International Journal of Computer Theory and Engineering, Vol. 1, No. 5, December, 2009
1793-8201

A Novel Approach for Estimation and Optimization of Memory In Low Power Embedded Systems

Srilatha C and Dr. Guru Rao C V

***Abstract* - The paper describes a method for estimation and optimization of memory size in low power embedded systems. This approach can be treated as a pathfinder to efficiently optimize the memory module, in turn optimizing the design time. It can be even employed for high level memory exploration applications while successfully meeting the performance – cost design metrics of the system. The paper concludes with an implementation example of a Speech Recognition module, showing an effective reduction in the memory requirement of the system after memory optimization. Depending upon the results, even algorithm based optimization can be done with an aim of further reducing the memory size.**

***Keywords* - Memory, Embedded systems.**

I. Introduction

In today's embedded systems, memory represents a major bottleneck [1] in terms of cost, performance, and power. Optimal designing of memory space is very crucial in obtaining a cost effective embedded system. Also, a huge amount of array processing is being involved in current day embedded applications. Hence it is very critical to come out with methodologies for memory size estimation. A huge amount of array processing is being involved in current day embedded applications, which require both on-chip and off-chip memories. Thus it is important to efficiently predict the memory requirements for the data structures and code segments for that particular application. Memory requirement is defined as the number of locations needed to satisfy the storage requirements of a system. It is very important to effectively predict the system's memory requirements without synthesizing, in order to obtain a high profile end product, as it results in a reduced design time. Here we aim at reusing of memory space, thus giving a fast estimate of memory size. Though addressing becomes complex, it is preferable to allow sharing among arrays which aids in optimizing the memory size. Depending upon the results, even algorithm based optimization can be done with an aim of further reducing the memory size.

The paper is organized as follows:

Section 2 briefly reviews some previous work done in the area of memory estimation and optimization. The proposed methodology is described in Section 3. Section 4 gives a brief description about the embedded speech recognition front end module, while its experiment set up is explained in section 5. Section 6 and 7 concludes the estimation results and optimization strategies of the task implemented.

II. Related Work

Memory estimation methodology can employed for high level memory exploration applications while successfully meeting the performance-cost design metrics of the system. It is very important to provide good memory size estimates with reasonable computation effort without performing complete memory assignment for each design. In data dominated applications, such as digital image, video or speech signal processing applications, summing up the sizes of all the arrays is the most straightforward way to get an upper bound of the memory requirement

For general purpose systems whose area of application is vast, the dynamic memory allocation is supported by custom managers [2]. Also, [3] [4] showed memory optimizations and techniques to reduce memory footprint along with power consumption and performance factors on static data for embedded systems. Array based data flow preprocessing considers program size as well as data size [5] is applicable only for partially fixed execution ordering. In [6], the design metric constraints were area and number of cycles, while the proposed methodology also considers power consumption. Live variable analysis along with integer point counting method [7] is not applicable for large multi-dimensional loop nest as it needs complex computations. [8] Is based on analysis of memory size behavior taking into account that signals with non-overlapping lifetimes share same memory locations. Also it showed upper and lower boundaries for memory map, while this paper presents an approach that tries to gives a very close estimate.

Memory system design for video processors [9] had constraints on area, cycle time. [10] proposed data memory size and number of cycles as design metrics. Memory allocation problem [11] was solved by meeting optimum cost but efficient memory access modes were not exploited. To reduce the power consumption during memory optimization, loop transformation reordering [12] was introduced, while loop transformation ordering is much beneficial. Our approach works for multimedia applications involving large array processing. It can be even employed for high level memory exploration applications while successfully meeting the power, performance – cost design metrics of the system.

III. Approach

The output of the approach is an optimized range of memory size. The estimated memory size for the given input application lies in this range. Memory is basically

divided into two segments, namely- the program memory and the data memory, which includes heap and stack. When a function calls another function which in turns calls another function etc., the execution of all those functions remains suspended until the very last function returns its value. This chain of suspended function calls is the stack, because elements in the stack (function calls) depend on each other. Heap is defined as a portion of memory used to store variables. These variables do not depend on any other variables. They can be accessed in random. Heap starts growing from the bottom of the program segment. For dynamic allocation of memory, memory locations are reserved. And those locations are returned back when there is a memory free. Stack and heap scale up and down in reverse directions as shown in figure 1.

