pages 784-787 , Oct. 2013.

[12] Sang-Woo Jun , Ming Liu, Kermin Elliott Fleming et al.” Scalable multi-access flash store for big data analytics”. In Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays,Pages 55-64,2014

[13] Lisa Wu, Raymond J. Barker, Martha A. Kim, et al.” Hardware Partitioning for Big Data Analytics”Published in Micro, IEEE (Volume:34 , Issue: 3 , Pages 109-119).March 2014.

[14] Alok Choudhary , Ramanathan Narayanan , Berkin Özıs et.al. “Optimizing data mining workloads using hardware accelerators ”.In Proc. of the Workshop on Computer Architecture Evaluation using Commercial Workloads, 2007.

[15] Becerra, Y. ,Beltran, V. ; Carrera, D. et.al. “Speeding Up Distributed MapReduce Applications Using Hardware Accelerators”. In ICPP International Conference on Parallel Processing, Pages 42-49, Sept 2009.

[16] Christoforos Kachris ,Georgios Ch. Sirakoulis, Dimitrios Soudris.” A configurable mapreduce accelerator for multi-core FPGAs”. In proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays, Pages 241-241, 2014.

[17] Yi Shan, Bo Wang, Jing Yan.” FPMR: MapReduce framework on FPGA”. In Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays, Pages 93-102, 2010.

[18] Jack Dongarra , Alexey Lastovetsky.” An Overview of Heterogeneous High Performance and Grid Computing”. White paper 2006.

[19] Henri Casanova.” Distributed computing research issues in grid computing”. In ACM SIGACT News Volume 33 Issue 3, Pages 50-70, Sept 2002.

[20] Browne, J.C.” Grid computing as applied distributed computation: a graduate seminar on Internet and Grid computing”. In IEEE International Symposium on Cluster Computing and the Grid. Pages 239 - 245 ,April 2004.

[21] Shruti N. Pardeshi, Chitra Patil, Snehal Dhumale.” Grid Computing Architecture and Benefits”. In International Journal of Scientific and Research Publications, Volume 3, Issue 8, August 2013.

[22] Alaa R. Alameldeen, David A. Wood.” Adaptive Cache Compression for High-Performance Processors”. In Proceedings of the 31st annual international symposium on Computer architecture, Page 212, 2004.

[23] Subramanian, R., Smaragdakis, Y. , Loh, G.H. “Adaptive Caches: Effective Shaping of Cache Behavior to Workloads”. In 39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006. MICRO-39,Pages 385 – 396, Dec 2006.

[24] Nimrod Megiddo, Dharmendra S. Modha.” Outperforming LRU with an Adaptive Replacement Cache Algorithm”. In Journal Computer, Volume 37, Issue 4, Page 58-65, April 2004

[25] Alan Jay Smith.” Cache Memories”. In ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 14 Issue 3, Pages 473-530, Sept. 1982.

[26] Jih-Kwon Peir, Windsor W. Hsu and Alan Jay Smith.” Implementation Issues in Modern Cache Memory”. Technical Report, 1998.

[27] Intel white paper. ”Cache memory”, 2006.

[28] David A. Patterson , John L. Hennessy. “Computer Organization and Design, 5th Edition”, book published in Oct 2003.

[29] David A. Patterson and John L. Hennessy.” Computer Architecture, Fifth Edition: A Quantitative Approach”. Book published in 2011.