International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-4, April 2013

Energy Efficient Partitioning Of Last Level Cache Memory with Cooling Management for Memory and CPU Subsystems

Vikash Sharma, Jayant Kumar

Abstract*— This paper presents a technique to improve the over- all performance of the multiprocessor chip. Efficient partitioning of last- level cache memory in a multi-processor chip can increase the performance significantly. The concept is to first allocate the fixed number of ways for a core and then forced the cache data to be way aligned so that a particular way is owned by a core at a particular time. At the time of access, cores cooperate with each other to migrate the ways between them so that a core has to consult only those ways which it has owns to find its data from which dynamic energy can be saved and unused ways can be power-gated for saving the static energy. This paper also presents a cooling management strategy for memory and CPU subsystems. It manages the temperature of memory and CPU subsystems by activating the memory and CPU actuators which does the required action in the subsystems of memory and CPU. It considers the thermal and power states of CPU and memory, thermal coupling between them and fan speed to arrive at energy efficient decisions.

***Index Terms*— last level cache memory (LLC), CPU actuator, Memory actuator, RAP, WAP**

I. INTRODUCTION

For the improvement of performance of a processor, cache memory plays a significant role. Normally in the chip multiprocessor, multi-level cache memory is used in which the last level cache (LLC) is the most and frequently shared among all the cores on the chip. So, it is necessary to decrease the energy consumption of last level cache memory by partitioning it and our method is one of the efficient methods for increasing the performance of chip multiprocessors (CMPs). The concept of partitioning the cache memory is to give the resources to those applications that can benefit most from them .Several techniques [1],[2] have already been developed for partitioning the last level cache memory but most of them have focussed on single core designs. According to those methods, they turn off the particular ways of cache memory for reducing the static energy and predict the numbers of ways that will be accessed to reduce dynamic energy but these methods are not directly applicable to last level cache memory of CMPs due to filtering effects. The previous approaches [1], [2], [4] also did not consider energy savings. Our method not only reduces the dynamic energy but also static energy while maintaining high performance.

Manuscript published on 30 April 2013.
***** Correspondence Author (s)
Vikash Sharma*, M.Tech VLSI, ABV- IITM, Gwalior, India.
Jayant Kumar, M.Tech VLSI, ABV- IITM, Gwalior, India.

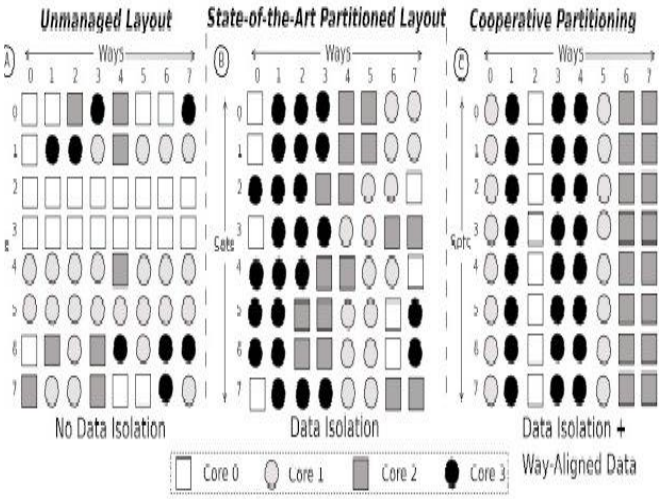

Figure.1 Example of Partitioning the Cache Memory

This approach forces the cached data to be way-aligned, so a particular way is owned by a single core at any time. Cores cooperate with each other to migrate ways between themselves after partitioning decisions have been made. Upon access to the cache, a core needs only to consider the ways that it owns to find its data which saves the ***dynamic energy***. Unused ways can be power gated for ***static energy*** savings. The operation of Cooperative partitioning can be easily understood with the help of example which is shown in figure.1

In this example, there are four cores considered whose data are distributed in different ways of the cache memory among all the sets. In figure (a), it is an example of unmanaged cache where data belongings to the cores are entirely mixed across all the sets and ways. So in this case power consumption is high because a core has to consult all the ways of the cache memory to access its data. Now the earlier techniques are used in figure (b) where a fixed number of ways is allocated to each core across all the sets. But within the sets, data from each core can reside in any way, so a core has to access all the ways to find its data. Again the problem of power dissipation is not yet resolved. Our approach is one step forward to the previous approaches is shown in figure (c). In this approach, the data is enforced to be way-aligned, so that a way is owned entirely by a single core at a time. So there is no need to access all the ways, as a result dynamic energy is reduced and the unused ways can be power gated which reduces the static energy. This scheme guarantees that its data will never be found anywhere else in the cache.

