FOREX Publication
Open Access | Rapid and quality publishing

International Journal of
Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 387-393 | e-ISSN: 2347-470X

A Classy Memory Management System (CyM2S) using an Isolated Dynamic Two-Level Memory Allocation (ID2LMA) Algorithm for the Real Time Embedded Systems

K. Siva Sundari[1], R. Narmadha[2] and S. Ramani[3]

[1]*Research Scholar, Sathyabama Institute of Science and Technology, Chennai, Tamil Nadu 600119*
[2]*Professor, Sathyabama Institute of Science and Technology, Chennai, Tamil Nadu-600119, narmadha1109@gmail.com*
[3]*Associate Professor, Department of Electronics and Communication Engineering, Sreenidhi Institute of Science & Technology, Hyderabad-501301, Telangana, ramanis@sreenidhi.edu.in*

***Corresponding Author:** K. Siva Sundari; E-mail: sivasundari2029@gmail.com

ABSTRACT- Due to an increased scalability, flexibility, and reduced cost complexity, the dynamic memory allocation models are highly preferred for the real-time embedded systems. For this purpose, the different types of dynamic models have been developed in the conventional works, which are highly focused on allocating the memory blocks with increased searching capability. However, it faced some of the problems and issues related to the factors of complex operations, high time consumption, memory overhead, and reduced speed of processing. Thus, this research work objects to design an advanced and intelligent dynamic memory allocation mechanism for the real-time embedded systems. Here, a Classy Memory Management System (CyM^{2}S) is developed by using an Isolated Dynamic Two-Level Memory Allocation (ID^{2}LMA) algorithm for efficiently allocating the memory blocks with simple searching. The CyM^{2}S helps to reduce the fragmentation rate and time consumption by optimally allocating the memory blocks. In this model, the small buffer has been maintained for surplus pointers, and the allocated blocks comprise the metadata and payload data. During evaluation, the performance of the proposed CyM^{2}S- ID^{2}LMA technique is validated and compared by using the measures of memory allocation time, release time, execution, and processing speed.

Keywords: Dynamic Memory Management, Classy Memory Management System (CyM^{2}S), Isolated Dynamic Two-Level Memory Allocation (ID^{2}LMA), Embedded Systems, Memory Fragmentation, and Optimal Memory Allocation.

ARTICLE INFORMATION
Author(s): K. Siva Sundari, R. Narmadha and S. Ramani

Special Issue Editor: Dr. S. Gopalakrishnan;
Received: 10/05/2022; Accepted: 20/06/2022; Published: 30/06/2022;
E- ISSN: 2347-470X;
Paper Id: 0422SI-IJEER-2022-28;
Citation: 10.37391/IJEER.100254
Webpage-link:
https://ijeer.forexjournal.co.in/archive/volume-10/ijeer-100254.html

This article belongs to the Special Issue on **Intervention of Electrical, Electronics & Communication Engineering in Sustainable Development**

1. INTRODUCTION

In recent days, the modern embedded systems [1, 2] have been increasingly used in different types of application systems like mobile, networking, multimedia, and etc due to its efficient features and characteristics. Typically, embedded systems [3, 4] are considered as the kind of computers mainly designed for performing the specialized functions. Among the other domains, it plays a vital role in the digital vision technology [5], and its general architecture model is shown in Fig 1. The modern embedded systems are developed based on the combination of software and hardware components, which performs certain functions according to the requirements. In embedded systems, memory management is one of the most crucial and essential task need to be addressed, because which are mainly used to ensure the better system performance. Generally, the memory management techniques [6, 7] are classified into the following types. Static memory management and Dynamic memory management.

The static memory management [8] solutions are considered as the worst-case solutions for the embedded systems, because it requires to store the large amount of data in the storage medium. Hence, it requires more memory for processing the application at the initial stage [9], and for every time, the data is required to be stored. In the software applications [10, 11], the memory blocks could not be de-allocated during the execution time, so the data requests are not processed on the same embedded systems. Moreover, the static memory allocation models [12] cannot predict the required amount of memory for the real time applications, which results in an increased memory overhead.