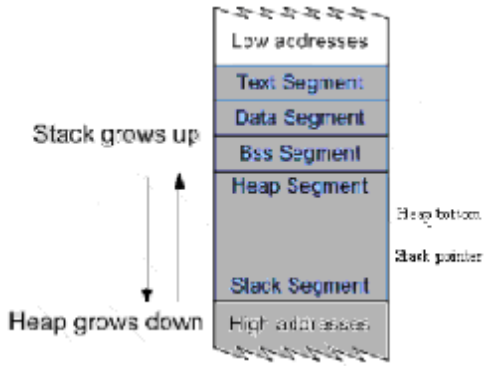

Figure 1: Memory Organization

In the due process, any collision is treated as an error. At any instant of time, the total memory size of a system is calculated as the cumulative sum of the program memory, stack and heap. Thus taking into account the actual code size, a memory trace can be developed. This results in giving the final memory requirement profile of the system.

Also, C language offers considerably good control of memory usage, over other memory-managed languages like Java, allowing us to precisely optimize memory allocations. Any memory location has to be tracked which the program dynamically allocates and then releases that memory when the program no longer needs it. Otherwise, the program will either introduce memory leaks or consume memory inefficiently. C language also allows manipulation and access of memory via pointers. To dynamically request memory buffers the *malloc*(), *realloc*(), or *calloc*() function calls are used. To release these resources once they're no longer required, *free*() is employed. The system's memory allocator satisfies these requests by managing the heap. A program can erroneously or maliciously damage the memory allocator's view of the heap. For example, this corruption can occur if your program tries to free the same memory twice or if it uses a stale or invalid pointer.

Any given input application can be divided into 3 layers.
1. Process Control Flow: sequencing
2. Loop Hierarchy and Indexed Signals:
Single- assignment code
3. Arithmetic, Logical and Data-dependent
Operations

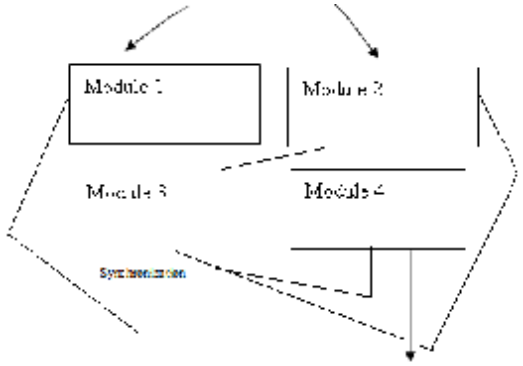

Figure 2: Layer 1- process control flow

```
For (i= 0; I<N; i++)
      For (j=0; j<M; j++)
        If (I = = 0)
        B[i] [j] = 1;
         Else
         B[i] [j] = func1(A[i][j], A[i-1][j]);
```

Layer 2 – Loop Hierarchy

```
Int
        Funct1(int a, int b)
        {
          Return a*b;
         }
```

Layer 3 – involves only Arithmetic, Logical and Data-dependent Operations

The following is the algorithm developed for effective memory estimation and optimization, being divided into three parts. Taking the input application description which may be constituting of parallel constructs, the first part analyses the memory size variations, by approximating the memory trace. The second part estimates the memory size while the third part optimizes the predicted size.

The following is the proposed memory estimation and optimization algorithm developed.

Step1: Take the input specification containing multidimensional arrays
Step 2: Perform data dependant analysis
} Memory analysis

Step 3: Hierarchical rewriting
Step 4: Hiding undesired constructs
Step 5: Code expansion
Step 6: Data flow analysis
Step 7: Partitioning
} Memory size estimation

Step 8: Data flow transformations
Step 9: Loop control transformations
Step 10: Data reuse
Step 11: Memory assignment
Step 12: Memory organization
} Memory optimization

