International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
IJISAE
ISSN:2147-6799 www.ijisae.org Original Research Paper

Memory Optimization Using Distributed Shared Memory Management for Re-engineering

R. N. Kulkarni [1], Venkata Sandeep Edara*[2]

Submitted: 23/04/2023 **Revised**: 25/06/2023 **Accepted**: 02/07/2023

Abstract: Restructure the old input C++ programming system to make it suitable for reengineering, which necessitates humongous memory. This study has made significant contributions to the disciplines of system-on-chip design and on-chip memory management, among others. This paper presents a new technique for optimizing memory subsystems during system design that is suited to specific applications like re-engineering. This work can be improved to optimize memory controller activities and design challenges for distributed shared memory management. It has the capacity to manage an increasing number of processors simultaneously, as well as security and faster execution. The Xilinx 14.2 integrated simulation environment produces outstanding results when the proposed memory optimization approaches are fully integrated into the simulation, optimization, and code generation cycle. The proposed memory device has an extremely low on-chip power consumption of 0.0082 W, which amply demonstrates how well memory has been improved.

Keywords: *Memory management, Power Consumption, Shared distributed memory (SDM), Memory mapping Mechanism, Re-engineering*

1. Introduction

This A variety of high-speed applications, such as consumer embedded products, audio and video processing, multimedia animation (including 3D animation), and weather forecasting, were created by IC designers as a result of the development of VLSI technology [1]. Field programmable gate arrays (FPGAs) have only lately demonstrated their viability in decreasing the overall computational load placed on the CPU core of the primary processor. Data on the reconfigurable fabric, which is used by both the software and the hardware, is mapped using a memory mapping technique designed specifically for this resolution. This memory mapping approach is the foundation of the entire methodology. Parallel and distributed processing are two of the most cutting-edge areas of research being investigated right now in computer architecture. The shared memory architecture is generated in systems with physically distributed memories by incorporating the two aspects of both approaches. One of the most widely researched subjects in multiprocessor networks is the study of distributed shared memory. Traditional single-processor architectures are unable to push ahead with the expanding number of high-speed applications, requiring the creation of extremely sophisticated multiprocessor architectures. A multiprocessor, also recognised as a Multi core SoC or Multi processor System on Chip, is a device that enables the integration of multiple Intellectual Property (IP) cores in a single chip, which could be in the form of high-speed digital signal processors, controllers, hardware accelerators, memory blocks, and I/O blocks to perform multiple tasks while operating at different clock frequencies.

Memory-management components of the operating system split each memory resource and load data across multiple types of memory at the system level. Part of the main-memory content, for example, must be shifted to secondary storage, the disc. Virtual-memory and physical-memory layouts, redeployment to avoid fragmentation, and other more sophisticated jobs are among the more difficult tasks. The main memory is split into 2 parts: one for the operating system to handle globally, and one for each programme being performed, which requires a different type of memory management [2].

In real-time operating systems, memory management is handled in one of two ways: using a stack or using a heap. The stack management technique is used for context switching between jobs, whereas the heap management mechanism is used for dynamically giving space to tasks [3]. Even many researchers have investigated the most recent methodologies for co-optimizing test time and test power reduction techniques. These actions that incur overhead are seen as vital after prefetching, even if they are not finished. To limit the amount of time spent reconfiguring them, they must be re-used (In future iterations of the same task graph execution). As a result, reusing previously loaded configurations is one of the most

1 Ballari Institute of Technology and Management, Ballari- 583104, Karnataka, India. ORCID ID : 0000-0002-9948-1398
2 Sasi Institute of Technology and Engineering, Tadepalligudem- 534101, Research Scholar, VTU Belagavi, Karnataka, India ORCID ID : 0009-0004-6318-6735
** Corresponding Author Email: evsandeep@sasi.ac.in*

cost-effective ways to reduce overall reconfiguration expenses [4].