Due to these facts, the dynamic memory management techniques [13] are highly preferred for the modern embedded systems, since it efficiently allocates the memory blocks for execution. The main reason of using the dynamic memory allocation models is, it ensures the flexibility of memory acquisition during the run time. The different types of dynamic memory allocation models [14, 15] developed in the conventional works are as follows: Sequential fit, Buddy allocator, Indexed fit, Bitmapped fit, Doug Lead (DLmalloc),Half-fit, Two-Level Segregated Fit (TLSF),TCmalloc. These techniques are mainly developed for dynamically allocating the memory spaces [16, 17] for

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 387-393 | e-ISSN: 2347-470X

executing the applications with an efficient memory and resource utilization. However, it limits the major issues [18, 19] of increased processing time, execution time, complexity in bitmapping operations, overhead, and memory fragmentation. Hence, the proposed work objects to develop an intelligent dynamic memory allocation model for the real-time embedded systems.

The other portions of this paper are segregated into the followings: *Section 2* reviews the conventional dynamic memory allocation strategies used for the embedded systems, which also discusses about the pros and cons of each mechanism according to its characteristics. Then, *Section* 3 provides the detailed explanation about the proposed CyM^{2}S dynamic memory allocation model with its working model and illustration. *Section 4* presents the simulation and comparative analysis of both existing and proposed techniques based on various performance measures. Finally, the overall paper is summarized with its obtainments and future scope in *Section 5*.

2. RELATED WORKS

This section investigates the conventional dynamic memory management mechanisms for optimally allocating the tasks in embedded systems. Also, it examines the advantages and disadvantages of each technique according to its operational characteristics and functions. Ouyang, et al [20] implemented a wait free dynamic memory management scheme for enhancing the performance of embedded systems. From this study, it was analyzed that wait free memory allocation mechanism provides the better results for the real time embedded systems. Kloda, et al [21] implemented a deterministic hierarchy based memory management mechanism for a high-end embedded systems. The proposed technique incorporates the functionalities of Invalidation Driven Allocation (IDA) mechanism. Here, the memory mapping and virtualization processes have been performed for optimizing the memory with minimal processing time. In addition to that, the coloring-based partitioning was performed to segregate the resources, and to obtain the temporal isolation. However, this framework limits with the major problems of increased overhead, and computational complexity, which degrades the performance of entire system. David, et al [22] suggested a tiny machine learning technique for performing an efficient dynamic memory management in embedded systems. This work developed an interpreter-based approach for reducing the hardware requirements and avoiding other external dependencies. The tensor flow model was mainly used to increase the robustness and scalability of network. During memory optimization, the meta-data could be stored in the persistent memory to run the model with the run time tensors. Zhou, et al [5] deployed a deep learning networks model for task allocation, and memory management in embedded systems. This paper mainly objects to ensure the security, and reduce the complexity of embedded systems by using a dynamic integrity measurement algorithm. In addition to that, the Rate Monotonic Scheduling (RMS) mechanism was developed to schedule the tasks according to its priority and memory usage. Rodriguez, et al [23] introduced a new framework, named as, ARTICo^3 for increasing the performance of embedded system by optimizing the utilization of memory. The key characteristics of this framework were hardware based, supports task and data level parallelism, high performance embedding, and dynamic adaptability. Moreover, an automated tool chain methodology was employed to handle the computing resources with better memory management. However, it has the limitations of increased execution time, complex processing, and ineffective solution space. Wang, et al [24] implemented a training Deep Neural Network (DNN) algorithm for dynamic GPU memory management. This paper objects to apply the dynamic strategy for efficiently optimizing the available workspace with cost consumption. The key benefits of this work were maximized system performance, reduced training time, and complexity. Almatary, et al. developed a lightweight compartmentalization-based architecture for ensuring the scalability, compatibility, and availability of embedded systems. This architecture includes the components of Memory Protection Unit (MPU), and Memory Management Unit (MMU) for optimizing the memory usage. Alaswad, et al. suggested a direct memory management mechanism for increasing the performance of low power embedded micro-controllers. This paper intends to develop the next generation embedded technology for the current industrial relevant application systems. Armijos, et al. object to monitor the memory usage of FPGA systems, where the memory was optimized during the task scheduling and execution. After executing all the buffers, the memory plan could be re-executed for recycling the non-persistent buffers, which increases the lifetime of embedded systems.

According to this review, it is analyzed that the existing dynamic memory management methodologies are mainly developed for optimizing the memory usage of embedded systems. But, it faced the challenges related to the following factors: Increased storage overhead, High time consumption for processing, Difficult to handle the bags, Wastage of memory, Complexity in memory optimization. Hence, the proposed work objects to implement an advanced optimization mechanism for optimally allocating the memory space for an embedding operation.

