VOL.3 NO.1 March 2014 - ISSN 2233 – 1859

UOIuBIH
ORSinBIH
Operations Research Society in Bosnia and Herzegovina

Southeast Europe Journal of Soft Computing

Available online: www.scjournal.com.ba

IUS Soft Computing Research Group

Intelligent Memory Allocation based on Fuzzy Logic

Alma Husagic-Selman, Ali Aburas, Suvad Selman

International University of Sarajevo, Faculty of Engineering and Natural Sciences, Hrasnicka Cesta 15, Ilidža 71210 Sarajevo, Bosnia and Herzegovina

Article Info

Article history:
Article received on February 2014

Keywords:
Memory Allocation, Fuzzy Logic, Fuzzy Inference System.

Abstract

Based on the Computerized Parkinson’s Law “work expands so as to fill the time available for its completion” (Thimbleby, 1993) it can be deduced that regardless of the size of the memory, there will always be programs to completely fill, or even overload that memory. Thus intelligent/sensible memory allocation process is crucial to system’s performance. However, due to the constant increase of processing power and the growth and spread of distributed systems, such as grid and cloud computing, memory allocation becomes a great challenge in the area of memory management today. Making allocation intelligent, so that the memory fragmentation and response time are reduced would be great, and in this research, this was attempted. The research presents Fuzzy Allocator, memory allocator based on fuzzy inference system. The allocator manages to sort the incoming memory requests according to their size and the size of free memory slot (hole). The output of the fuzzy allocator is the order in which the allocation of memory will be performed on the incoming memory requests. It reorders the incoming memory request queue so that the response time is reduced, and fragmentation is minimized.

1. INTRODUCTION

CPU requests memory to store the running processes and memory allocator needs to respond to those requests immediately. The memory allocation is therefore real-time problem, and as usual, real-time problems are much more difficult to solve than the offline ones. Memory allocator must keep track of all parts of memory, those that are free and those that are in use. It must not reshuffle running processes, and it must be able to find a slot for newcoming processes as soon as that is requested. As processes come to and go from the memory, the memory gets fragmented and free space within the memory resembles scattered holes. It would be great if allocator might fill the coming processes into fitting holes, so as to reduce the memory fragmentation, but using existing techniques it is still not possible. Due to this, idea of making allocator intelligent is inevitable. This research is actually trying exactly that, to make memory allocator intelligent by using fuzzy logic, or more precisely Fuzzy Inference System (FIS).

The paper is organized as follows: the second part presents theoretical background on memory hardware and memory allocation algorithms and related work in the area of memory allocation, the third part elaborates on materials and methods used in this research, the forth part presents the fuzzy allocator simulation results, the fifth part discusses the results and the last part gives recommendations over the improvements of this work.

2. THEORETICAL BACKGROUND

Before going into explanation of allocation process, it is important to understand the generic memory hardware, memory architecture, and allocation techniques.

2.1 Memory Hardware

3D semiconductor memory consists of multiple banks, each representing single memory array **(Rixner et al, 2000).** Each array is connected via a set of lines, namely address, data and control lines, which are generally used to carry address and data, to indicate which function is to be performed and to report the memory status. Memory array is actually where the data is stored, and it represents a set of cells, each accessed via respective wordline and bitline, and each carrying one or more bits of information or data. The address lines are connected to address decoder, which would help select the appropriate cell in the memory and data line would then carry the data to and from the memory, based on the function issued by the control line (Gulak, 1998).

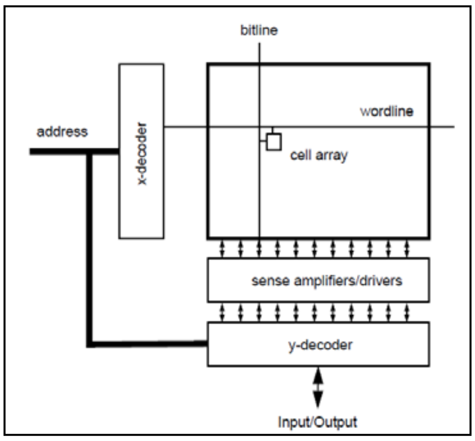

Figure 1: Generic Memory Architecture (Gulak, 1998)

Figure 1 presents generic memory architecture in 2D. Modern DRAM memory organization contains multiple memory arrays connected in parallel, with each array having its sense amplifiers and decoders. This one set of memory array and its decoders is called a bank, and it represents the 3rd dimension in memory architecture. The memory access then consists of three non-uniform access latencies, namely, bank precharge, row access and column access (Liu et al, 2010).

