ADAPTIVE BLOCK PINNING BASED: DYNAMIC CACHE PARTITIONING FOR MULTI-CORE ARCHITECTURES

Nitin Chaturvedi[1] Jithin Thomas[1], S Gurunarayanan[2]

[1]Birla Institute of Technology and Science, EEE Group, Pilani, India
nitin80@bits-pilani.ac.in
[2] Birla Institute of Technology and Science, EEE Group, Pilani, India
sguru@bits-pilani.ac.in

ABSTRACT

This paper is aimed at exploring the various techniques currently used for partitioning last level (L2/L3) caches in multicore architectures, identifying their strengths and weaknesses and thereby proposing a novel partitioning scheme known as Adaptive Block Pinning which would result in a better utilization of the cache resources in CMPs. The widening speed gap between processors and memory along with the issue of limited on-chip memory bandwidth make the last-level cache utilization a crucial factor in designing future multicore processors. Contention for such a shared resource has been shown to severely degrade performance when running multiple applications. As architectures incorporate more cores, multiple application workloads become increasingly attractive, further exacerbating contention at the last-level cache. Several Non-Uniform Cache Architecture (NUCA) schemes have been proposed which try to optimally use the capacity of last-level shared caches and lower access times on an average. This is done by continually monitoring the cache usage by each core and dynamically partitioning it so as to increment the overall hit ratio.

KEYWORDS

Chip Multiprocessor, Non-Uniform Cache Architecture, L2 Cache

1. INTRODUCTION

There has been a tremendous development in silicon technology consequently leading to a high density packing of transistors on chips. Each new process technology increases the integration density thus allowing for higher clock rates and also offers new opportunities for micro-architectural innovation. It has facilitated the integration of multiple cores onto a single chip. Taking the present trend into account, the number of cores on a single chip will definitely keep increasing in the future. This increment in the available processing power proves quite attractive for multi-application workloads, which in turn exerts additional pressure on the memory, especially the on-chip memory system. Two major reasons for this are the increasing speed gap between processors and the main memory as well as the limited on-chip bandwidth available [1]. Most viable among the possible solutions is to use the last level shared cache in a more efficient manner by reducing the expenses in terms of latency, power and requests to off-chip memory. Previous research has shown that a last-level multi-core cache can be organized as private partitions for each core or having all cores sharing the entire cache. Previous results show that the shared cache organization can be utilized more flexibly by sharing data between cores. However, it is slower than a private cache organization. In addition, private caches do not suffer from being polluted by accesses from other cores by which we mean that other cores displace blocks without contributing to a higher hit rate. Non-uniform cache architectures (NUCA) are a proposed hybrid private/shared cache organization that aims at combining the

better of the two extreme organizations by combining the low latency of small (private) caches with the capacity of a larger (shared) cache. Typically, frequently used data is moved to the shared cache portion that is closest to the requesting core (processor); hence it can be accessed faster [2]. Recently, NUCA organizations have been studied in the context of multi-core systems as a replacement for a private last-level cache organization [3, 4]. As a result, on a miss in one partition, all other partitions are first checked before accessing main memory. While this hybrid scheme provides fast access to most blocks, it can suffer from pollution because of the uncontrolled way by which partitions are shared among cores [1].