3. PROPOSED METHODOLOGY

This section presents the detailed description about the proposed dynamic memory allocation algorithm used for optimizing the memory usage of embedded systems. The main contribution of this work is to satisfy the high level memory requirements of embedded systems with minimal time consumption, and optimized fragmentation rate. For this purpose, a Classy Memory Management System (CyM^{2}S) is developed in this paper, which incorporates the following procedures for an efficient memory handling: Isolated Dynamic Two-Level Memory Allocation (ID2LMA) algorithm and Programmable Large Free Memory Slots Creation. The proposed CyM^{2}S is mainly used to enhance the searching rate of available free memory blocks with increased speed of processing. This is a kind of dynamic memory allocation model that highly improves the efficiency of

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 387-393 | e-ISSN: 2347-470X

memory allocation, and reduces the fragmentation rate. Typically, the embedded systems are more space constrained and small in nature, hence it is highly essential to optimize the memory utilization for better performance of the system. In this framework, the memory usage has been maximized by separately defragmenting the blocks, which includes the operations of memory allocation, and move of defragmented blocks. The memory organization model of the proposed CyM^{2}S is shown in Fig 1.

In this model, the memory is structured into data blocks and points, where multiple points can be used to construct the complex data structures. Then, the small buffer has been maintained for surplus pointers, and the allocated blocks comprises the metadata and payload data. Here, the optimization of blocks help to increase the system performance with reduced time consumption. When compared to the conventional techniques, the proposed CyM^{2}S has an increased ability to handle the complex system functions, hence it is suitable for the embedded application systems.

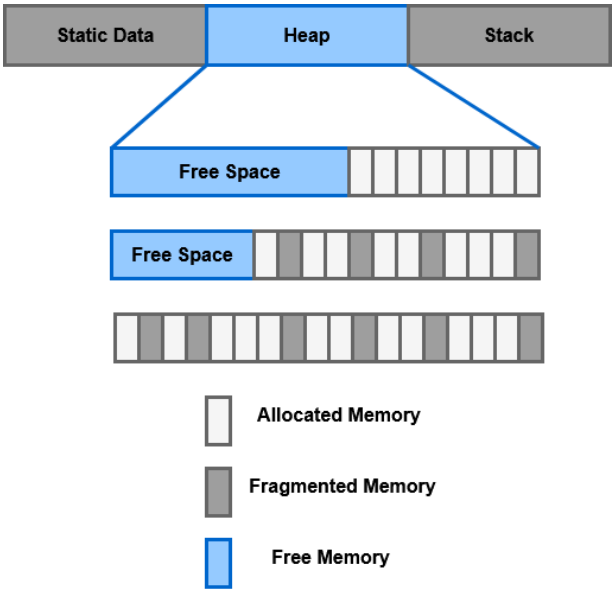

Figure 1: Typical memory allocation scheme

The main API functions used in this model are as follows: memory allocation, memory de-allocation, and pointer operations.

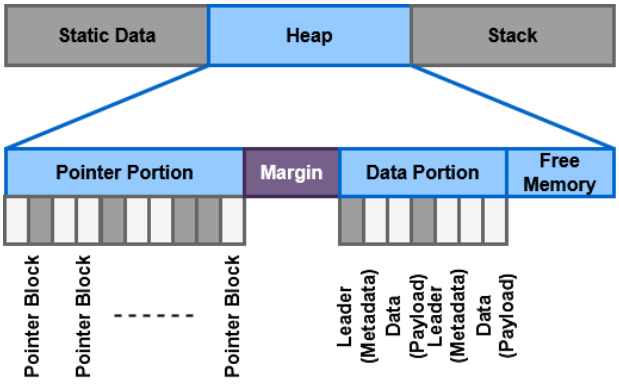

Figure 2: Memory organization of CyM^{2}S

The ID^{2}LMA algorithm is mainly used to quickly allocate the available memory blocks based on the best-fit principle. The key benefits of using this approach are as follows:

- It efficiently minimizes the time complexity of operations.
- It enables the simple searching process.
- Dynamic memory allocation with increased efficiency.