2.2 Memory Allocation Algorithms

The driving force behind every memory allocation algorithm is minimization of memory allocation time and reduction of memory fragmentation. Attempting that, numerous memory allocation algorithms emerged, and according to Wilson et al (1995), they can be sorted in following basic groups, sequential fit, segregated storage and fits, buddy systems, indexed fit and bitmaps. Each group is briefly presented below, and for detailed description of these algorithms please refer to (Johnstone&Wilson, 1997).

- **Sequential fit**, based on sequential search of doubly linked lists of all free blocks of memory. Due to that, they are not good for real-time systems (Masmano et al, 2004). These algorithms include First fit, Best fit, Next fit and Worst fit.
- **Segregated storage and fits**, where the lists of free slots are segregatedinto classes of different size, or different size range. Newly deallocated blocks are entered the class according to their size. Due to this segregation, this group of algorithms is good for real-time systems. Fast fit, Good fit and Best fit aresome of the algorithms used in this group.
- **Buddy systems**, uses splitting and coalescing the portions of memory. Memory is hierarchically split into portions called buddies. Some of the algorithms are Binary buddy, Fibonacci buddy and Double buddy.
- **Indexed Fit**, uses advanced tree-like structures to index the free memory blocks, Algorithms include Best-Fit, ”Fast-Fit”, etc. In real-time systems, indexed fir algorithms may outperform Segregated free lists (Masmano et al, 2004).
- **Bitmap Fit** is the extension of Indexed Fit, with the difference that it uses a bitmap to find out the state of the memory slots. The bitmap is stored in small portion of memory, and is easily accessible by the allocator. Example is Half fit algorithm.

For detailed description of these algorithms please refer to (Johnstone&Wilson, 1997).

2.3 Related Work

Most of the work regarding improvement and optimization of memory allocation was done using conventional, non-intelligent techniques, such as forcing the architectural, mathematical or hardware methods to improve memory allocation (list of references.) on different systems, such as embedded system, heterogeneous CPU-GPU or distributed systems.
On embedded and hierarchical systems, Sima&Bertels (2009) presented runtime memory allocation algorithm, based on the assumption that the memory is a form of reconfigurable logic array. The algorithm approached memory by computing the stores for each memory object mathematically. The implementation was done by directly allocating memory in the local scratch-pad memories. Some other works also approach the problems from mathematical matrix-based point of view, and can be found in (Schenk et al, 2000; Christen et al, 2007; Volkov&Demmel, 2008). The above mentioned works approach memory allocation from hardware architecture point of view.
Masmano et al (2004) developed a new dynamic storage allocation algorithm called Two Level Segregated Fit memory allocator for Real-Time Operating Systems (RTOS). It provided explicit allocation and deallocation of memory blocks. The algorithm assumed immediate coalescing, or merging of free blocks as soon as they were released. It implemented Good-fit strategy and allowed no reallocation or memory clean-up. To avoid non-uniform behavior it used the same strategy for all block sizes. Each free block in the memory belong to certain segregated list,