Improving the Accuracy
Step 13: Go to step 2

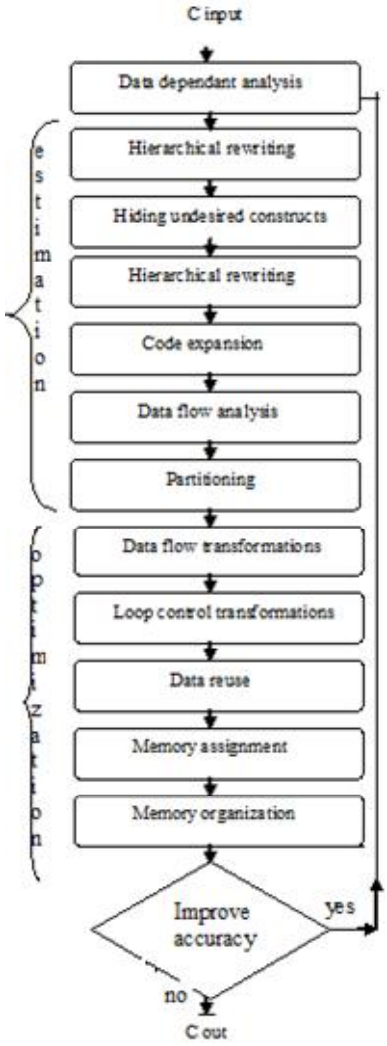

Figure 3: Memory estimation and optimization flowchart

Step 1 involves taking of the input specification containing multidimensional arrays for further processing. Computation of data dependence of array elements is carried out in step 2.

Step 3: Hierarchical rewriting

It involves hiding of code parts without data transfer and storage exploration freedom in "layer 3" functions.

```
Consider:    for (i=0; I<N, i++)
                  If (i<10) funcA ( );
                  functB ( );

Hierarchical rewriting
       for (i=0; I<4, i++)
                  functA ( );
                  functB ( );
             for (i=10; i<N, i++)
                  functB ( );
```

Step 4: Hiding undesired constructs

It hides data-dependent conditions, scalar and logic operations "layer 3".

```
Consider:    for (j=0; j<=N, j++)
                  If (j= = 0)
                     a[j] = b[j];
                  Else
                  If (a[j-1] = = b[j])
                     a[j] = in[j];

                   else
                   a[j] = b[j] + 5;

Hiding of undesired code
for layer 2:    for (j=0; j<=N, j++)
If (j= = 0)
a[j] = b[j];
Else
a[j] = f(a[j-1], b[j], in[j]);

for layer 3:
                  int f(intx, int y, int in)
{
If (x= = y)
Return in;
Else return y+5;
}
```

Step 5: Code expansion

It performs the Code expansion of functions with Multidimensional data-flow.

```
Consider:   int *p = malloc(200 * 200 *sizeof(int));
for (i=0; i<200; i++)
for (j=0; j<200; j++)
*p = f(i,j)
P++;

Int p [200] [200]'
for (i=0; i<200; i++)
for (j=0; j<200; j++)
p [i] [j] = f(I,j)
```

Step 6: Data flow analysis

It involves Array/pointer data-flow analysis along with single assignment to increase optimization freedom. Depending upon the input specification, data-flow chains (recursions and conditions) and less crucial data types (weight based) can be removed.

Single assignment clearly describes the data flow.

```
Consider:    s[0] = 0;
for (i= 1; i<=4; i++)
s[i] = s[i-1] + x[i];
sum = s[4];
```

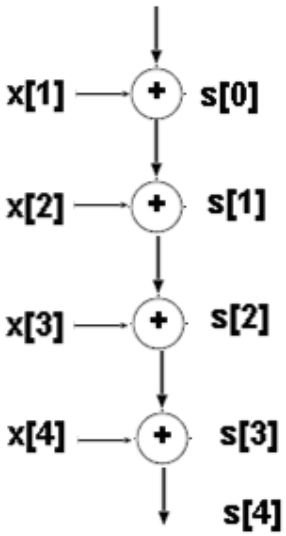

Figure 4: Single assignment

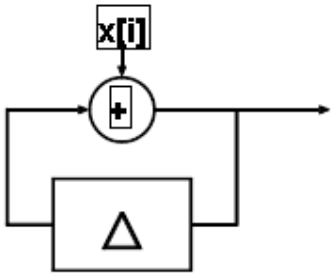

Figure 5: Multiple assignment

Consider the loop structure for a video task application for motion estimation.

```
For (y= 0……y)
For (x = 0……x)
z_min = infinity;
For (z_y = 0…..M)
For (z_x = 0…..M)
z = 0;
For (p_y = 0….N)
For (p_x = 0….N)
z += abs(prev[] – curr[]);
If (z<z_min)
z_min = z;
```

Where: X – x axis
Y = y axis
M = previous frame
N = current frame

Interchange is not possible between lines 5 and 7.

Making a small change in the above program at line 6,

```
For (y= 0……y)
For (x = 0……x)
z_min = infinity;
For (z_y = 0…..M)
For (z_x = 0…..M)
z [z_x] [z_y]= 0;
For (p_y = 0….N)
For (p_x = 0….N)
z [z_x] [z_y]+= abs(prev[] – curr[]);
If (z<z_min)
z_min = sad;
```

Interchange is possible between lines 5 and 7. Thus single assignment aids in order to obtaining full degree of freedom. Figure 5 shows the need of Single assignment for array data-flow analysis.