Kandemir et. al. extend their prior work to multi-processor systems in [5] by using a compiler-based scratchpad management mechanism. Array-dominated embedded applications, as previously stated, are the most common use case. When several processors are working in the same array or collecting data, data reuse is a critical notion for decreasing energy and delay. Se-Jun Kwon et al. (2017) proposed an efficient and effective fast compression technique for in-memory data [6]. The suggested technique takes advantage of the features of in-memory data that are often identified to increase efficiency.

Dynamic Frequent Pattern Compression was developed by Yuncheng Guo et. al. as an adaptive non-volatile memory write approach based on frequent pattern compression [7]. The contents of cache lines are encoded using more types of common data patterns when utilizing DFPC, resulting in a greater compression ratio. In addition to regular metrics and cache hit rates, Esha Choukse et. al. included proposed mechanisms to incorporate correct virtual address translation, identify a region in the programme that is reflective of the compression ratio, as well as regular metrics and cache hit rates, in their methodology [8]. Dual dictionary compression was introduced as a last-level cache compression solution by David Kaeli [9]. It is feasible to make greater use of available die space while also boosting system memory speed by compressing data in the final level cache (LLC).

Memory is a vital component of real-time applications due to the amount of time and money it takes to manage it properly. As a result, real-time system designers concentrate on minimising the worst-case execution time and memory consumption. A cost-effective memory manager is required in order to increase the performance of the real-time system. Most real-time systems employ static memory allocators for the random allocation and de-allocation of memory blocks, and this is the case for the most majority of such systems. The memory is allocated before the application enters the core real-time stage, in which the full physical memory is available as a single block and can be used as needed, at the time of compilation or launch of the programme, whichever occurs first. There is a problem with excessive memory consumption since the memory required to hold objects cannot be easily recovered because automatic memory management is not available. Programmers must create and manage distinct private memory pools in order to avoid wasting memory space on unneeded operations. Real-time systems based on multicore and multiprocessor architectures have grown in size and complexity, necessitating the flexible use of existing resources, such as memory [10], as a result of this growth. For real-time applications to achieve the predictable performance and flexibility necessary in the multiprocessor system era, dynamic memory management allocators will eventually be required [11], [12].

The process of uncovering the technological principles of a technology, object, or system by analyzing its structure, function, and operation is known as reverse engineering [13]. Reverse engineering binary executable code has proven effective in a variety of situations. It's used to move a system to a newer platform, unbundle monolithic systems into components and utilise them separately, and interpret binary code for untrusted code and malware. The ubiquitous use of C++ in many modern applications necessitates an understanding of the disassembly of C++ object-oriented code. Today, reverse engineering binary code is common, particularly for untrusted code and malware. Software re-engineering is the process of improving or changing existing software so that it can be understood, managed, and reused as new software. Re-engineering is required when the system's software architecture and platforms become absolute and must be modified. The value of software re-engineering stems from its capacity to recover and reuse items that already present in an out-of-date system. This will definitely reduce system maintenance costs and lay the groundwork for future software development. Furthermore, re-engineering needs huge memory. This paper presents design and optimization of distributed memory for re-engineering.

Memory access is a major performance constraint in many data-intensive applications. The system's performance and power usage are both impacted by delays in access. As a result, a memory hierarchy is the most popular solution to alleviating this bottleneck and ensuring efficient memory design. Between the CPU and the main memory, various levels of memory are utilised. It's common practise to keep small, quick memories close to the central processing unit. Memory instances get larger and slower as you go down the memory hierarchy. With growing memory size, energy efficiency declines in the same way that it has been previously stated. As can be seen in Figure 1, embedded and SoC devices use a variety of memory hierarchy systems.