which holds the blocks of similar size, and the list is ordered by physical address. Each block within the list contains a header, which links it to the previous and next free block. The problem with this method is relatively high fragmentation which results from the bitmap-based mapping of segregated lists.
Xao et al (2004) proposed adaptive memory allocation method in their IEEE Transaction paper. Their method of memory reservation adaptively reserves a small set of workstations to provide special services to the jobs demanding large memory allocations. In other words, it will dynamically allocate additional distributed, shared memory resources for large tasks, and immediately after these task finish with the memory usage, the systems will adaptively switch back to the normal load sharing state. This method of memory allocation is specific for distributed or cluster systems.
Kim &Peng (2004) described the problem of memory as having conflict graph behavior. Based on that, they presented a memory allocation and assignment method, where memory partitioning was done to customize the memory architecture and optimize memory area and power consumption. The method extracted the useful exploration region to trade off area with energy consumption, and then performed an iterative multi-way partitioning is to optimize area and power. the method showed the reduction of cost in both, memory array space and power consumption.
In the field of artificial intelligence, and specifically fuzzy logic, nothing much was done to solve the problem of memory allocation, but the single paper by Zalevsky et al in 2002. In their paper “CPU and Memory Allocation Optimization using Fuzzy Logic”, Zalevsky et al claim fuzzy logic to be a powerful tool for dealing with unpredictable problems that are generally hard to define mathematically, such as allocation of memory or CPU. As part of the research presented in the above mentioned work, Zalevsky et al used previously designed optical fuzzy logic controller to optimize the CPU and memory distribution between multiple users on shared server machine. The algorithm was based on the observation of patterns of memory and CPU usage between heavy and light users. This pattern was then used as a base for derivation of the set of rules by the fuzzy logic inference engine that would result in optimization of system’s computing ability (Zalevsky et al, 2002). The rules were taking into consideration the user’s CPU time, the percentage of time using the server and the size of memory used. They found that the optimal rule dimension is 5 by 5 rule set, and their results showed the improvement in the utilization of CPU and memory by reducing the time each user used the shared system.
The work presented in this report goes further than what was proposed by Zalevsky et al, in the sence that the fuzzy algorithm is based not on the number of users, but rather on the process size. The overall concept of these two works is completely different, as our fuzzy allocator can be applied or extended to multiple systems, and can be modified to include even process priority in the input dimension. The following pages of this report will elaborate on the methods used for fuzzy allocator and will present and discuss the simulation results.

3. MATERIAL AND METHODS

Fuzzy Inference System (FIS) is one of the most popular methods of fuzzy logic (S. Guillaume, 2001). Its beauty lies in its simplicity, and its direct relation to human logic. Fuzzy Inference Systems (FIS) can be used to perform various tasks, such as classification, process simulation and diagnostics, process control and online decision support (S. Guillaume, 2001). This is the reason why this method was chosen for memory allocation problem. Following part briefly explains the concept of FIS and presents FIS-based memory allocator, called Fuzzy Allocator.

3.1 Basics of FIS

FIS consists of inputs, set of rules and outputs. The rules are used to map the inputs to the outputs. The input data are generally fuzzy, although fuzzy inputs may be combined with crisp inputs in order to produce fuzzy output (Ross, 2004). Figure 2 shows the block diagram of simple FIS with three inputs and one output.

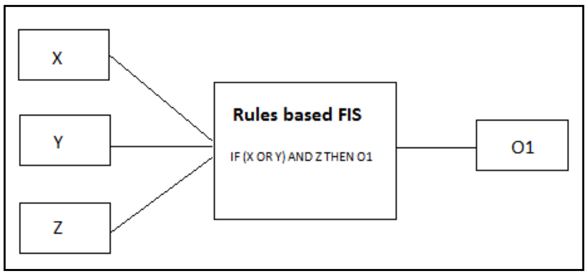

Figure 2: Block diagram of 3 input – 1output FIS.

In FIS, inputs must be fuzzified using the input membership functions. Numerous membership functions exist (e.g. triangular, Gaussian, sigmoid), and they are generally used to map the input value into fuzzified value between 0 and 1 (Ross, 2004).
After fuzzification, the rules for FIS must be set. The number of rules depends on the number of inputs and number of membership functions of each input. For multiple inputs, taking all possible rules would slow down the performance of the system. As such, it is preferred to choose the most influential rules, which are in turn represented by the mostinfluential variables. Variable selection and rule reduction are usually called structure optimization, and represent the most important step in the process of rule generation (S. Guillaume, 2001). There are multiple ways in which rule reduction can be performed, by grid partitioning and by clustering. Guillaume (2001) discusses methods in which grid partitioning may be performed. One method is implements all possible combinations of the given fuzzy sets as rules. This method

would generate 36 rules for our given inputs. The drawback here is the fact that large number of inputs would generate huge number of rules, which would make the system slow and impractical. Some of these rules may never even be used. Another method would be to choose the number of fuzzy sets dynamically, i.e. as the system runs, the rule sets get additional relevant rules. In another method, Wang and Mendel proposed that, when the number of rule combinations increases, only one rule should be used per data pair. They proposed a procedure to be followed when implementing this method and it can be found in (Wang & Mendel, 1992). The last method is using decision trees, which are a subspace of all possible rules. This generates incomplete rules but require a predetermined fuzzy partitioning. For clustering, Fuzzy C-means is the most popular method, and details may be found in (Guillame, 2001; Ross, 2004).