Consider the following array;

```
For (i= 0; i<N; i++)
For (j = 0; j<N; j++)
a[i] [i+j] = …….. ;
```

```
a[1*i + 0*j + 0][1*i + 1*j +0] = ……
a[x(i,j)] [y(i,j)] = …….
```

Where x(i,j)] = 1*i + 0*j + 0
y(i,j) = 1*i + 1*j + 0

0< i<N
0<j<N

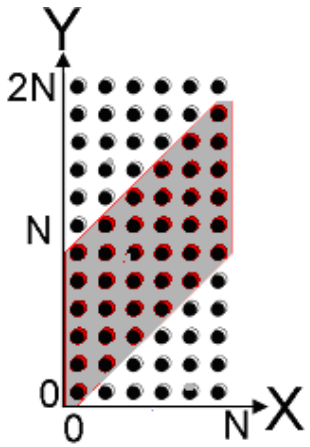

Figure 6: Array data flow analysis

Step 7: Partitioning

Partitioning of graphs to exploit divide-and-conquer concept is implemented in order to shorten the exploration time.

Step 8 and 9: Data flow and Loop transformation

It aims at goals regularity and locality of reference. Loop reordering allows arrays to share memory space, thereby reducing the size of the on chip memory. Loop interchange helps to reduce the number of memory reads. The number of memory accesses and the size of storage significantly reduce. However, each transformation has its own special legality test based on the direction vectors and on the nature of loop bound expressions.

During implementation the following are the two key positions identified where memory usage alters. Also memory traces are captured. Consider that the bottom address of heap is H.

1. For dynamic allocation of memory, memory locations are reserved. And those locations are returned back when there is a memory free. When a memory function is called:

- if there exists any free memory location at the center of the heap, then H does not change.
- if the called function capacity is less than the heap size, then H does not change.
- Else, H changes. Ie. Increases.
- Else, any location that is very close to H is emptied, then H decreases.

2. with changing of the stack pointer.

IV. Embedded Speech Recognition Front-End Module

Speech recognition is rapidly becoming one of the most popular embedded real-time multimedia applications. For such sensitive applications, entire processing has to be done using embedded modules. Hence, memory analysis of such a system is very valuable. Markov method is employed for time variants having discrete state spaces. Each of the discrete space state gives out speech perceptions as per its probable distribution. Thus obtained speech perceptions can be either discrete or continuous. They basically represent frames ie. Durations of fixed time. As the states cannot be observed directly, it is termed as hidden Markov model. The following speech recognition algorithm contains of two parts. They are the search algorithm and the processing part [13]. First the entire input speech is converted into vectors representable in probability space. Then the high probable events of the space are identified with the help of the algorithm. This search algorithm basically runs under tightly constrained environment. The following is the speech processing algorithm developed.

Step 1. Input Timing Waveform

Step 2. Premphasis

Step 3. Hamming window

Step 4. Coefficients for autocorrelation

Step 5. Estimation of level

Step 6. Recursion

Step 7. Speech parameters

Step 8. Speech functions

Step 9. Hidden Markov model

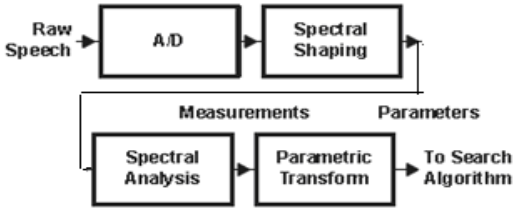

Figure 7 Block diagram - Speech Processing

Figure 7 shows the block diagram of the speech processing embedded system. The analog to digital converter converts the input speech signal into a digital equivalent by using the sampling technique. Depending upon the processor and the converter the sampling rate can be up to a maximum of 8KHz per second. In order to include some frequency parameters into the signal, Spectral shaping is carried out with the help of an FIR filter. Hence this stage is known as a preemhasis stage, which basically employs a single coefficient filter. Next stage is the windowing. In order to eliminate spectral leakage this stage is introducing. To find any data errors within the windows, hamming window is implemented. Data frames can be obtained with the help of data window and the sampled signal. The outputs generated by the system are obtained by orthogonalisation of filter outputs. The outputs obtaining from the filter are 14 Mel spaced values. These output values along with other corresponding values, speech level, energy difference, speech characteristics and the relative speech level form a 32 vector element. This is known as generalised speech parameter. Generalized speech feature is defined as the 10 vector element obtained by multiplying generalised speech parameter by a linear transform value [14].