For example, core 3 has to access only way 1, way 2 and way 3 so that remaining ways can be shut down for energy savings. Due to cooperative takeover principle (which is discussed in next section), migration is five times faster on average than the existing approaches and returns less data back to memory.

Today, modern servers use a fan subsystem to reduce the temperature of servers. The power consumed by a fan is cubically related to the air flow rate which is also a serious concern for the power efficiency. The cooling subsystem also becomes inefficient when it operates far from the point due to power thermal distribution. There is a need to account workload allocation policies to minimize the cooling energy by creating a better thermal distribution. Normally due to cost and area constraints, a common set of fans is normally used to cool both the CPU and memory subsystems. For such scenarios, the inlet temperature of the downstream becomes a function of the temperature of the upstream in addition to the primary inlet of the temperature. This problem is solved by our proposed management technique which considers thermal and power states of a CPU and memory, thermal coupling between them and fan speed to derive at energy efficient decisions. It has *CPU and memory actuators* to implement its decisions. The memory actuator reduces the energy of memory by performing cooling aware clustering of memory pages to a subset of memory modules. The CPU actuator saves cooling energy by reducing the hot spots between and within the CPU sockets and minimizing the effect of thermal coupling. So in this way the overall performance of the multiprocessor system is increased.

II. LITERATURE REVIEW

We divide our literature section in two subsections. In first section, we will explain how last level cache memory is partitioned with our proposed methodology and in second section, the cooling management of CPU and memory subsystems will be demonstrated.

2.1 Partitioning Of Cache Memory

In a chip multiprocessor, a multi- level cache memory is used in which the last level cache is being the frequently used and often shared among all the cores on the chip. So for decreasing its energy consumption, partitioning of last-level cache memory is necessary. The introduction of our scheme (Cooperative Partitioning) has been already given in the last section which states that this achieved energy saving is two-fold. First dynamic energy can be reduced on each access because a core has to consult only its own ways. This method guarantees that data will never be found anywhere in the cache. Second when a whole way is unused by any core it can be turned off to save static energy.

Our scheme is split into two parts: first it monitors cache usage and determines the optimal partitions (fixed number of ways) for the running application. The second enforces the required partitioning and enable static and dynamic energy saving. Here the overview of cache partitioning can be explained by figure [2]. This figure shows that the core's interface to the ways from where data is to be read and write. This read/write operation on ways by core's can be performed in two phases. In the first phase, Last Level Access is monitored and partition decisions are made by processor that which core will access which way according to their needs. In the second phase, the ways are gradually migrated between cores and unused ways are turned off.

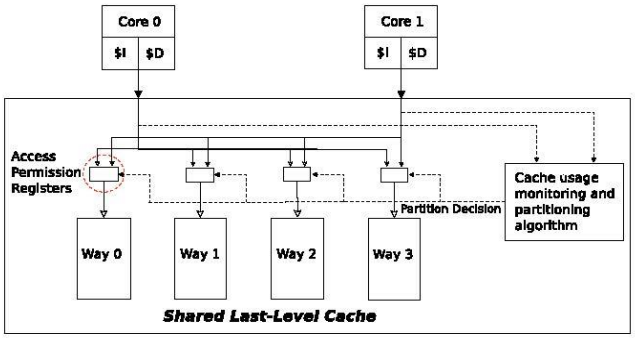

Figure 2. Data allocation across partitioning Schemes in shared last-level caches.

To perform this principle, we use Access Permission Registers to allow specific ways to read and write. These registers are termed as ***Read Access Permission Register (RAP)*** and ***Write Access Permission Register (WAP)***. By using these registers the core will interact with the ways whether it is allowed for read permission or write permission for a particular way, and to monitor it we use a modified UCP look-ahead algorithm which is used by cores to obtain extra ways when their performance increases above a threshold. This threshold value controls the decrease in miss ratio for each application and prevents each core to give more ways to go. Here both RAP and WAP both have one bit per core to indicate whether the core can read or write from the associated way.

