

## **An Integrated Approach for Execution Time Reduction in Embedded Applications**

Shilpa.C<sup>1</sup>, K. Sivasankari<sup>2</sup>

<sup>1</sup>(Electronics and communication engineering, Akshaya college of engineering and Technology, INDIA)

<sup>2</sup>(Electronics and communication engineering, Akshaya college of engineering and Technology, INDIA)

---

**ABSTRACT:** Scalable memory multiprocessors are becoming increasingly popular platforms for high-performance scientific computing. In order to improve application performance on these machines, it is essential to divide computation among processors. In complex embedded systems, the growing trend is to build a multiprocessor system on chip (MPSoC). A MPSoC consists of multiple heterogeneous processing elements, a memory hierarchy and input/output components which are linked together by an on chip interconnect structure. Many embedded systems employ software managed memories known as Scratch Pad Memories (SPM). In SPM, execution time of applications on systems can be accurately predicted. Scheduling the tasks and partitioning the embedded application on the processors are two critical issues in such systems by using ASAP and ALAP algorithms. In this paper, an integrated approach is presented for effective task scheduling and SPM partitioning by round robin algorithm. The execution time of embedded applications is reduced by performing round robin algorithm. Results on Lame benchmark show the significant improvement from our proposed technique.

**KEYWORDS:** Execution time, Memory partitioning, Multiprocessor system on chip(MPSoC), Scheduling, Scratchpad memory.

---

### **I. INTRODUCTION**

Architectures with multiple processors on a single chip have become an attractive solution to achieve performance in both high-end and low-end computing due to clock and power constraints. An MPSoC consists of multiple heterogeneous processing elements, memory hierarchies and I/O components interconnected by complex communication architecture provides the flexibility of simple design, high performance and optimized energy consumption. While embedded systems become increasingly complex, the increase in memory access speed has failed to keep up with the increase in processor speed. This makes the memory latency a major issue in scheduling embedded applications on embedded systems. A MPSoC is an attractive solution to the increasing complexity and size of embedded applications on embedded applications. Execution time complexity any size of embedded applications; this means that data caches are not suitable since it is hard to model the exact behavior and to predict the execution time of programs. To alleviate such problems, many modern MPSoC systems use software controlled memories known as scratch pad memories (SPMs), which allow execution times to be predicted with accuracy. Unfortunately, SPMs are expensive and hence they are usually of limited size. Many multiprocessor systems on chip models use a memory hierarchy with slow off chip memory and fast on chip scratchpad memories. Such hierarchy means that proper allocation of variables to the on chip memory is an essential part in reducing the off-chip accesses. The computation time of a program on a processor depends on the amount of SPM allocated to the processor executing this task. The problem of task scheduling and memory allocation on MPSoCs is a NP (Non Polynomial) complete problem. Traditionally, these two steps are performed separately where tasks are usually scheduled first and formed separately where tasks are usually scheduled first and the SPM budget is then partitioned among the processors. Such a decoupled technique minimize the computation time of the whole application. The integration of those two steps is critical to improve the performance. To the best of our knowledge this is the first work to present integrated approach to task scheduling and memory partitioning by round robin algorithm.

### **II. PREVIOUS COMPARITIVE STUDIES**

Many research groups have studied the problem of task scheduling of applications on multiple processors where the objective is to minimize execution time. Benini et al [3] solved the scheduling problem using constraint programming and the memory partitioning problem using integer linear programming. Kwok and Ahmed [2] presented a comparison among algorithms for scheduling task graphs onto a set of homogenous processors on a diverse set of benchmarks based on set of assumptions.