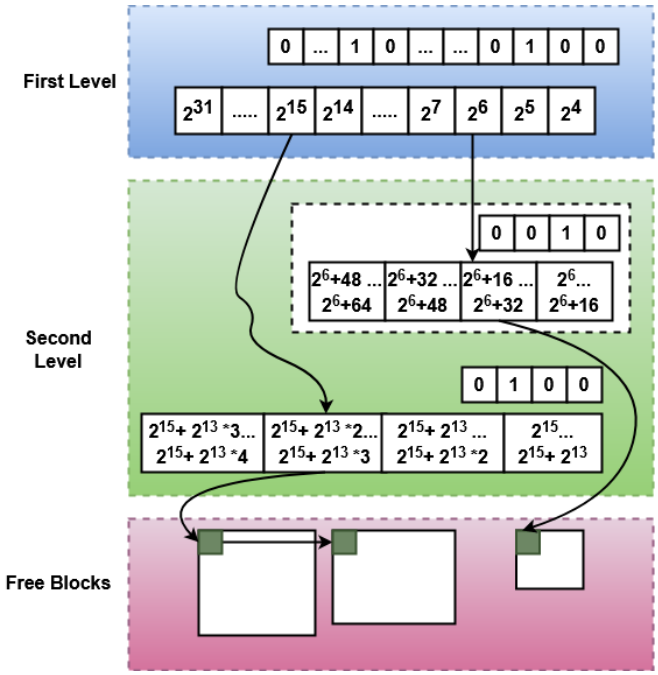

Figure 3: Data structure model of ID^{2}LMA

In this model, the two-level segregated list array has been maintained to link the available free memory blocks with varying sizes. During the first level, the free memory blocks are split into the order of 2n, where n = 4, 5 … 31. According to the specifications of users, the second level can segment the memory blocks of first level. During this process, the related bitmap is constructed for each list of array, where the segregated list comprises the free memory blocks. If the free blocks are available, the bitmap is represented as 1; otherwise, it can be represented as 0. The data structure model of ID^{2}LMA algorithm is shown in *Fig 4*.

The main factor of using this ID^{2}LMA technique is, it identifies the free memory blocks with minimal time consumption by segregating the list index as shown in below:

$$SI_L1 = \lfloor(\log_2(s)\rfloor \quad (1)$$

$$SI_L2 = (s - 2^{SI_L1})\frac{2^{No_{SL}}}{2^{SI_L1}} \quad (2)$$

Where, SI_L1 and SI_L2 are the first and second segregated list index respectively, s indicates the size, and No_{SL} is the number of intervals in the second segregated list. If there is any available free block in the first list indexSI_L1, it can be identified based on the value of SI_L2 and is allocated to the application. If there is no available free blocks, the position of index is updated and the searching is enabled at the position of $SI_L1 + 1$. If it comprises the available free blocks, the

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 387-393 | e-ISSN: 2347-470X

corresponding block can be allocated; otherwise, the searching can be continued until identifying the available free blocks. The ID^{2}LMA can easily locate the available free memory blocks with minimal time consumption, which is the key benefit of using this technique. Here, a memory access statistic table has maintained for task allocation, which includes the attributes of size of memory, number of applications, releases, interval of reentering, and operational memory time as shown in *Fig 5*.

Figure 4: Memory accessing table

In this data structure memory model, the bitmap value is considered as consistent, where the value of bits are represented according to the type of memory block and class size. During the system execution, the proposed model frequently allocates and releases the memory blocks, and the memory blocks that are executed for a long period are identified with its block information. Based on this information, the memory blocks having different size are updated for allocation. Here, the memory type determination is also performed for simplifying the process of memory allocation, and its working model is illustrated in *Fig 6*.