When making fuzzy rules, logical operators, such as AND, OR and NOT are used. Assuming A and B are fuzzy inputs and C is a fuzzy output, the logical operators will be represented as shown in Eq. 1-3.

$$C= A \cup B \qquad \text{(Eq. 1)}$$
$$C=A \cap B \qquad \text{(Eq. 2)}$$
$$C=A \sim B \qquad \text{(Eq. 3)}$$

Mathematically, these logical operators may be expressed via combination of min-max and complement operations (Zadeh, 1994). This is shown in following equations, (Eq. 4 and 5).

$$\mu_{(A \cup B)}(x)=\max\{\mu_{(A)}(x),\mu_{(B)}(x)\}\forall x\in X \qquad \text{(Eq. 4)}$$
$$\mu_{(A \cap B)}(x)=\min\{\mu_{(A)}(x),\mu_{(B)}(x)\}\forall x\in X \qquad \text{(Eq. 5)}$$

where X represents a set of data or objects, x an individual value of the data set X; A and B represent other sets containing data and μ (x) generally represents membership function over the set X, so μA(x) represents membership function that connects the set X and A, and μB(x) represents membership function that connects the set X and B.

The rules result in fuzzy output, which actually is the result of combination of all inputs using min-max methods. This is done as follows: first the input values are changed using membership function into a membership values. This membership values will then represent cuts to the related output triangles. For example, if 280 MB of process size touches two triangles, Normal with membership of 0.3 and Low with membership of 0.143, these cuts will be drawn and the output functions will be reshaped to produce area which will represent fuzzy output and will need defuzzification.

To get a crisp number out of the fuzzy output, defuzzification is needed. There are multiple methods used in defuzzification, such as centroid method, weighted average, max membership and mean max membership method. In this work, we will use centroid defuzzification method, which returns the center of area under the fuzzy output curve. For detailed explanations of defuzzification methods please refer to (Ross, 2004).

3.1.1 Inputs and Output

Process size is based on common process sizes and is expressed in Megabytes. The membership functions are defined for very small, small, medium and large processes and their respective size ranges are given in Table 2. The size ranges are taken based on the observation of running processes via Task Manager. Various applications, such as web browser, text editor, mathematical software, programming language tools, simulation packages, etc, were initiated and run in order to see their respective process size in memory. The process size input is given in Figure 3.

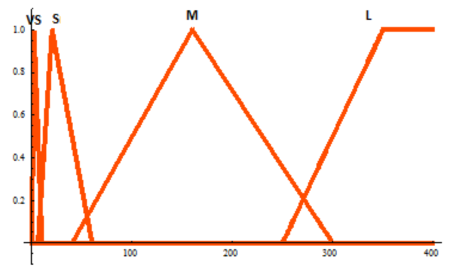

Figure 3: ProcessSize membership functions

Hole size reflects the empty or free spaceslots within the memory. The list of empty slots, technically called holes, is maintained by operating system in form of linked lists. The fuzzy allocator obtains this list and assigns the holes according to their sizes, which are reflected by membership functions listed in Table 1. Their ranges match the process membership sizes shown in Table 2.

Bankis the third input and representslogical memory bank, or the 3rd dimension of memory discussed above. Bank input holds two crisp values – *prev*, which represents previously used bank, or the bank which controller hardwarehad activatedbefore the new process allocation request came to the fuzzy allocator, and *new*, which represents any other bank, or any bank which the controller hardware will have to switch to (Figure 4). The reason for using bank input is because this information may help in optimizing allocation time by reducing the number of switches between the banks. The controller will not need to switch to another bank as long as the current bank contains sufficient holes to accommodate incoming processes.