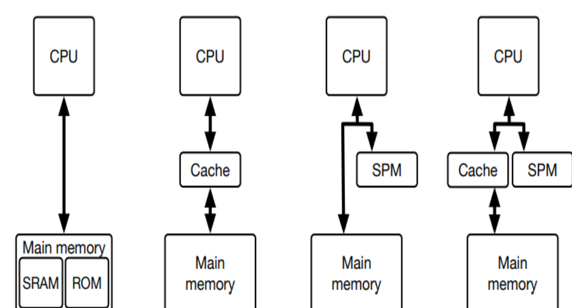

Fig. 1 Common memory architecture.

There is a unique global naming strategy in place to keep such disputes at bay for all of the sites that use the shared data. One way to get access to remote computers' shared

data is by having the distributed shared memory system's memory controller do physical-to-logical address translation. A global directory of shared data items will be maintained by the memory controller for all sites. Here, the memory controller is a software layer that sits on top of the local memory and allows for virtual shared memory. Fig. 2 shows how the system's memory controller will link to all of the system's local RAM and manage the system's virtual memory. Other than address conversion and fetching, it will schedule and optimize operations. However, if the shared data is of a smaller size, this approach would be inappropriate. In this case, the requesting site will be able to locate shared remote data after communicating with the memory controller.

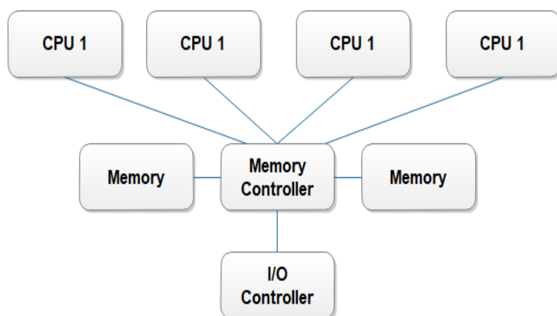

Fig 2. Memory controller Architecture.

The shared zone represented in Fig. 3 is an illusion created by the memory controller, which is connected to each site and so appears to be shared. When a process requests access to external data, a request will be made to the memory controller by the process requesting access to external data. The data is organised using the shared variable as a guide. Sequential consistency does an excellent job of handling the coherence semantics. Additional effort was required for commercial application in order to make the system the most successful possible from a variety of perspectives. The concepts and languages of object-oriented programming can also be utilised to develop graphical user interfaces (GUIs). Meanwhile, the message-passing structure is more complicated to construct but relatively easy to expand. Both the shared memory structure and the message transmission must develop in lockstep with each other in order to function properly. A realistic solution for parallel system architecture can be found in distributed shared memory, but only after careful analysis of the conditions and their justification.

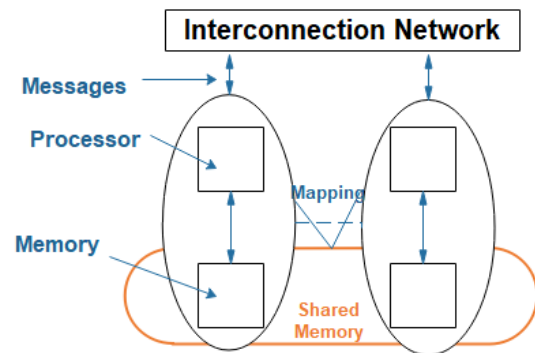

Fig. 3 Memory mapping Architecture using shared memory.

2. Proposed Memory Hierarchy and Methodology

When it comes to high-performance, large-scale multiprocessor systems, optimizing the performance of distributed shared memory is critical. In the past, several constraints were solved; however, the core problem continues to exist: Below, we go into further detail on the DSM method, locking shared space, thrashing, concurrent access, page faults, extension, transparency, huge database support, and cost. The objective of this research project was to create a novel distributed shared memory architecture that uses software parameters to significantly outperform existing structural designs while handling distributed shared address spaces [14]. Additionally, as we move closer to the deployment of software distributed shared memory, a slew of considerations will come into play. Implementing a distributed shared memory system is achievable through the use of an expanded virtual address space architecture in conjunction with the application of a new operating system paradigm. Because of the use of replication and granularity selection for data transfer, this unique DSM technique (figure 4) improves the overall efficiency of the system by employing a sequential consistency mechanism. Each system node is equipped with a mapping manager, which acts as a bridge between the node's local memory and the shared virtual memory space on the network. Write-up protocols are used to keep track of what each node is looking up and performing on a given time frame.