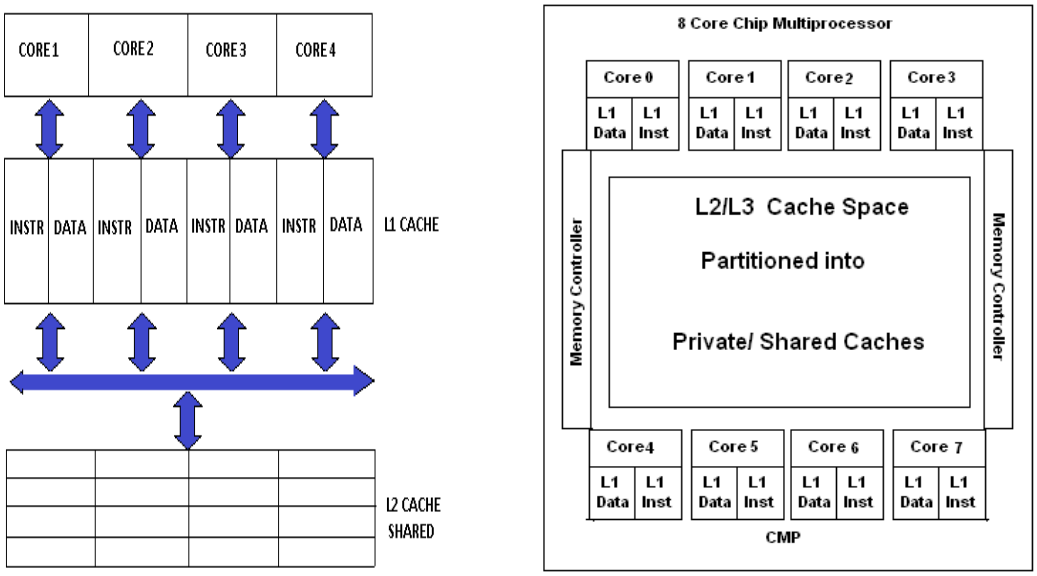


Figure 1. Multi-level Cache for CMP

In this paper we propose a novel technique to dynamically partition last-level caches known as Adaptive Block Pinning. It attempts to partition the cache into two major portions: a block pinned portion and a set of POP caches. The block pinned portion is partitioned on a set granularity. The rest of the report is organized as follows. The major NUCA architectures that have been studied are presented in Section 2. The motivation leading to the development of the novel scheme proposed in this paper is detailed in section 3. A novel portioning scheme based on the ones presented in Section 2 has been proposed in Section 4. Proposed implementation details and simulation methodology is presented in Section 5. The benchmark suite to be used is mentioned in Section 6 and Section 7 concludes.

2. RELATED WORK

The Kim et al. [5] introduced the concept of Non-Uniform Cache Architecture (NUCA). They observed that increasing wire delays would mean that cache access times were no longer constant. Instead, latency would become a linear- function of the line's physical location within the cache. On the basis of this observation, they designed several NUCA architectures by partitioning the cache into multiple banks and using a switched network to connect these banks.
Dybdahl and Stenstrom [1] proposed a novel non-uniform cache architecture in which the amount of cache space that can be shared among the cores is controlled dynamically. The adaptive scheme estimated, continuously, the effect of increasing/ decreasing the shared partition size on the overall performance. A sharing engine implemented the sharing of the last-level cache among cores. Each core had 3 levels of cache, with L3 cache being at the last level.

The shared L3 cache is statically partitioned into local caches of each core. Each local cache had a private as well as a shared partition. The L3 cache usage was controlled in two ways: (a) a part of the cache was private and inaccessible by the other cores, (b) the size of the private cache partition and the number of cache blocks in the shared partition of the cache was controlled on a per-core basis in order to minimize the total number of cache misses. The partition sizes were dynamically controlled by estimating two factors:

1. The gain in increasing the cache size per application/core
2. The loss in decreasing the cache size per application/core

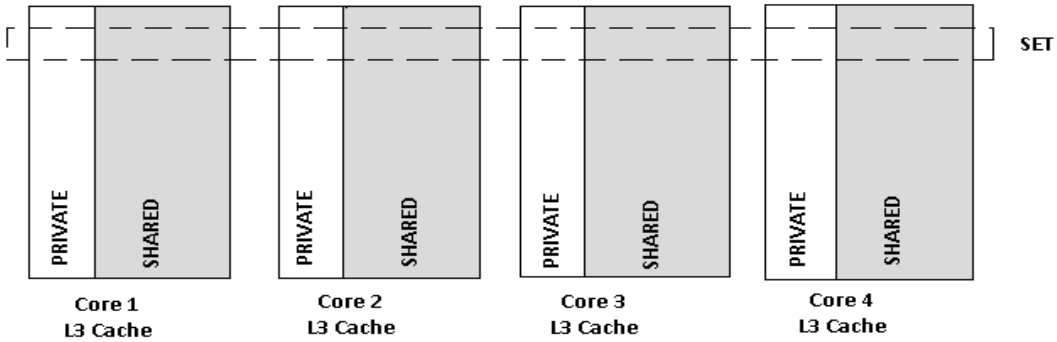
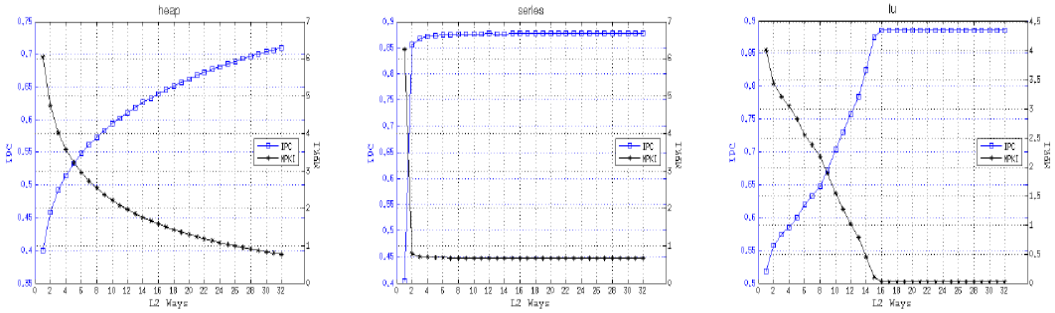