Output is the order in which the processes will be allocated space in memory. The order ranges from high to low, where high would represent the processes to be allocated first, and low the processes to be allocated last. The membership functions are as follows: High (H), Above Normal (AN), Normal (N), Below Normal (BN) and Low (L). These membership functions overlap, and according to the rules set, it may be deduced that some process might be given 0.2 membership value of H, and 0.6 membership value of AN membership function, thus the output will be defuzzified and based on that the process will be given an order.

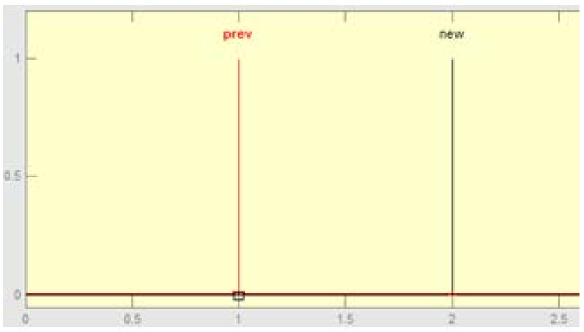

Figure 4: Two crisp values representin previous and new memory bank

Table 1: Fuzzy Inference System inputs and outputs

Inputs			Output
Process size	**Hole size**	**Bank**	**Order**
Very small-VS	Very small-VS	Previous-prev	High-H
Small-S	Small-S	New-new	Above normal-AN
Medium-M	Medium-M		Normal-N
Large-L	Large-L		Below normal-BN
			Low-L

Table 2: Membership function range (in MB)

Membership functions	Sizes (MB)
VS	[0-10]
S	[6-60]
M	[40-300]
L	[>200]

3.1.2 Rules

Rules to the FIS are made based on the process and hole sizes and bank, so that the allocation time is reduced to minimum. This was done by checking the empty holes in the previously accesses bank first. in other words, if the bank controller is currently accessing the bank 1, then the fuzzy allocator will first check that bank for a hole to accommodate some of the queued processes, and only if the bank does not contain the hole of desired size, the controller will switch to some other bank. Rules are formulated using Mamdanimodel.

If ProcessSize is X and holeSize is Y and bank is Z, then order is T.

X, Y and Z represent membership functions of each input and T is relative membership function of the output.

Following are several created rules:

- If ProcesSize is VS and hole size is VS and bank is prev, then order is H.
- If ProcesSize is S and hole size is M and bank is prev, then order is AN.
- If ProcesSize is S and hole size is L and bank is new, then order is L.

The number of all possible rules is the combinations of permutations of all membership functions for all inputs and output. Let N_X be the number of membership functions for input X, then the total number of possible rules $N_T$is:

$$N_T=N_{processSize}xN_{holeSize}xN_{bank} = 36$$

However not all rules are significant, and by reducing the number of rules, we get total of 16

Figure 2.The distribution of 2200 human chromosomes into seven Denver Group classes from A, to G.

4.RESULTS

The FIS-based fuzzy allocator was simulated using Matlab v. 2012b and Wolfram Matematica v. 8. Matlab contains toolboxes for fuzzy systems, and allow fast simulation of FIS, with graphical representations of membership functions, rules and the result. The simulation was later translated into Matematica code, where the rules were additionally optimized and fuzzy allocator performance was tested on the arbitrary memory request queue.

The simulation setup was tested as follows: first, the FIS was setup as described in previous sections. Three arrays *memory_bank*, *process* and *hole* were set. *memory_bank*represented the logical memory banks with busy and free slots, *process* represented incoming queue of memory requests, and *hole* represented list of free slots in the memory. Based on the rules, the simulation ordered the incoming request so that the overall performance is improved.

For example, if the size of the incoming process is 280 MB, and if the free slot of size 300 MB is situated on the previously accessed bank, the output value is 1.36 (1 represents H and 5 represents L order). The output is shown in Figure 5, where thefirst image in the figure represents AN (Table 1) output, and the height of the shaded region is the membership number of the output for this rule, and it amounts to 0.15. The second image shows the membership representing the order H, with membership value of 0.3. The third image in Figure 5, shows the result of defuzzification process, and red pointreflects the defuzzified value of the output, i.e. 1.21. From the output it may be deduced by almost certainty (i.e. by value 1.21) that this process has high priority in memory allocation. Figure 6 shows the related surface area of given input.