For a particular way of core there are three modes of operation: first if RAP and WAP both are set then it can perform both read and write operations. Second if RAP is set and WAP is unset then it can perform only Read operation. Third if RAP and WAP both are unset then core will perform neither read nor write operation. Actually, here RAP and WAP serves three purposes: (1) they enforce cache partitioning. (2) They enable dynamic energy saving because it access only way for which it is permissioned. (3) When no core access any way then whole way is turned off for static energy saving. At a given time, only one core can have full access to a particular way but during transition period when reconfiguration is taking place, one core has full access (*RAP and WAP both are set*) whereas another core can has only *Read access*. This lasts until the whole way is takeover by another core by transferring its ownership. When RAP and WAP both are set then it is necessary to reconfigure the cache for new partitioning. To achieve it we use a new technique *Co-operative takeover* [4]. In this scheme, to transfer each way, the donor and recipient cores co-operate each other to quickly flush dirty data back to memory and allow recipient core to take over the whole way. This scheme quickly transfers ownership of whole way, enables fast realization of dynamic energy saving when no donor core access this way. During the transition periods, multiple cores access the data in a particular way then it takes more time to complete the accessing procedure. Hence it consumes more energy. It states that at transition period dynamic energy consumption of multicore processors is high than normal. So we wish to transfer ways as quickly as possible to minimize the length of time that the way is transitioning.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

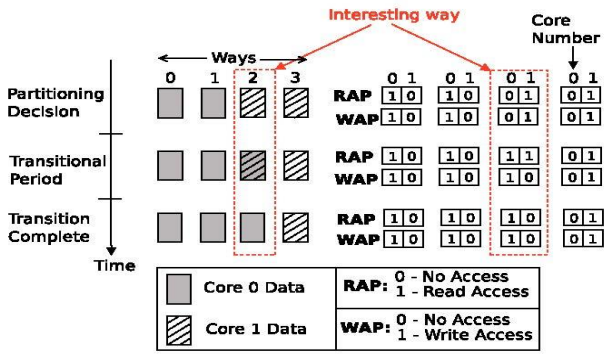

Figure 3. RAP and WAP register changes when transferring way 2 between cores.

To enable co-operative takeover, a particular bit vector is used for every line of the way. A bit vector is actually one bit per set per core. The donor bit vectors are reset at the start of the transition period. Whenever a donor core access the data from the particular set at a time, dirty data is flushed back to main memory and bit vector sets for that set. This happens whether it hits or misses on that particular access. When one bit vector is set and if one core has both read or write access on that set then another core will be able to do only read operation. Again in next turn a hit or miss operation will be performed. At that time dirty data will flush back to main memory and next bit will be set for any other set. And at the time when bit vector is set then ownership of the shared way will be transferred to another core. The concept of Cooperative takeover can also be explained with this example which is shown in figure5.

In this example there are *two cores and four cache ways*. Initially, the partitioning decision has been made and two ways are assigned to each core, but *core 1* wants to donate *way 2* to *core 0*. The takeover bit vector for core 1 is totally unset. In the second step, core 1 performs a read that hits in set c in the cache. Its own dirty data from this set in way 2 is flushed back to memory and the takeover bit is set. Following this, core 0 writes to the cache which misses in set b. In this case, core 1's dirty line from set b, way 2 is flushed and the corresponding takeover bit set as before. When the new line comes in from memory, it can be placed in way 2 instead of replacing an existing line in another way.

Core 0 then has a read hit in set d. In this case the line in way 2 is not dirty so does not need flushing, but the takeover bit is still set. In the fifth step, core 1 has a read hit in set b. However, the line in way 2 is now owned by core 0 and, even though it is dirty, does not need flushing back to memory. Core 1 can see this because the corresponding takeover bit is already set. Finally, core 1 has a read miss in set a. Again, no dirty data needs flushing, and the takeover bit is set. However, when the line comes into the cache, it will replace the data in way 3 (core 1's only way).