Figure 2. Each L3 local cache has a private and shared partition [1].

The former was estimated with the help of special registers known as shadow tags. Each set had a shadow tag for each core. Whenever a cache block was evicted from the L3 cache, the tag of that particular block was stored in the shadow tag associated with the core that had fetched the block into the L3 cache. On a cache miss, the tag of the miss is compared to the shadow tag for that set. If there is a match, a counter, namely 'hits in shadow tags' for the requesting core is increased. Thus, this counter associated with each core would indicate the number of L3 cache misses that could have been avoided had that particular core been given one extra block per set. The latter was estimated in the following manner. Each time there was a hit in the LRU block for the requesting core, a counter was increased for that core. This counter represents the number of cache misses that would have occurred if the cache size is reduced by one block per set. The sharing engine, mentioned above, re-evaluated the private partition sizes on a per-core basis. The core with the highest gain for increasing the cache size, i.e. the core with most hits to its shadow tags, was compared to the core with the lowest loss of decreasing the cache size, i.e. the core with the fewest hits to its LRU blocks. If the gain was higher than the loss, one cache block (per set) was provided to the core with the highest gain. The counters were reset after each reevaluation period. Srikantaiah et al. [6] proposed a novel dynamic partitioning scheme for managing shared caches in Chip Multi Processors, known as Adaptive Set Pinning. Set pinning was suggested on the basis of two crucial observations about the characteristics of non-compulsory misses in shared caches. The number of distinct memory addresses in the references that lead to inter-processor and intra-processor misses were measured. It was observed that a very low fraction of distinct memory addresses were responsible for inter-processor misses. This implied that most of the inter-processor misses happened due to a few selective blocks of memory, which were termed as 'hot blocks'. The second major observation was that these hot blocks were quite frequently accessed as well.Set pinning was designed to exploit these two observations by disallowing the large number of references for the few *hot blocks* responsible for causing inter-processor misses, from evicting L2 cache blocks. Instead, these *hot blocks* were stored in very small regions of the L2 cache, confined to be written by individual processors, called *Processor Owned Private* (POP) caches. Srikantaiah et al. also proposed a

technique called *adaptive set pinning* which improved the benefits obtained by set pinning, by adaptively relinquishing ownership of pinned sets.

Nikas et al. [5] proposed yet another partitioning scheme for Multi-core Architectures known as Adaptive Bloom Filter Cache Partitioning (ABFCP). The major idea that led to the development of this scheme is the fact that a key factor to the efficiency of a shared cache is the line replacement policy. It was observed that majority of CMP systems today employ the least recently used (LRU) policy, widely accepted as the best line replacement policy in uni-processor caches. The LRU policy, in the context of a shared cache, implicitly partitions the cache by allocating more space to the application with the highest demand, i.e. many accesses to different entries. However, not all applications benefit from exploiting additional cache resources. In a multicore system several applications will be running in parallel and the benefit to each one of obtaining additional cache resources will vary. Applications also exhibit distinct phases during execution, each with varying cache requirements. It is here that the concept of an application's cache utility comes into picture. It is the amount of performance gain with respect to an application in response to additional cache space allocation to it. As presented by Qureshi and Patt [8], this cache utility metric can be used to classify applications into three categories; high, low and saturating. High utility applications continue to benefit significantly from increases in cache resources. On the other hand, low utility applications gain little or no performance gains from increasing the cache space as the number of their misses remains stable.