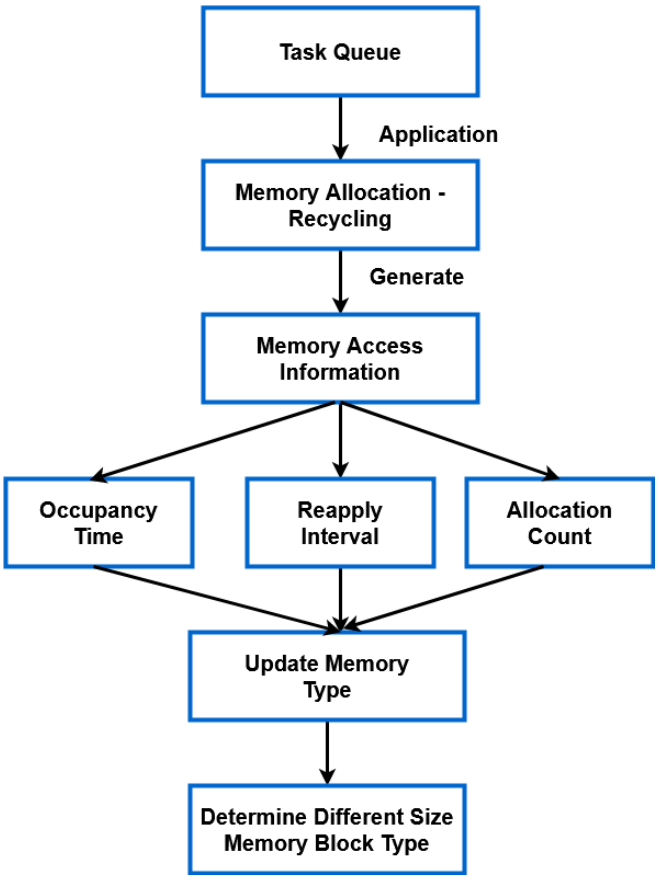

Figure 5: Flow of memory determination process

The proposed memory management operations could satisfy the following constraints:

1. The memory blocks having the status as occupied are not reallocated again, which is illustrated in below:

$$\forall MB, S \cdot Status\ (MB) = 1 \quad (3)$$

$$alloc\ (S) \neq MB \quad (4)$$

2. The available and free memory blocks are not released, and is represented as follows:

$$\forall MB, S \cdot Status\ (MB) = 0 \quad (5)$$

$$free\ (MB) \neq False \quad (6)$$

3. The adjacent two memory blocks are not overlapped, which is represented in below:

$$\forall MB, \left[MB, MB + size(MB)\right) \cap \left[r(MB), r(MB) + size\left(r(MB)\right)\right] = \emptyset \quad (7)$$

4. Here, minimum one non-free block can exist between two free blocks as represented below:

$$\forall MB, F \cdot Status(MB) = Status\ (F) = 0 \wedge MB < F \quad (8)$$

$$r(MB) < F \wedge Status\left(r(MB)\right) = 1 \quad (9)$$

5. The free blocks are not adjacent, which is expressed as follows:

$$\forall MB \cdot Status\ (MB) = 0 \quad (10)$$

$$Status\ \left(r(MB)\right) = 1 \wedge Status\left(left(MB)\right) = 1 \quad (11)$$

6. The free memory spaces can be allocated as shown in below:

$$\forall MB \cdot \exists S. Status\ (MB) = 0 \quad (12)$$

$$alloc\ (S) = MB \quad (13)$$

7. The allocated memory space is released after execution, which is represented as follows:

$$\forall MB. Status\ (MB) = 1 \quad (14)$$

$$free\ (MB) = true \quad (15)$$

During initialization, the searching table, dynamic memory area, and memory pool have been initialized. Then, the pool is treated as the large memory block, where the relevant attributes are initialized with its index structure. After initialization, the memory allocation has been performed for allocating the memory space according to the blocks of users. In this process, the index structure of memory is updated for further operations, which helps to minimize the execution time of processing. An efficiently memory allocation performed in this system is illustrated in *Fig 7*, where N indicates the number of times the memory block is updated, P indicates the count of memory block is released, K is the interval rate, and D denotes the duration of memory occupation. Then, the parameters N', P', K', and D' are the thresholds of the corresponding attributes.

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 387-393 | e-ISSN: 2347-470X

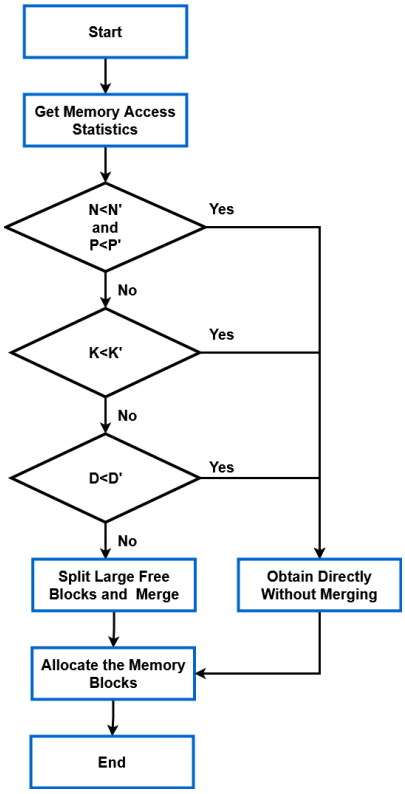

Figure 6: Memory allocation

Consequently, the available memory blocks can be searched from the next list of free block list. During the implementation process, the parameter of the largest memory block has been inserted in the index structure. The memory management system of the ID^{2}LMA identifies the appropriate memory blocks based on the updated memory size of task and index structure. After that, the memory block MB is identified and removed from the index structure S for updating the complete indexing structure. The memory split, memory free, memory insert, and merge operations have been performed for enabling the proper utilization of memory in the embedded systems. The primary advantages of the proposed CyM^{2}S dynamic memory allocation model are as follows: simple to implement, optimized memory utilization rate, minimal complexity, speed of processing, and reduced time consumption.