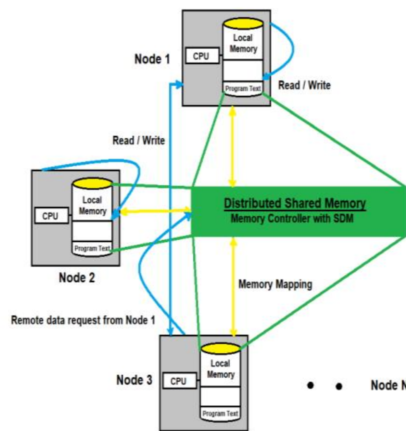

Fig 4. Architecture of distributed multi core architecture.

Configuration data must be downloaded from off-chip memory in order to map jobs onto the FPGA, as shown in Figure 5. A certain amount of energy is used and a delay is generated for each configuration data fetch from off-chip memory. To alleviate these overheads, a memory hierarchy consisting of two memories is used. One memory is optimised for High Speed (HS), while the other is optimised for Low Energy (LE). In comparison, HS memory has a shorter time delay and uses less energy than LE memory [15] –[18]. As a result, the configuration data is mapped in the suggested memory hierarchy using a mapping technique that decreases energy usage while still satisfying deadlines.

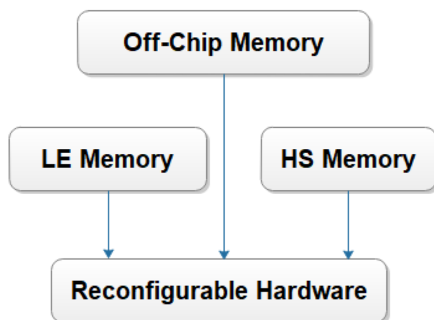

Fig 5. Memory hierarchy for mapping the configuration data.

Simulates memory operations, calculates access latency, and returns the result to the sim-outorder callback function. An example of the memory_ access latency function signature is as follows:

unsigned int memory_accessJatency(struct mem_t· mem, int chunks, md_addr_t addr, enum mem_cmd cmd, tick_t now);

This option has a mem t pointer that points to the sim-outorder simulation program's mem t data structure. The second input, chunks, defines the total number of bytes sought by the processor/cache when it requests memory access (in sim-outorder, it is the L2 cache block size since the L2 cache is the only cache that requests for memory access). The address that the processor/cache is looking for as the initial address is specified by the addr option. The processor cycle that happened when the processor/cache requested the memory access is the final parameter.

However, the latency number returned by memory access latency is a latency value relative to the 'now' argument, not the sum of the timing parameters (Tcmd, Trp, Trd, Teas, Twd, and data transmission).

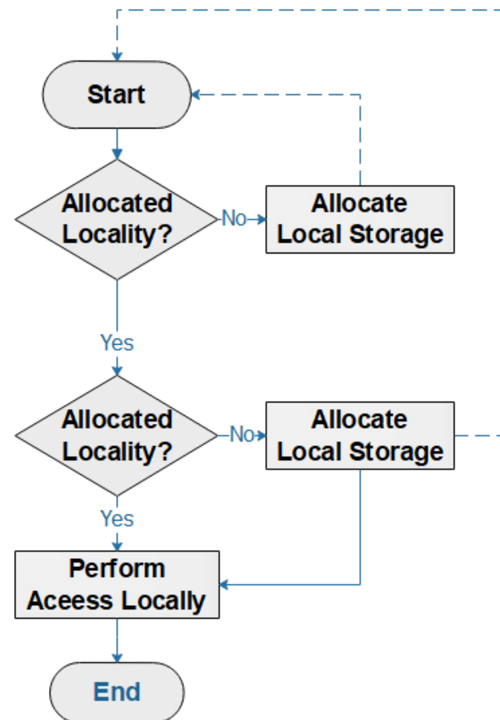

Fig 6. Flow of Memory Access.