(a) High Utility (b) Low Utility (c) Saturating Utility

Figure 3. Different cache utility profiles [7].

A Bloom filter is a structure for maintaining probabilistic set membership. It trades off a small chance of false positives for a very compact representation. A partitioning module is attached to the L2 shared cache and monitors all core accesses. The partitioning module needs to track the actual cache occupancy of each application running in each core, and therefore a core ID field is added alongside the address tag of each cache line. Whenever a core brings a line into the cache, its ID is stored in the core ID field. Bloom filters are used to monitor specific subset of cache misses known as *far-misses,* ie, those that would have been hits had the core been allowed to use more cache ways. The novel scheme that we propose attempts to combine the advantages of the various schemes described so far while at the same time eliminating their disadvantages.

3. NOVEL SHARED L2 PARTITIONING SCHEMES

3.1. Block Pinning

This novel scheme makes use of two important concepts: block pinning and POP caches. Block pinning refers to a cache management scheme where in each block's ownership is assigned to a particular core. Ownership of a block simply means the right to evict that particular block from memory. This scheme would definitely improve upon the performance benefits achieved by set

pinning by enforcing a different partition for each set instead of adopting a uniform partition for the entire cache. In effect it would reduce the number of intra-processor misses to a greater extent. Processor Owned Private (POP) caches would be used to store the *hot blocks* described above thereby eliminating inter-processor misses. Thus the entire L2 cache is divided into two major portions: a block pinned shared portion and a set of POP caches. In the block pinned portion each block has an additional field which stores the identifier of the core that currently owns the block. When there is no tag match in either the indexed set of the block pinned L2 cache or any of the POP caches, it is an L2 cache miss. There are three major cases to be considered in case of a cache miss:

(i) There is at least one block in the indexed set in the L2 cache not owned by any processor.

(ii) There is at least one block in the indexed set in the L2 cache owned by the requesting processor.

(iii) Every block in the indexed set in the L2 cache is owned by some processor other than the requesting processor.

In the first case, the referenced data is brought into the free block nearest to the requesting core. Moreover, the owner field of the block is set to the ID of the requesting core. In the second case, the LRU block among the blocks owned by the requesting core is evicted and the referenced data is brought into this block. Intra-processor misses can be significantly reduced because of reduced number of references that are eligible to evict data (limited to references from the owner processor). In the third case, no block can be evicted by the requesting core. This reduces the inter-processor misses which could only occur if the eviction had taken place. The POP cache owned by that processor is instead used to store the referenced data by evicting the LRU entry from that processor's POP cache. There is no replication or migration of cache blocks between the block pinned L2 cache and the POP caches. Every cache block is present in either the block pinned L2 cache or the POP cache of the processor which references it first. In L2 shared cache, colour of the block depicts the corresponding core it belongs to, white block depicts no owner.