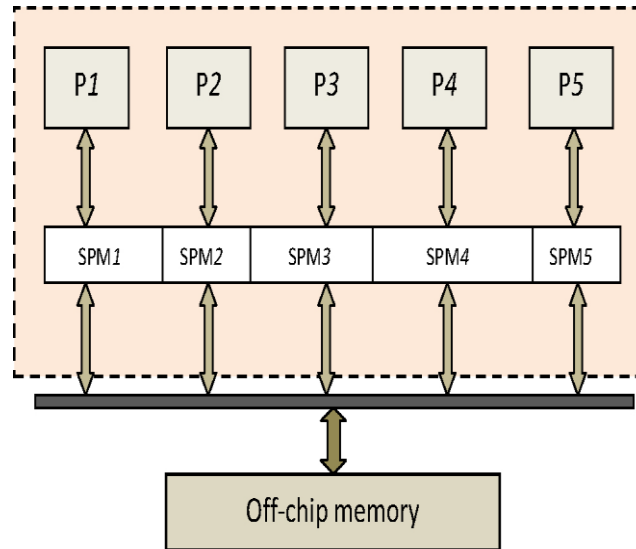


Fig. 1. Architectural model with five processors and sharing of SPM budget

De Micheli et al [4] studied the mapping and scheduling problem onto a set of processing elements as a hardware/software co-design. Neimann and Marwedel [5] used integer programming to solve the hardware/software co-design partitioning problem. A tool for hardware-software partitioning and pipelined scheduling based on a branch and bound algorithm. Similarly, Kuang et al [7] proposed an ILP solution for the partitioning problem with pipelined scheduling. Cho et al [8] proposed an accurate scheduling mode of hardware-software communication architecture to improve timing accuracy. Panda et al. [9],[10] presented a comprehensive allocation technique for scratchpad memories on single processor. Optimal ILP formulations for memory allocation for scratchpad memories were presented in [11] and [12]. An ILP formulation to the SPM allocation problem to reduce the code size was presented in [13]. Steinke et al. [14] formulated the same problem with the objective to minimize the energy consumption. Angiolini et al [15] optimally solved the problem of mapping memory locations to SPM locations using dynamic programming. Many authors have studied the memory allocation problem in MPSoCs by a focus on data parallelism in the context of homogenous multiprocessor systems. Kandemir et al [16] presented a compiler based strategy for optimizing energy and memory access latency of array dominated. In [17], the authors proposed an ILP formulation for the memory partitioning problem on MPSoC. Suhendra et al [18] studied the problem of integrated task scheduling and memory partitioning among a heterogeneous multiprocessor system on chip with scratch pad memory. This is the only paper that addressed this problem in an integrated approach for MPSoC. They formulated this problem as an integer linear problem (ILP) with the inclusion of pipelining. ILP solutions require long computation time for large applications.

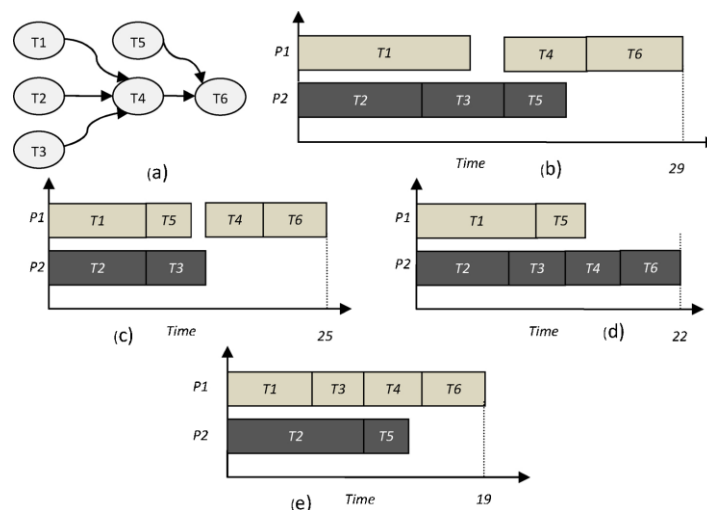


Fig. 2. (a) Task Dependence graph based on (b) no SPM, (c) equal partitioned SPM, (d) nonequal partitioned SPM, (e) integrated approach.