The function's return result can be a significant amount higher than the sum of the individual timings. Because the memory model may be idle or busy servicing the previous access when the processor/cache requests a memory access, this occurs. If the memory is processing another access, it must wait for the prior access to complete before servicing the current access. The memory access latency function (delay function) determines the earliest possible processor cycle (start time) for the memory access requested by the processor/cache. The latency function's primary goal is to achieve this. To determine if the access is a random type or a fast page type, the latency function will first call the get mem real bank, get mem bank, get mem channel, and is latched functions. The latency function will return a value after these functions have been called. The latency function will detect if the access is the initial memory access of the simulation (bus timer = 0) once the access type has been determined. If this is the case, the latency function will be disabled. If this is the case, the start time can simply be set to "now," and the access type can only be random (the initial access in a simulation must be random access). The latched row/page of all memory banks is set to row 0 when the mem t data structure is initialised, which means we must override

the result of the is latch function). As a result, if this isn't the first time the simulation has been accessed, latency is determined by comparing the value of the "now" parameter to the previous command start time (cmd timestamp + Tcmd) for the channel, with the greater value being used as the temporary start time. This is owing to the fact that no access can start before the previous one. The temporary start time will be increased as the function deals with more scenarios (i.e., moved further away from the current time).

An access control method verifies whether local memory holds a valid copy of the data after it is allocated. If this is not the case, the mechanism uses an access policy to obtain up-to-date information or additional access permissions. The validity of a copy may be determined by the type of access; for example, duplicated copies are valid for reading but not for writing in most protocols [19] –[21]. Many researchers have explored the ways to restructure The Legacy C++ Program [22] –[26]. The access policy includes both the coherence protocol and the global allocation policy, which specify which node a request is forwarded to and how it is processed. Memory access policy for shared data objects residing locally and remotely is shown in Figure 6. This policy employs both messaging and access constraints. The policy may send signals telling other sites to invalidate their copies before permitting the local node to write its copy.

3. Results and Discussion

The RTL schematic for the cache memory compression block is shown in Figure 7. It is a design abstraction that represents the circuit in terms of digital signals flowing between hardware registers and is used in the design of integrated circuits.

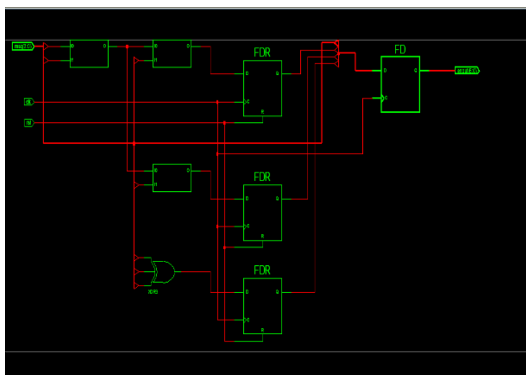

Fig. 7 RTL schematic memory block.

The compression algorithms RTL schematic is displayed in Figure 8. Figure 9 summarizes the device utilization for the implemented design. After the synthesis, this is demonstrated. The number of devices used during the design and implementation phases can be calculated using this