4. RESULTS AND DISCUSSION

This section validates the results of proposed CyM^{2}S dynamic memory allocation model in embedded systems by using various evaluation metrics. It includes the parameters of as memory allocation time, release time, execution time, and processing speed. For evaluating the performance of the proposed system, the four different real time embedded applications have been considered in this analysis. It includes 3D Path Planning (3DPP), Histogram of Oriented Gradients (HoG), Space Time Adaptive Processing (STAP), and Infrared H2RG Detector (ESA). Fig 7 and Table 2 shows the speedup performance of the proposed CyM^{2}S dynamic memory allocation model for the four different real time embedded applications. Here, the speedup is estimated with respect to the pre-allocated task data structures. From the results, it is analyzed that the proposed CyM^{2}S model could efficiently increase the speedup performance for all kinds of applications.

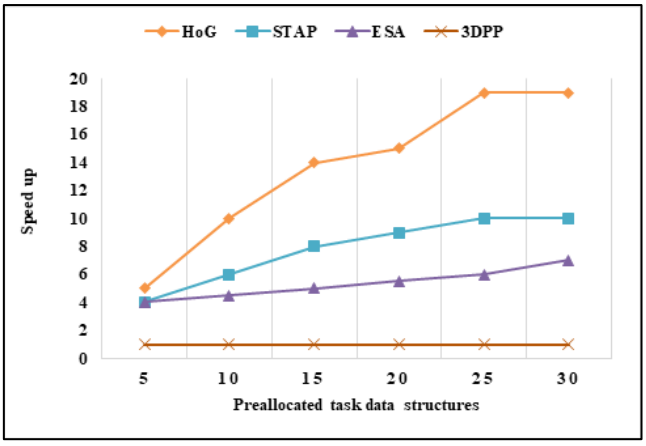

Figure 7: Pre-allocated tasks Vs speedup

Figure 8 evaluates the memory allocation time of conventional TLSF, bitmapped TLSF, and proposed CyM^{2}S- ID^{2}LMA techniques with respect to varying memory block size. Typically, the memory allocation time is mainly estimated to validate the efficiency of allocation methodologies, which is calculated according to the average time spent for the memory allocation process. It is calculated as shown in below:

$$Memory_{Alotime} = \frac{\sum TT_i}{NM} \quad (16)$$

Where, TT_i is the time taken for allocating the memory blocks, and NM indicates the total number of allocated blocks of memory. From the obtained results, it is evaluated that the proposed CyM^{2}S- ID^{2}LMA outperforms the other approaches with reduced allocation time.

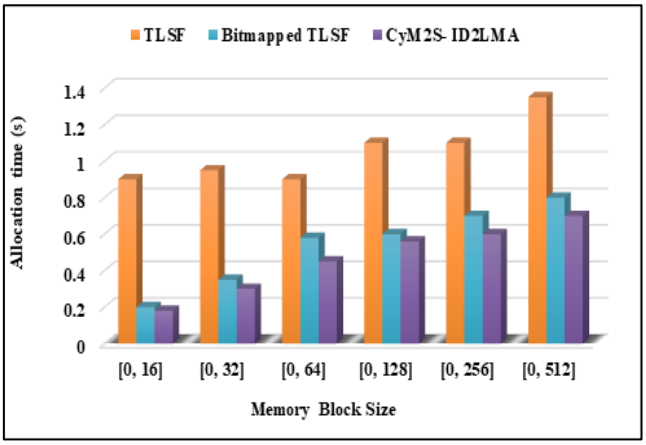

Figure 8: Allocation time Vs memory block size

Similarly, the memory release time of existing and proposed dynamic memory allocation models are validated and compared with respect to varying memory block size as shown in Figure 9. The memory release time is also considered as one of the most essential parameter used for validating the memory allocation efficiency and performance of the models.