Just for comparison, another input of same memory and hole size is given, with hole situated on different memory bank - [280, 300, 2], and respective output and surface area are shown in Figures 7 and 8. The defuzzified value in this case is 3.32, and it can be said that this process' priority is between N with membership of 0.3 and BN with membership of 0.14. Thus defuzzification using centroid method gives us the order of 3.32, in the space of importance from 1 to 5, where 1 represents the highest priority and 5 the lowest.

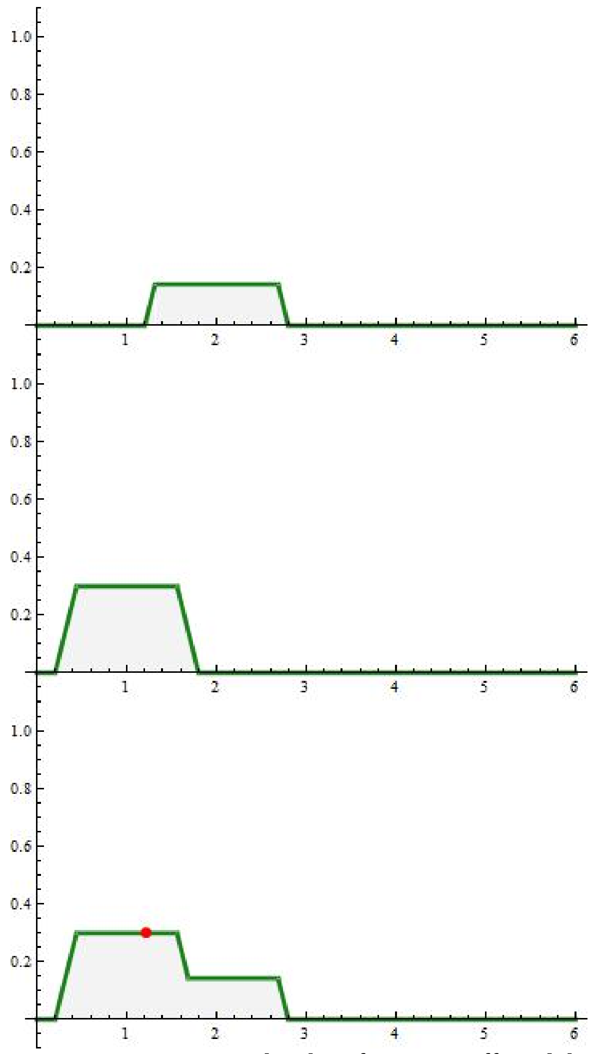
Figure 5: Output membership functions affected by the input (in order: AN order, H order, and fuzzy result – 1.21).

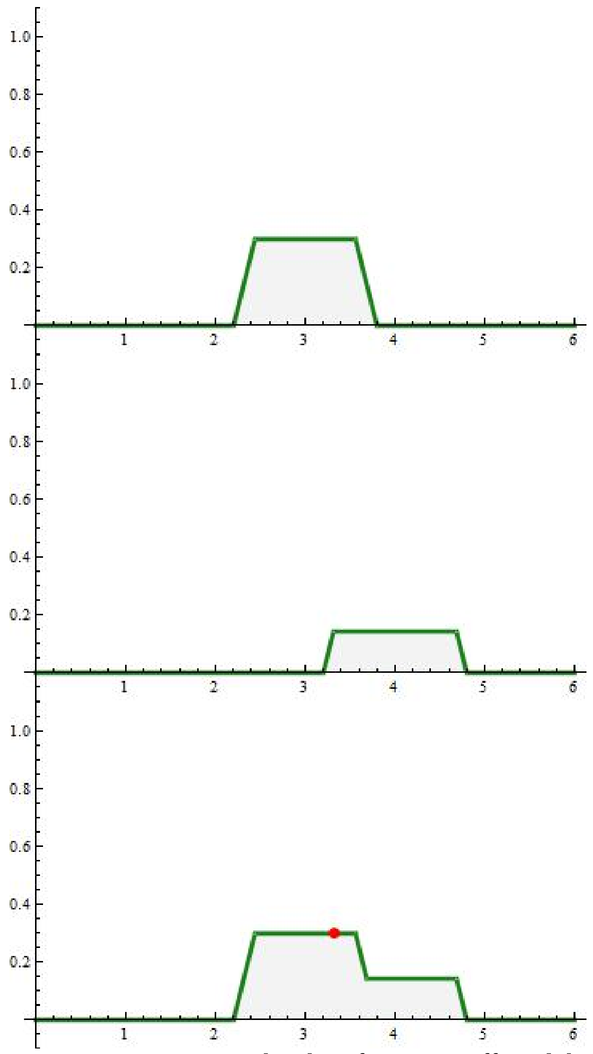
Figure 7: Output membership functions affected by the input (in order: N order, BN order, and fuzzy result3.32).