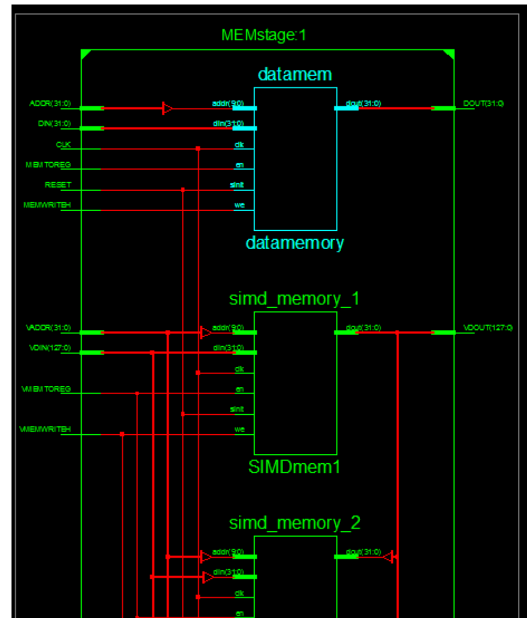

Fig. 8 RTL Schematic of the distributed memory using compression algorithm.

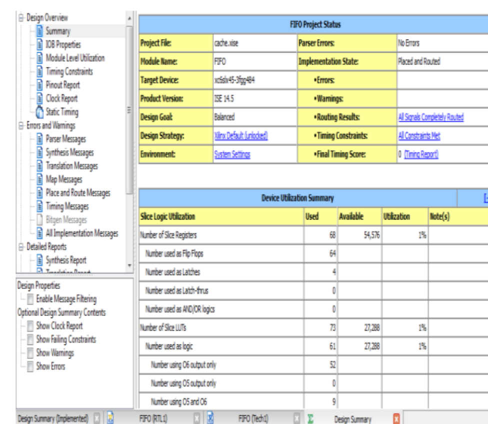

Fig. 9 Device Utilization Summary.

The suggested shared distributed memory (SDM) algorithm's power analysis is shown Table 1. This analysis displays overall power and junction temperature. The Figure 10 gives timing summary for the distributed memory. The proposed memory device consumes very less on-chip power consumption of 0.0082 W which clearly signifies the optimization of memory to a great extent. Furthermore, junction temperature of the distributed shared memory is nearly room temperature (25.4 ^{0}C), which clearly depicts using SDM the current processor operating properly with out any flaw and even not dissipating more heat. Henceforth the proposed distributed memory is very efficient for the next generation high speed computing.

Table 1 Power consumption and junction temperature of the Proposed Shared Distributed Memory.

On-chip total Power	0.0082W
Junction Temperature	25.4 ^{0}C

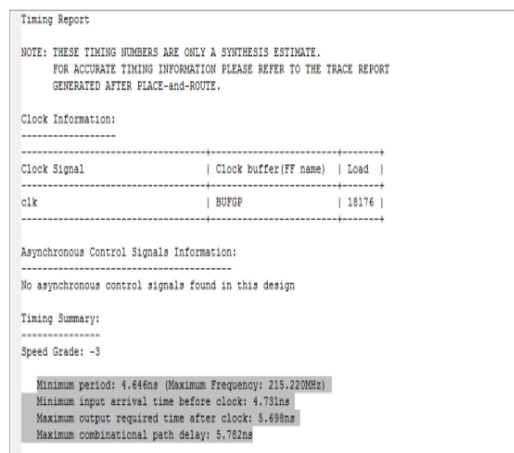

Fig. 10 Timing summary.

4. Conclusion

A Scientists are now able to use high-performance computers as a viable alternative to the more out-of-date experiential and theoretical approaches. While attempting to explain the language used to create distributed memory systems, this part is included. Single-site networks are used for this memory controller. IP address of the memory controller node was utilised to establish a connection between the client machine and the memory controller node. A message on the output terminal of the memory controller verifies the connection. If all of the client processes are running on the same system, the memory controller can be connected to the local host. As a large-scale DSM system, future efforts can be prepared to improve algorithm strategies in order to expand the number of nodes. All will be able to access shared data items at the same time. To improve distributed shared memory management, this work can be improved to optimise memory controller tasks and design difficulties. It has the ability to handle an increasing number of processors at once, higher execution speed, and security.