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 387-393 | e-ISSN: 2347-470X

Moreover, it is estimated based on the amount of time required to insert into the memory or the time taken for merging the adjacent memory blocks during the release operation. It is calculated as follows:

$$Memory_{reltime} = \frac{\sum RT_i}{NM} \quad (17)$$

Where, RT_i indicates the release time of i^{th} block. Based on this evaluation, it is observed that the proposed CyM^{2}S-ID^{2}LMA technique overwhelms the other approaches with minimal release time for processing blocks of memory.

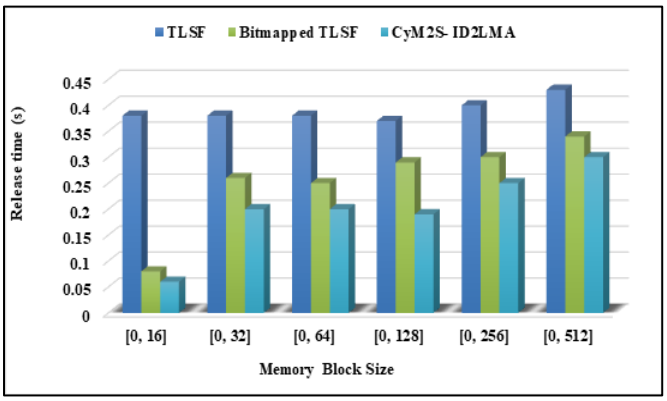

Figure 9. Average release time Vs memory block size

Figure 10 validates the execution time of standard dynamic memory allocation, and proposed CyM^{2}S- ID^{2}LMA models with respect to the number of requests. Normally, the performance of entire memory allocation system is determined based on the time of processing the requests. According to this analysis, it is observed that the proposed technique overcomes the standard model with reduced execution time for processing all requests.

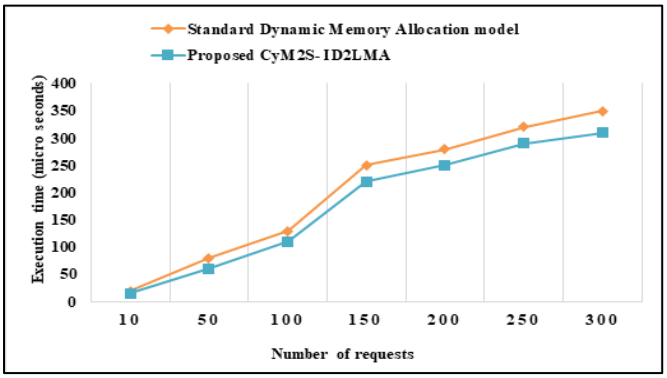

Figure 10: Execution time Vs Number of requests

5. CONCLUSION

This paper presents an intelligent and advanced dynamic memory allocation model, named as, CyM^{2}S- ID^{2}LMA for the real time embedded systems. The main purpose of this work is to to satisfy the high level memory requirements of embedded systems with minimal time consumption, and optimized fragmentation rate. Also, it objects to perform an efficient memory handling by incorporating an ID^{2}LMA algorithm with the programmable large free memory slots creation model. Here, the utilization of memory is maximized by separating the defragmented memory blocks, and it includes the operations of memory allocation, and move of defragmented blocks. In the proposed work, the main purpose of using the CyM^{2}S model is to handle the complex system functions with simple memory management operations. Moreover, the ID^{2}LMA algorithm is deployed for increase the speed of processing with minimal time complexity, simple searching, and ensured efficiency of memory handling. Also, it helps to increase the searching rate of available free memory blocks by using the best-fit principle. During evaluation, the performance of the proposed technique is validated and compared by using various evaluation measures such as memory allocation time, release time, execution time, and processing speed.

REFERENCES

[1] Z. Shen, K. Dharsee, and J. Criswell, "Fast Execute-Only Memory for Embedded Systems," in 2020 IEEE Secure Development (SecDev), 2020, pp. 7-14.

[2] S.-H. Park, J.-H. Lee, S.-W. Cho, and S.-H. Kim, "A Flash Memory Management Method for Enhancing the Recovery Performance," IEMEK Journal of Embedded Systems and Applications, vol. 13, pp. 235-243, 2018.

[3] M. Bazzaz, A. Hoseinghorban, and A. Ejlali, "Fast and predictable non-volatile data memory for real-time embedded systems," IEEE Transactions on Computers, vol. 70, pp. 359-371, 2020.

[4] M. Strobel and M. Radetzki, "Design-time memory subsystem optimization for low-power multi-core embedded systems," in 2019 IEEE 13th international symposium on embedded multicore/many-core systems-on-chip (MCSoC), 2019, pp. 347-353.