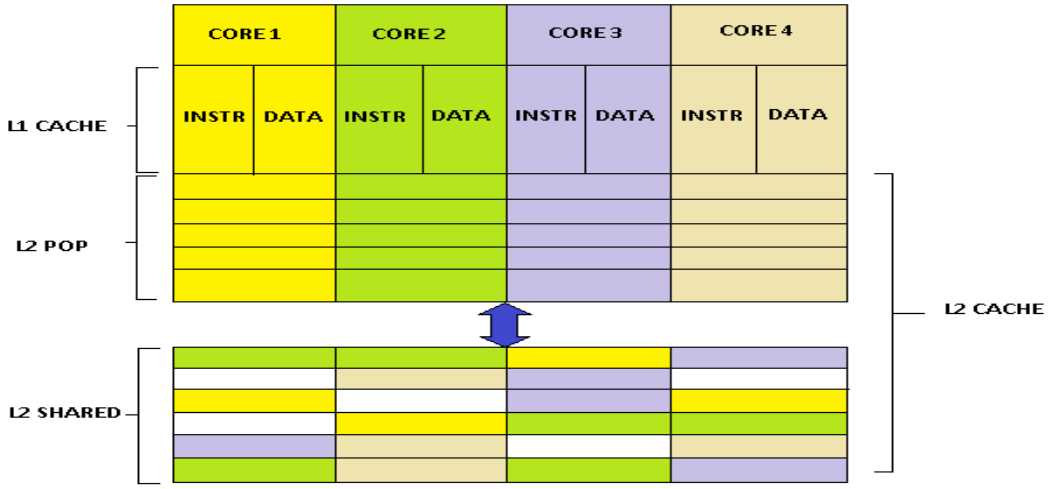


Figure 4. In L2 cache, colour of the block depicts the corresponding core

3.2. Adaptive Block Pinning

As in set pinning, the simple first–come first–serve policy of block pinning favours the processors that first access a block. This can lead to acquirement of ownership of a large

number of blocks by the first few active processors. Such domination in ownership of blocks by a few processors leaves few blocks for the other processors and hence over stresses the POP caches of those processors. Such domination arises because once owned by a core, a block can never be owned by any other core.

Thus, we need to come up with an adaptive policy where cores dynamically relinquish ownership of blocks. In adaptive set pinning, this was done with the help of a *confidence counter* for each set which indicated the confidence of the system that the current processor should be the owner of the set. However, following the same suit in our scheme would result in maintaining a confidence counter for each block leading to a substantial storage overhead which would negate the benefits achieved by our present scheme. Hence an alternative method would be used to measure this confidence. This makes use of *shadow tag registers* which were used in the adaptive NUCA architecture suggested by Dybdahl and Stenstrom [1]. Corresponding to each set, shadow tag is maintained for each core, which stores the tag of the block that was last evicted by that particular core. In addition to this two counters are maintained per set for each core.

(i) C_{gain} : Each time a cache miss occurred, the tag of the requested block is compared to the shadow tag. In case of a match, this counter is incremented.

(ii) C_{loss} : Each time a hit occurs on the LRU block this counter is incremented.

At the end of constant monitoring intervals, the partitioning algorithm would be invoked. The linear partitioning algorithm presented as part of ABFCP [6] is used for this purpose.

Algorithm 1 Linear Partitioning Algorithm

```
cores = N
for core i = 0 to N do
   gain_1[i] = a × C_far−miss
   lose_1[i] = C_LRU
end for
order gain_1, lose_1 from min to max
while max gain_1 > min lose_1 do
   increase allocation[i] by 1, decrease allocation[j] by 1
   remove i and j from gain_1, lose_1
   cores -= 2
   if cores ≤ 1 then
      return allocation;
   end if
end while
return allocation;
```

Figure 5. Linear Partition Algorithm [6]

The algorithm reads the *gain* and *loss* counters of each core. Then, in each iteration, the maximum gain value is compared against the minimum loss value. If the former is greater, then the allocation of the core associated with that C_{gain} counter is increased by one way, while the core associated with that C_{loss} counter is deprived of one cache way. The process is repeated until no cores are left to be considered or the maximum gain value is smaller than the minimum loss value. In the worst case N/2 comparisons need to be performed, where N the number of cores. Therefore, the complexity of this algorithm is O(N).

Now in case of a cache miss, suppose that there is no free block in the indexed set, ie, one which is not owned by any processor. In this case, the replacement mechanism counts the number of cache blocks in the accessed set that belong to the miss invoking core. If this number is equal to or greater than the number imposed by the partitioning algorithm then the referenced data is placed in its POP cache by evicting the LRU block in the POP cache. Otherwise, the LRU block of an over-allocated core is evicted and the ownership is taken up by the miss invoking core.

Thus, if the number of ways allocated to a core is increased, then the new ways are consumed only on cache misses. This lazy reallocation allows the cache to retain the previously stored cache blocks until the space that they occupy is actually needed.