In order to decrease the energy used by the system for the refresh operations themselves as well as the energy used during the refresh activities themselves, further scheduling techniques can be used. Also, proposed distributed memory can be useful to restructure the input legacy C++ programming system.

Acknowledgment

Author thanks Dept. of Computer Science and Engineering, Sasi Institute of Technology and Engineering, Tadepalligudem for giving us opportunity to work under High Performance computing lab.

5. References and Footnotes

Ethics approval

This article does not contain any studies with human or animal subjects.

Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability Statement

The data cannot be made publicly available upon publication because no suitable repository exists for hosting data in this field of study. The data that support the findings of this study are available upon reasonable request from the authors.

Funding No funding received to carry out this research work.

References

[1] Ahmed, M.R., Zheng, H., Mukherjee, P., Ketkar, M.C. and Yang, J., 2021, April. Mining message flows from system-on-chip execution traces. In 2021 22nd international symposium on quality electronic design (ISQED) (pp. 374-380). IEEE.

[2] Shan, L. and Sun, H., 2021, May. Distributed collaborative simulation middleware based on reflective memory network. In 2021 IEEE 24th international conference on computer supported cooperative work in design (CSCWD) (pp. 274-279). IEEE.

[3] Cao, Y., Mukherjee, P., Ketkar, M., Yang, J. and Zheng, H., 2020, March. Mining message flows using recurrent neural networks for system-on-chip designs. In 2020 21st International Symposium on Quality Electronic Design (ISQED) (pp. 389-394). IEEE.

[4] Adetomi, A., Enemali, G. and Arslan, T., 2017, September. Towards an efficient intellectual property protection in dynamically reconfigurable FPGAs. In 2017 Seventh International Conference on Emerging Security Technologies (EST) (pp. 150-156). IEEE.

[5] Wen, H. and Zhang, W., 2021. Cache Leakage Reduction Techniques for Hybrid SPM-Cache Architectures. Journal of Circuits, Systems and Computers, 30(01), p.2150008.

[6] Kwon, S.J., Kim, S.H., Kim, H.J. and Kim, J.S., 2017, January. LZ4m: A fast compression algorithm for in-memory data. In 2017 IEEE International Conference on Consumer Electronics (ICCE) (pp. 420-423). IEEE.

[7] Guo, Y., Hua, Y. and Zuo, P., 2018, March. DFPC: A dynamic frequent pattern compression scheme in NVM-based main memory. In 2018 Design,