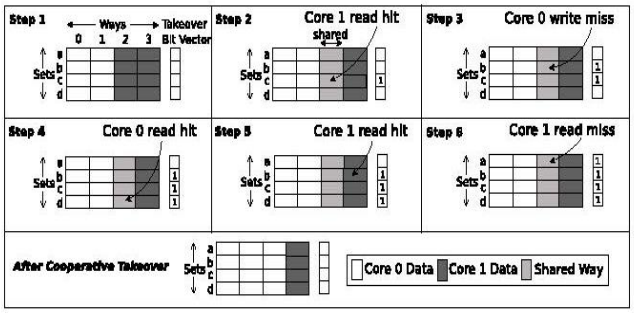

Figure 4. An example of cooperative takeover

At this point, the all takeover bits are set and therefore core 0 takes complete ownership of way 2. This is achieved simply by resetting the bit for core 1 in the RAP register for way 2, meaning that it no longer has read permission for that way (write permission had already been withdrawn). In this way the efficient partitioning of last level cache memory is performed which reduces both static energy and dynamic energy.

2.2. Thermal Management For Memory And CPU Subsystems

For the need of high performance, a wide server system with the multiple CPU sockets and large amount of memory subsystems are used which results higher power densities (dissipation), increase in operational cost of machine, it also causes thermal hot spots that cause substantial effect on reliability, performance and leakage power. It also requires a complex and energy hungry cooling subsystems. Traditionally it is assumed that CPU subsystem is the major source of power consumption in server system but Instead of CPU, the memory subsystems are the other major power hungry component of the system which consumes 35% of total system energy. Several designers have developed techniques to improve the efficiency of CPU subsystem but less attention is given to memory subsystems.

Some solutions are also proposed for memory subsystems but there are also several ignorance can be easily found. Like, in [5] a technique is used to activate a subset of memory modules but this approach increases the power density of active memory modules which can cause thermal problems in memory. In [6], memory throughput is adjusted to keep the memory temperature in safe zone but this solution does not consider the number of active memory modules. A number of core level dynamic thermal management schemes (DTM) have been proposed for managing thermal emergencies between the single CPU sockets but these techniques are limited to single CPU socket. Also, none of the techniques consider the dynamics of cooling subsystems and its energy costs which we have discussed earlier.

Now we will explain how thermal dependencies are created between CPU mad memory subsystems with the help of figure (5). Actually due to area and cost constraints, a single cooling fan subsystem is used to cool both the CPU and memory subsystems. Figure shows the diagram example of combined ***thermal and cooling model*** (Intel Quad Core dual Socket Xeon E5440 server) for CPU and memory

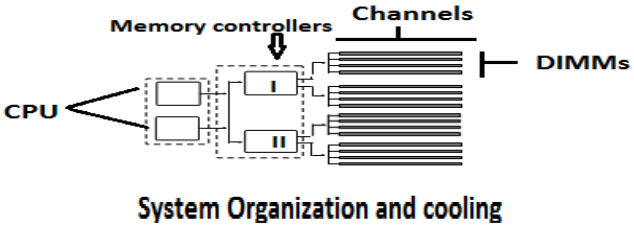

Figure 5

Where CPU is placed closer (upstream) to cooling fan while the memory is placed downstream.

This server supports two off-chip memory controllers where each is connected to memory by two memory channels; each channel is connected four DIMM (Dual in Memory Modules).

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Due to same cooling resource thermal dependencies between the CPU and memory are created. Cooling systems use fans to generate air flow to reduce the temperature of hot components. So, when cooling fans are used to reduce the temperature of heat sink of CPU subsystems, as we know that '*heat energy is neither created nor destroyed',* the heat energy of CPU subsystems is transferred to the memory subsystems. So the inlet temperature of memory is the function of its case temperature and transferred heat energy of CPU subsystems. . It means that the memory and CPU are thermally connected as the CPU heats the common air flow which is produced by the fan before it reaches the memory subsystem. As a result, *the temperature of the memory is a strong function of the hottest CPU and the inlet temperature.* Mathematically, cooling is mainly modelled through variable convective resistance which is a function of the air flow rate and expressed as:-

$$R_{conv} \propto \frac{1}{AV\alpha}$$

Where,

A= heat sink effective area

V= air flow rate

α = factor with a range (0.8 – 1.0)

For a given heat flow, the temperature of memory **(T)** is directly proportional to convective resistance:-

$$T \propto R_{conv}$$

To estimate the power dissipation of fan, we use the relation as:-

$$\frac{P_{V2}}{P_{V1}} = \left[\frac{V_2}{V_1}\right]^3$$

Where,

P_{v1} and P_{V2} represent the fan's power dissipation at V_1 and V_2 respectively.

It proves the thermal dependencies between the CPU and memory subsystems.