4. SIMULATION METHODOLOGY

For evaluating the performance of the CMP requires a way of simulating the environment in which we would expect these architectures to be used in real systems. We will use Virtutech Simics [9] full system functional simulator extended with Multi-facet GEMS which is popularly used in the research community. The heart of GEMS is the Ruby memory simulator. The Ruby Cycle is our basic metric of simulated time used in the Ruby module of the GEMS simulator. The Ruby module is the basic module for the memory system configuration and interconnection network design The Ruby cycles are the recommended performance metric in GEMS. The base line configuration to be used is given below

Number of cores	8
Core processor	Out-of-order SPARCv9
Main memory size	4 GBytes
Memory bandwidth	512 bytes/cycle
On-chip wire delay	1 cycle
Off-chip wire delay	20 cycles
Switch delay	1 cycle
Private L1 data caches	8 KBytes
Private L1 instruction caches	8 KBytes
Shared L2 NUCA cache	1 MB

Figure 6. Base line Configuration

5. PARSEC BENCHMARK SUITE

The Princeton Application Repository for Shared-Memory Computers (PARSEC) has been recently released [10]. This benchmark suite includes emerging applications and commercial programs that cover a wide area of working sets and allow current Chip Multiprocessor technologies to be studied more effectively. In this paper we will evaluate a subset of the PARSEC benchmark suite. We also consider the simlarge inputs of PARSEC benchmarks in our simulations. We will fill the cache by executing 100 million instructions and finally we collect the statistics for the following 500 million instructions.

6. CONCLUSION

The current advanced submicron integrated circuit technologies require us to look for new ways for designing novel microprocessors architectures and non-uniform cache architectures to utilize large numbers of gates and mitigate the effects of high interconnect delays. In this paper, we have discussed a novel dynamic partitioning scheme known as Adaptive Block Pinning which attempts to partition the last-level shared cache in a judicious and optimal manner thereby increasing overall system performance. Future work could be directed at modifying this scheme to work for multi-banked caches. It could also aim at reducing the latency penalties by attempting to place each cache block nearest to the core that most frequently uses it. In order to analyze the impact of various parameters on the performance of Multi-core Architectures, we will vary the number of cores, which in turn changes local L2 cache size, to see the scalability of the proposed architecture.

REFERENCES

[1] Haakon Dybdahl and Per Stenstrom. An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors. In *Proceedings of the 13th Annual International Symposium on High Performance Computer Architecture, 2007.*

[2] Nitin Chaturvedi and Pradip Harindran. A Novel Shared L2 NUCA Cache Partitioning scheme for Multi-core Architectures. International Conference on Emerging Trends in Engineering ICETE- Feb-2010

[3] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. ISCA, 2006.

[4] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in CMPs. ISCA, 2005.

[5] Shekhar Srikantaiah, Mahmut Kandemir and Mary Jane Irwin. "Adaptive Set Pinning: Managing Shared Caches in Chip Multiprocessors". Proceeding of ASPLOS'08, ACM/IEEE, pages 135-144 March, 2008.

[6] Konstantinos Nikas, Matthew Horsnell and Jim Garside. An Adaptive Bloom Filter Cache Partitioning Scheme for Multicore Architectures. IEEE Explore-2008.

[7] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high performance runtime mechanism to partition shared caches. In Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pages 423–432, Orlando, Florida, USA, 2006.

[8] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F.Larsson, A. Moestedt, and B. Werner. *Simics: A Full System Simulator Platform*, volume 35-2, pages 50–58. Computer, 2002.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *International Conference on Parallel Architectures and Compilation Techniques*, 2008.

Nitin Chaturvedi

Received M.Sc (Electronics) from Devi Ahilya University Indore in 2002 and M.Tech from Panjab University Chandigarh in 2005. Presently working as a Lecturer and pursuing Ph.D on Multi-cores Architectures from Birla Institute of Technology and Science, Pilani, India. His field of interest is Computer Architectures, Digital CMOS Circuits.

S Gurunarayanan

Received M.Sc Physics, M.E and Ph.D. from Birla Institute of Technology & Science Pilani, India. Presently, he is working as Professor having more than 22 years of teaching experience in Electrical Electronics Engineering Group, Birla Institute of Technology and Science, Pilani, India. He is working in the field of reconfigurable computing, Computer Architectures, VLSI System Designs.

Jithin Thomas

Jithin Thomas is final year student of B.E (Computer Science) Birla Institute of Technology & Science pilani, India.