[5] J. Zhou, "Real-time task scheduling and network device security for complex embedded systems based on deep learning networks," Microprocessors and Microsystems, vol. 79, p. 103282, 2020.

[6] I. Georgiev and I. Georgiev, "Some Analysis of the Timing Parameters in Real-time Embedded Systems," in 2020 International Conference on Information Technologies (InfoTech), 2020, pp. 1-4.

[7] Y.-P. Liang, Y.-T. Fang, S.-H. Chen, Y.-T. Chen, T.-Y. Chen, W.-L. Wang, et al., "Brief Industry Paper: An Energy-Reduction On-Chip Memory Management for Intermittent Systems," in 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021, pp. 429-432.

[8] L. Papadopoulos, C. Marantos, G. Digkas, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, "Interrelations between software quality metrics, performance and energy consumption in embedded applications," in Proceedings of the 21st International Workshop on software and compilers for embedded systems, 2018, pp. 62-65.

[9] R. Wittig, M. Hasler, E. Matus, and G. Fettweis, "Queue based memory management unit for heterogeneous MPSoCs," in 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 1297-1300.

[10] V. Venkataramani, M. C. Chan, and T. Mitra, "Scratchpad-memory management for multi-threaded applications on many-core architectures," ACM Transactions on Embedded Computing Systems (TECS), vol. 18, pp. 1-28, 2019.

[11] A. A. Clements, N. S. Almakhdhub, S. Bagchi, and M. Payer, "{ACES}: Automatic compartments for embedded systems," in 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 65-82.

[12] T. Poggi, P. Onaindia, M. Azkarate-askatsua, K. Grüttner, M. Fakih, S. Peiró, et al., "A hypervisor architecture for low-power real-time embedded systems," in 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 252-259.

[13] S. Branco, A. G. Ferreira, and J. Cabral, "Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey," Electronics, vol. 8, p. 1289, 2019.

International Journal of
Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 387-393 | e-ISSN: 2347-470X

[14] M. Labbé and F. Michaud, "Long-term online multi-session graph-based SPLAM with memory management," Autonomous Robots, vol. 42, pp. 1133-1150, 2018.
[15] K. Maeng and B. Lucia, "Adaptive dynamic checkpointing for safe efficient intermittent computing," in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, pp. 129-144.
[16] R. Tabish, R. Mancuso, S. Wasly, R. Pellizzoni, and M. Caccamo, "A real-time scratchpad-centric OS with predictable inter/intra-core communication for multi-core embedded systems," Real-Time Systems, vol. 55, pp. 850-888, 2019.
[17] L. Shaofeng, Q. Lei, and Y. Mengfei, "Verification of a TLSF Algorithm in Embedded System," in Formal Methods and Software Engineering: 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1–3, 2021, Proceedings, 2020, p. 331.
[18] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V," in NDSS, 2019.
[19] R. Zeng, "Embedded Linux Operating System Network Accelerated Operation Method Based on ARM Processor," in 2021 Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS), 2021, pp. 315-319.
[20] X. Ouyang and Y. Zhu, "wfspan: wait-free dynamic memory management," ACM Transactions on Embedded Computing Systems (TECS), 2022.
[21] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, P. Valente, and M. Bertogna, "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2019, pp. 1-14.
[22] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, et al., "TensorFlow lite micro: Embedded machine learning for tinyml systems," Proceedings of Machine Learning and Systems, vol. 3, pp. 800-811, 2021.
[23] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. De la Torre, "Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework," Sensors, vol. 18, p. 1877, 2018.
[24] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, et al., "Superneurons: Dynamic GPU memory management for training deep neural networks," in Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming, 2018, pp. 41-53.
[25] S. Munaf, Dr. A. Bharathi, Dr. A. N. Jayanthi (2016), Double Pumping Low Power Technique for Coarse - Grained Reconfigurable Architecture. IJEER 4(1), 10-15. DOI: 10.37391/ijeer.040103. http://ijeer.forexjournal.co.in/archive/volume-4/ijeer-040103.php
[26] Victoria Satuluri, Ratna Babu Yellamati (2015), Design of Middleware and Software Embedded Development Kit For Area Based Distributed Mobile Cache System. IJEER 3(3), 44-49. DOI: 10.37391/ijeer.030301. http://ijeer.forexjournal.co.in/archive/volume-3/ijeer-030301.php