V. Experimentation

The proposed algorithm for memory estimation and optimization was implemented on TMS320C6701 Floating Point Digital Signal Processor. Firstly, the algorithm was developed in C-language programming environment. In the second phase, the C-language algorithm was ported to the processor platform. Thus speech recognition embedded application is implemented using the proposed methodology.

VI. RESULTS

Memory trace result

Using the proposed memory estimation methodology, the memory analysis results are shown. Memory trace for the implementation is shown in fig. 7. It considers program and data segments on the X axis and required memory size on the corresponding Y axis. A plot of it results in a memory trace which is the estimated size that caters the storage requirements of both program and data segments in accomplishing the task of speech recognition.

The program segment size of this implementation is 68Kbytes and the data segment memory required is 14Kbytes. In total, 82Kbytes memory is needed.

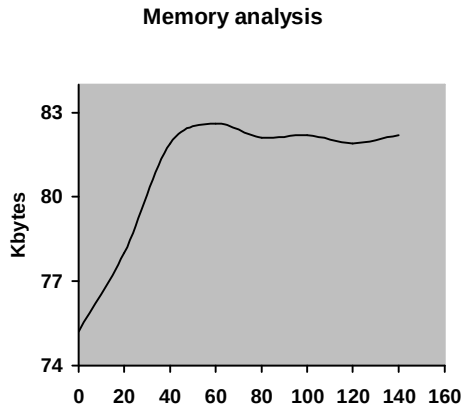

Figure 8: Memory estimation for Program and data segments

Memory optimization

For sensitive applications involving large array processing, the entire processing has to be done using embedded modules. While using such modules, care should be taken to meet optimized profile for the design metrics. Fig. 9 shows an optimized memory plot. It considers program and data segments on the X axis and required memory size on the corresponding Y axis. Employing memory optimization methodology results in a reduction of 28Kbytes. With help of loop transformation technique, relatively a good amount of memory size requirement is reduced for the arrays.

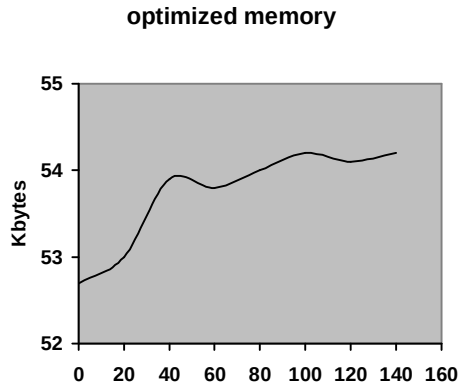

Figure 9: Memory optimization for Program and data segments

VII. Conclusions

This paper describes a method for estimation and optimization of memory size in low power embedded systems. The approach presented efficiently optimizes the memory module, in turn optimizing the design time. High level transformations such as loop transformations are applied to reduce the number of off chip memory accesses and also the on chip memory requirement. We validate this estimation and exploration procedure by performing experiments on a Speech Recognition module, showing an effective reduction in the memory requirement of the system. Fig. 10 shows the final comparative analysis between the estimated memory size and the optimized memory requirements.

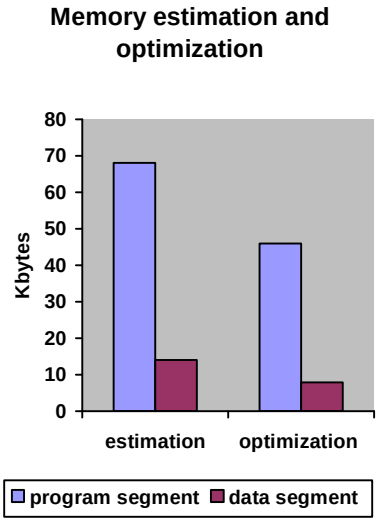

Figure: 10 Memory estimation and optimization

Acknowledgment

Author Srilatha would like to acknowledge and thank Azimuth foundation (CCR 4942/1997) for all the Grant-in-Aid provided for the research work. This work was supported in part by Aurora's Scientific Technological & Research Academy, India. Also her sincere feelings of gratitude to Mr. Ramesh Nimmatoori for all his support and encouragement which helped her to write this paper and wishes to thank Dr. C V Guru Rao for his valuable suggestions.