Automation & Test in Europe Conference & Exhibition (DATE) (pp. 1622-1627). IEEE.
[8] Choukse, E., Erez, M. and Alameldeen, A.R., 2018, October. Compresso: Pragmatic main memory compression. In 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (pp. 546-558). IEEE.
[9] Lahiry, A. and Kaeli, D., 2017, November. Dual dictionary compression for the last level cache. In 2017 IEEE International Conference on Computer Design (ICCD) (pp. 353-360). IEEE.
[10] Rumyantsev, A., Krupkina, T., Losev, V. and Maksimov, A., 2020, January. Development of a Measurement System-on-Chip and Simulation on FPGA. In 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus) (pp. 1851-1854). IEEE.
[11] Frolova, P.I., Chochaev, R.Z., Ivanova, G.A. and Gavrilov, S.V., 2020, January. Delay matrix based timing-driven placement for reconfigurable systems-on-chip. In 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus) (pp. 1799-1803). IEEE.
[12] Jain, A., Soner, S. and Holkar, A., 2010, October. “Reverse engineering”: Extracting information from C++ code. In 2010 2nd International Conference on Software Technology and Engineering (Vol. 1, pp. V1-154). IEEE.
[13] Strobel, M., Radetzki, M.: Design-time memory subsystem optimization for lowpower multi-core embedded systems. In: 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 347–353 (2019). DOI 10.1109/MCSoC.2019.00056.
[14] H. Fan et al., "High-Precision Adaptive Slope Compensation Circuit for System-on-Chip Power Management," 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), 2019, pp. 1-2.
[15] N. Arun Vignesh, Ravi Kumar, R. Rajarajan, S. Kanithan, E. Sathish Kumar, Asisa Kumar Panigrahy, and Selvakumar Periyasamy, “Silicon Wearable Body Area Antenna for Speech-Enhanced IoT and Nanomedical Applications”, Journal of Nanomaterials, vol. 2022 , Article ID 2842861, 9 pages, 2022. https://doi.org/10.1155/2022/2842861
[16] Devi, M.P., Ravanan, V., Kanithan, S., N.A.Vignesh, “Performance Evaluation of FinFET Device Under Nanometer Regime for Ultra-low Power Applications”, Silicon (2022). https://doi.org/10.1007/s12633-022-01772-x
[17] D. Chen, J. Edstrom, Y. Gong, P. Gao, L. Yang, M. McCourt, J. Wang and N. Gong, "Viewer-Aware Intelligent Efficient Mobile Video Embedded Memory", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 4, pp. 684-696, 2018.
[18] Calinescu, G., Fu, C., Li, M., Wang, K. and Xue, C.J., 2018. Energy optimal task scheduling with normally-off local memory and sleep-aware shared memory with access conflict. IEEE Transactions on Computers, 67(8), pp.1121-1135.
[19] Arun Jayakar, S., Rajesh, T., Vignesh, N.A. and Kanithan, S., 2022. Performance Analysis of Doping Less Nanotube Tunnel Field Effect Transistor for High Speed Applications. Silicon, 14(12), pp.7297-7304.
[20] Z. Zhao, Y. Sheng, M. Zhu and J. Wang, "A Memory-Efficient Approach to the Scalability of Recommender System With Hit Improvement", IEEE Access, vol. 6, pp. 67070-67081, 2018.
[21] Deepa, R., Devi, M.P., Vignesh, N.A. and Kanithan, S., 2022. Implementation and performance evaluation of ferroelectric negative capacitance FET. Silicon, 14(5), pp.2409-2419.
[22] Handigund, S.M. and Kulkarni, R.N., 2010. An Ameliorated Methodology for the design of Object Structures from legacy ‘C’Program. International Journal of Computer Applications, 975, p.8887.
[23] Dr. Shivanand M. Handigund, Dr. Rajkumar N. Kulkarni, “An Ameliorated Methodology for the Abstraction and Minimization of Functional Dependencies of legacy ‘C’ Program Elements “. International Journal of Computer Applications (0975 – 8887) Volume 16– No.3, February 2011.
[24] Dr. R.N. Kulkarni, Venkata Sandeep Edara, "A Novel Approach to Restructure The Legacy C++ Program", Journal of Huazhong University of Science and Technology, Volume: 50 Issue: 05 , 2021, ISSN: 1671-4512.
[25] Dr. R.N. Kulkarni, P.Pani Rama Prasad, "Abstraction of UML Class Diagram from the Input Java Program", Int. J. Advanced Networking and Applications, Volume: 12 Issue: 04 Pages: 4644-4649(2021) ISSN: 0975-0290.
[26] R.N.Kulkarni and S.Shenaz Begum, "Abstraction of 'C' Program from Algorithm", Springer Nature Singapore Pte Ltd. 2021, Research in Intelligent and Computing in Engineering, Advances in Intelligent Systems and Computing 1254.
[27] Nair, K. S. S. . (2023). Rapidly Convergent Series from Positive Term Series. International Journal on Recent and Innovation Trends in Computing and Communication, 11(3), 79–86. https://doi.org/10.17762/ijritcc.v11i3.6204
[28] Ana Silva, Deep Learning Approaches for Computer Vision in Autonomous Vehicles , Machine Learning Applications Conference Proceedings, Vol 1 2021.