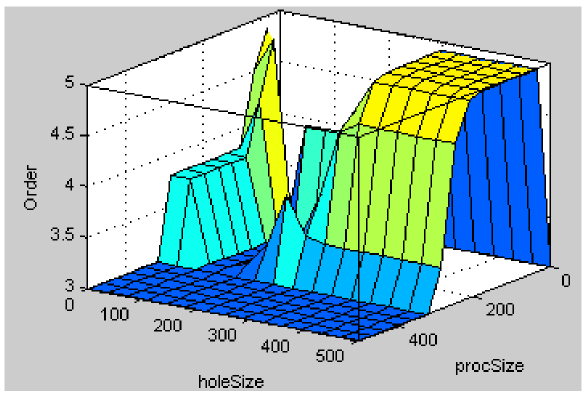

Figure 6: Surface area for first input [280, 300, 1]

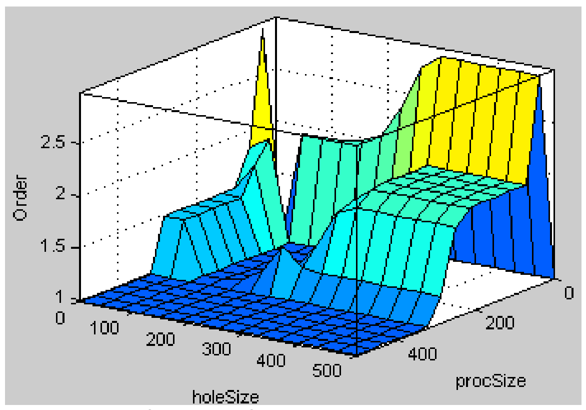

Figure 8: Surface area for second input [280, 300, 2]

5. DISCUSSION

Allocation of any process is done based on the size of the process, size of the memory hole, and location of that memory hole on the memory bank. Based on these three inputs, the Fuzzy Allocator gives the order in which the processes should be allocated the memory space, or memory holes. In other words, the fuzzy allocator directly gives the best possible order in which memory should be allocated for arriving processes. The order of allocation is represented by thedefuzzified numbers in ascending order.

The algorithm gives the ordering of memory allocation such as to minimize the memory latency, by allocating the memory holes on the current memory bank before moving to another memory bank.

6. CONCLUSSION

Memory allocation is widely known as one of the main problems and research interests in the area of memory management. Due to the constant increase of processing power and the growth and spread of distributed systems, the problem of memory allocation becomes even more complex. Creating a memory allocation system which closely resembles human intelligence would be great, and this research is trying to do exactly that.

The problem of memory allocation is attempted using Fuzzy Inference System, with three inputs and one output. The system is called Fuzzy Allocator.

The fuzzy allocator takes process size, hole size and memory bank as the input, and with the set of important rules, gives the order in which memory should be allocated to the coming processes. This system first tries to fill the currently active memory bank, and then moves to the next memory bank. In such a way latency which results from shifting between the banks is minimized. This system also minimizes the fragmentation within each memory bank, by trying to fit the process in memory hole as close to the processes size.

This work shows that fuzzy logic should be used to optimize the process of memory allocation in fast and efficient way. The authors recommend further research in this field with possible inclusion of some other methods of fuzzy logic, namely Fuzzy Pattern Recognition, in the problem of optimization of memory allocation.