References

[1] Peter Grun, Nikil Dutt, and Alex Nicolau, "Access Pattern Based Memory and Connectivity Architecture Exploration", *ACM Transactions on Embedded Computing Systems*, Vol. 2, No. 1, February 2003, Pages 33–73.

[2] RTEMS Research, O.-L. A. RTEMS, "Open-source real-time operating system for multiprocessor systems" http://www.rtems.org. 2002.

[3] Panda PR, Catthoor F, Dutt ND, Danckaert K, Brockmeyer E, Kulkarni C, "Data and memory optimizations for embedded systems". *ACM Trans. Des. Automat. Elect. Syst.* 2001, 6. 2, 142–206.

[4] Benini L, De Micheli G, "System level power optimization techniques and tools". In *ACM Trans. Des. Aut omat. Embed. Syst.* 2000.

[5] PG Kjeldsberg, F Catthoor, EJ Aas, "Storage requirement estimation for data intensive applications with partially fixed execution ordering", *Proceedings of 8th International Workshop on Hardware/Software Codesign, San Diego*, May 3-5, 2000, pp56-60.

[6] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye, "Influence of Compiler Optimizations on System Power," 37*th IEEE/ACM Design Automation Conference*, 2000, pp. 304-307.

[7] Y Zhao, S Malik, "Exact memory size estimation for array computations without loop unrolling", *Proceedings of 36th ACM/IEEE Design Automation Conference, New Orleans LA*, June 1999, pp811-816.

[8] P. Grun, F. Balasa, N. Dutt, "Memory size estimation for multimedia applications", *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, Seattle WA, March 1998,pp.145-149.

[9] S. Dutta, W. Wolf, and A. Wolfe, "A Methodology on Evaluate Memory Architecture Design Tradeoffs for Video Signal Processors,"

IEEE Transactions on Circuits and Systems for Video Technology, vol.8, no.1, Feb. 1998.
[10] P. R. Panda, N. D. Dutt, and A. Nicolau. "Data Cache Sizing for Embedded Processor Applications." Technical Report ICS-TR-97-31, University of California, Irvine, June 1997.
[11] H. Schmit and D. E. Thomas, "Array Mapping Behavioral Synthesis", *ISSS*, 1995.
[12] F. Catthoor, F. Franssen, S. Wuytack,, L. Nachtergaele, and H. De Man, "Global Communication and Memory Optimizing Transformations for Low Power Signal Processing Systems", *Workshop on VLSI Signal Processing*, La Jolla, CA, Oct. 1994.
[13] Netsch, Lorin, "Acoustic Feature Processing Reference Guide," Texas Instruments.
[14] Yu-Hung Kao, "Robustness Study of Free-Text Speaker Identification and Verification," PhD Thesis, University of Maryland, 1992.

BIOGRAPHY

Ms. Srilatha C received her Bachelor's Degree in Instrumentation Engineering from Osmania University, Hyderabad, India. She is a Master degree holder in Embedded Systems from Jawaharlal Nehru Technological University, Hyderabad, India. Currently, she is an Assistant Professor at Aurora's Scientific Technological & Research Academy. She has six years of teaching experience at college level. Her area of interest includes embedded systems, Real time systems. She is carrying out her research work in the field of embedded systems under the guidance of Dr. C V Guru Rao, Principal, KITS College, Warangal, India. She is a life member of Computer Society of India and Instrumentation Society of India.

Dr. Guru Rao C V received his Bachelor's Degree in Electronics & Communications Engineering from VR Siddhartha Engineering College, Vijayawada, India. He is a double post graduate, with specializations in Electronic Instrumentation and Information Science & Engineering. He received his M.Tech in Electronic Instrumentation from Regional Engineering College, Warangal, India and M.E in Information Science & Engineering from Motilal Nehru Regional Engineering College, Allahabad, India. He is a Doctorate holder in Computer Science & Engineering from Indian Institute of Technology, Kharagpur, India. With 24 years of teaching experience, currently he is the Principal, KITS Warangal, Andhra Pradesh, India. He has more than 25 publications to his credit. He is the Chairman, Board of Studies for Computer Science & Engineering and Information Technology, Kakatiya University, Warangal. Also, he is the Editorial Board member for International Journal of Computational Intelligence Research and Application journal. He is a life member of Indian Society for Technical Education, Instrumentation Society of India, and member of Institution of Engineers, Institution of Electronics & Telecommunications Engineers and Institution of Electrical & Electronics Engineers (USA).