For these limitations, we present our model which offers improvements in server power efficiency by dynamically reducing the cooling and memory energy costs. It maximizes the energy efficiency in the server by controlling the number of active memory modules while minimizing the cooling costs. It also schedules the workload between CPU sockets to create a more balanced thermal distribution which reduces the thermal spots and minimize the thermal coupling effect. ***Integrated solution for both CPU and memory subsystem is necessary due to thermal dependencies between them when both share same cooling resources.***

For performing these operations, we proposed a model whose architecture consists of ***controller, two actuators (memory and CPU) and sensors (temperature and power)*** which are shown in figure 6. First the controller analyses the data provided by the sensors to arrive at appropriate decisions and then these decisions are used for implementing the actuators according to the situation. The decision takes into account thermal dependencies between CPU sockets and memory. The controller is implemented in the operating system layer.

Now we will explain the functionalities of each section of architecture of our model:

Controller: JETC actions are modeled using a state machine with four discrete states to handle four distinct control scenarios of memory, CPU and fan speed. The transitions between states are triggered as a function of *fan speed, temperature (of CPU and memory) and the number of active memory modules.* The four discrete states are:-

1. **State S_c** – In this state, model activates only the CPU actuator when there are thermal problems in CPU subsystem but no thermal problems in the memory.
2. **State S_m –** In this state, model activates only the memory actuator when there are thermal problems in the memory subsystem but no thermal problems in the CPU.
3. **State S_i –** When there are no thermal problems in the system and fans are balanced then model does not need to act so it can stay in idle state. In this state, no socket or memory scheduling is performed and the minimum number of active memory modules (in general, one DIMM per memory channel) is set to maximize bandwidth.
4. **State S_{mc} –** In this case, there is a need of a joint action from CPU and memory actuators to ignore thermal problems. The difference in heat generation disturbs the thermal distribution of memory due to thermal coupling that may lead to imbalance in speed of fans. So a transition in this state can be made possible in the following scenarios: - (i) imbalanced fan speed with thermal problems in memory (high temperature or number of active memory modules is higher than default (ii) temperature exceeds threshold in both CPU and memory.

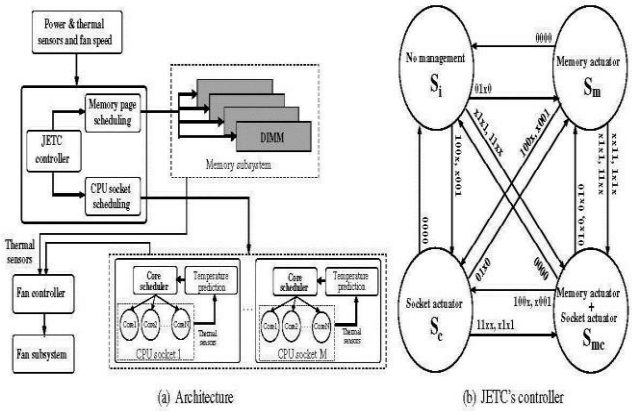

Figure 6. PROPOSED COOLING AND THERMAL MANAGEMENT MODEL

CPU actuator: It can reduce the cooling energy of the memory by creating a balanced thermal distribution in memory since CPU and memory are thermally connected. Initially the CPU actuator calculates the desired change in power values in each of the CPUs to balance fan speed and to reduce the cooling energy of the memory subsystem. It traverses the workload from the hot CPU to cooler threads to have a fine control of the total power on each socket and to reduce the chance of errors.

Memory actuator: This actuator keeps the temperature of memory modules within the safe zone while minimizing energy. It also accounts for fan speed and CPU heat generation since they affect the memory temperature. Actually, this actuator either decreases or increases the power dissipation of memory modules by adjusting the number of active memory modules while rest are in self-refresh mode.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

When temperature is high and there are no active modules which can accept extra additional memory pages, then we choose b/w activating a new memory module and letting fan to cool it which depends on the energy budget.
The problem of thermal dependencies is resolved as follows: When there is an imbalance in speed of fans, the model activates the CPU actuator to perform scheduling to minimize the difference in heat generation between CPU sockets by migrating workload from a hot CPU to a cooler one. This balances the temperature across the memory modules. At the same time, the Memory actuator performs energy aware thermal management by activating just what is necessary of memory modules to meet thermal and performance constraints.