REFERENCES

M. Christen, O. Schenk, and H. Burkhart, General-Purpose Sparse Matrix Building Blocks using the NVIDIA CUDA Technology Platform, Book of Abstracts, First Workshop on General Purpose Processing on Graphics Processing Units, Boston, Oct 04, 2007.

D. Diwase, S. Shah, T. Diwase and P. Rathod. “Survey Report on Memory Allocation Strategies for Real Time Operating System in Context with Embedded Devices” International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622, Vol. 2, Issue 3, May-Jun 2012, pp.1151-1156.

S. Guillaume (2001). “Designing Fuzzy Inference Systems from Data: An Interpretability-Oriented Review”. IEEE Transactions on Fuzzy Systems, Vol. 9, No. 3, June 2001.

M. S. Johnstone& P. R. Wilson (1997). “The Memory Fragmentation Problem: Solved?”. Proceedings of the 1st International Symposium on Memory Management (ISMM '98), pp. 26 – 36. DOI: 10.1145/286860.286864

G. Karady, “Introduction to Fuzzy Logic Systems”. Retreived from WWW http://enpub.fulton.asu.edu/powerzone/loadforecast/fuzzy.htm on 23.01.2013.

N. Kim & R. Peng. “A Memory Allocation and Assignment Method Using Multi-Way Partitioning”.Proceedings 2004 IEEE International SOC Conference, September 12-15, 2004, Hilton Santa Clara, CA, USA. ISBN: 0-7803-8445-8.

M. Masmano, I. Ripoll, A. Crespo, and J. Real (2004). “TLSF: a New Dynamic Memory Allocator for Real-Time Systems”. Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS’04). pp. 79-86. DOI: 10.1109/ECRTS.2004.35

S. A. McKee (2004). “Reflections on the memory wall”.Proceedings of the 1st conference on Computingfrontiers (CF '04).ACM, New York, NY, USA, 2004.DOI=10.1145/977091.977115.

A. Moallem& S. A. Ludwig (2009). “Using Artificial Life for Distributed Scheduling”.Proceedings of the 2009 ACM symposium on Applied Computing (SAC ‘09), March 08-12, 2009, Honolulu, Hawaii, USA. pp. 1091-1097. ISBN: 978-1-60558-166-8

T. Ross, “Fuzzy Logic with Engineering Applications”. John Wiley & Sons, Aug 16, 2004.

V. Saxena, Y. Sabharwal& P. Bhatotia, (2010). "Performance evaluation and optimization of random memory access on multicores with high productivity",2010 International Conference onHigh Performance Computing (HiPC), 19-22 Dec. 2010. DOI: 10.1109/HIPC.2010.5713168.

O. Schenk, W.Fichtner& K. G¨artner “Scalable Parallel Sparse LU Factorization with a Dynamical Supernode Pivoting Approach in Semiconductor Device Simulation”. Proceedings of the 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation.August 21-25, 2000, Lausanne, Switzerland. (2000).

H. Thimbleby (1993), “Computerized Parkinson’s Law”, Computing & Control Engineering Journal, October 1993.

V. Volkov& J. W. Demmel. “LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs”.LAPACK Working Note 202. EECS Technocal Report, 2008.

L.-X.Wang and J. M. Mendel, “Generating fuzzy rules by learning from examples,” IEEE Trans. Syst., Man, Cybern., vol. 22, pp. 1414–1427, Nov./Dec. 1992.

P. R. Wilson, M. S. Johnstone, M. Neely & D. Boles, (1995). “Dynamic Storage Allocation: A survey and critical review”. In 1995 International Workshop on Memory Management.Springer Verlag LNCS.Kinross, Scotland, UK, 1995.

L. Xiao, S. Chen & X. Zhang. “Adaptive Memory Allocations in Clusters to Handle Unexpectedly Large Data-Intensive Jobs”. IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 7, July 2004

L. A. Zadeh, “Fuzzy Logic, Neural Networks, and Soft Computing”. Communications of the ACM, Vol. 37, No. 3,March 1994.

Z. Zalevsky', E. Gur& D. Mendlovic. “CPU and Memory Allocation Optimization using Fuzzy Logic”.Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation V, Proceedings of SPIE Vol. 4787 (2002).