III. IMPLEMENTATION & RESULTS

There are two simulators needed for the two operations which are discussed in last sections. For partitioning the last level cache memory we need ***Marss-x 86 simulators*** which is worked on the LINUX environment. We modeled both a two-core and a four-core system to fully evaluate the effects of sharing and partitioning the last level cache. All level 1 caches are private and all processors share a common level 2 cache. We model the DRAM conflicts and bus queuing delays and use Cacti [29] at 45nm to get energy information. Finally, we assume a 5 million cycle phase interval for monitoring and partitioning decisions, as in prior works. We evaluated this partitioning in terms of performance and energy consumption. With this method, approximately 35% of total average dynamics energy and 28% of total static energy can be reduced.
Now for the implementation of the cooling management model for memory and CPU subsystems ***Hot Spot Simulator*** is used which is also worked on the Linux environment. Initially we experimented the workload in our server and collect the real power traces from memory modules. These results are used to estimate the temperature by using the given Hot Simulator .We also measured used CPU core base by using the method described in [5]. With this method, the total 55% average energy reduction in cooling and memory subsystems is observed (with less than 0.9% performance overhead). In this way, with the employment of these two methods the total energy of the system can be reduced to greater extent.

IV. CONCLUSION

Our proposed technique of partitioning the last level cache (LLC) memory maintains the high performance while saving significantly both static and dynamic energy. It forces the cache data to be way aligned so that a core has to consult only those which it has owns by which migration id five times faster than earlier approaches. We have also considered the thermal dependencies between the CPU and memory subsystems due to shared cooling resources between them. In this way the overall energy and thermal efficiency of server system is increased.

V. ACKNOWLEDGMENT

Authors are thankful to ABV-Indian Institute of Information Technology and Management, Gwalior for providing the infrastructure support for carrying out present research work

REFERENCES

[1] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *MICRO*, 1999R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors:A machine learning approach. In *MICRO*, 2008
[2] Karthik T. Sundararajan, Vasileios Porpodas, Timothy M. Jones. Cooperative Partitioning: Energy-Efficient Cache Partitioning for High-Performance CMPs *HPCA 2011*
[3] Ajami, K. Banerjee, and M. Pedram. Modeling and analysis of nonuniform substrate temperature effects on global interconnects. *IEEE Trans. on CAD*, pages 849–861, 2005
[4] R. Ayoub, K. R. Indukuri, and T. S. Rosing. Temperature aware dynamic workload scheduling in multisocket cpu servers. *TCAD*, 30(9):1359–1372, 2011.
[5] Raid Ayoub Rajib Nath Tajana Rosing. JETC: Joint Energy Thermal and Cooling Management for Memory and CPU Subsystems in Servers *HPCA 2011*
[7] K. Flautner, N. S. Kim, S. M. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In ISCA, 2002.
[8] M. Ghosh, E. Ozer, S. Ford, S. Biles, and H.-H. S. Lee. Way guard: a segmented counting bloom filter approach to reduc- ing energy for set-associative caches. In ISLPED, 2009.
[9] F. Guo, Y. Solihin, L. Zhao, and R. Iyer. A framework for providing quality of service in chip multi-processors. In MI- CRO, 2007.
[10] L. R. Hsu, S. K. Reinhardt, R. Iyer, and S. Makineni. Com- munist, utilitarian, and capitalist cache policies on CMPs: Caches as a shared resource. In PACT, 2006.
[11] R. Iyer. CQoS: A framework for enabling QoS in shared caches of CMP platforms. In ICS, 2004.

Vikash Sharma has received the Bachelors of Engineering degree in Electronics from Madhav Institute of Technology and Science Gwalior (M.P.) in 2012. He is currently pursuing Masters from ABV-Indian Institute of Technology and Management Gwalior (M.P.) with specialization in VLSI design. His main research interests are VLSI Design, antenna design, thermal Modeling, Signal processing, and testing and testability of circuits. He is currently working on fault collapsing techniques and reduced test generation methods with the help of implication fault graph. He has also worked on the designing of patch antenna at the bachelor's level.

Jayant Kumar has received the B.Tech. degree from BBDNITM Lucknow (U.P) in Electronics and Communication Engineering in 2012 and currently pursuing M. Tech. degree from ABV- Indian Institute of Information Technology & Management, Gwalior (M.P.) with specialization in VLSI Design. His main research interests are Computer Architecture, Digital Signal Processing and MOS VLSI Design. He is currently working on Approximation of Adders and Process Variation. He has also worked on Current Source Modeling.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)