### III. PROBLEM DEFINITION

Fig.1 depicts a typical MPSoC which consists of multiple processors, a shared SPM budget divided among the processors, and a global off chip memory that can be accessed by all processors. Our techniques can also be used for an architecture where each processor has private SPMs of other processors. Dividing an application into a set of tasks where one or more independent tasks can be executed in parallel on the available processors. Parallelism leads to potential for speeding up the execution time; this is a major issue in embedded processors. Embedded applications usually consist of computation blocks, which are treated as tasks. There are usually dependences between tasks that should be respected in the schedule. Problem is defined based on task dependence graph (TDG). TDG is a directed acyclic graph with weighted edges where each vertex represents a task in the embedded application. An edge from task 1 to task 2 represents a scheduling order. Data is transferred from task 1 to task2 and executed. Communication cost is the weight of this edge. Processor starts executing only after task 1 performs necessary data communication. Processors are mapped with necessary tasks. The processors are heterogeneous. Data variables are accessed by execution cycles of task. Data variable accessed by SPM is 100 times faster than the off chip memory. By narrowing the gap with the processor's speed there is good utilization of SPM. The problem is defined as 1) schedule the tasks on available processors 2) SPM partitioning among the processors 3) data variables are assigned to certain task on processor P with a private SPM assigned. The objective is minimization of execution time in cycles of execution of the embedded application on the MPSoCs architectural model

### IV. ROUND ROBIN ALGORITHM

The scheduler maintains a queue of ready processes and a list of blocked and swapped out processes. The Process Control Block of newly created process is added to end of ready queue. The Process Control Block of terminating process is removed from the scheduling data structures. The scheduler always selects the Process Control Block from the head of the ready queue. This is a disadvantage, since all processes are basically given the same priority. Round robin also favors the process with short CPU burst and penalizes long ones. When a running process finishes its time slice, it is moved to end of ready queue. A time slice is an amount of time that each process spends on the processor per iteration of the Round Robin algorithm. All processes are executed in a first come first serve manner but are pre-empted after a time slice. The process will either finish in the time slice given to the process will be returned to the tail of the ready queue and return to the processor. An embedded applications is divided into tasks and for each tasks ASAP ( As Soon As Possible ) values is calculated. Based on the increasing order of values, tasks are stored in list. When the list is full, first task from list is taken as shown in Fig. 3

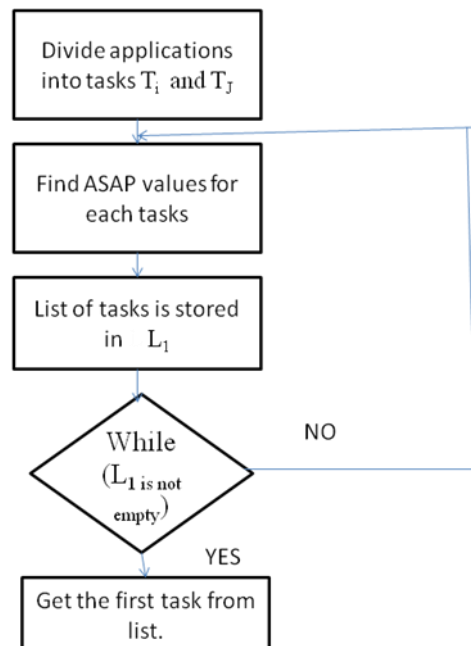


Fig. 3. Flowchart describing task allocation to List

For the tasks from list, elasticity and minimum execution time of each task is calculated. If elasticity of task 1 is minimum than task 2, then task 1 is assigned to processor as shown in Fig. 3.

### V. INTEGRATED APPROACH

Mostly works are treated task scheduling and memory partitioning as two decoupled steps that are performed by first scheduling the tasks onto processors and then partitioning memory among processors. The problems due to independent approach are solved in an integrated fashion. In the proposed technique the integrated approach is applied to lame benchmark applications to show the significant improvement of our integrated approach in pre-emptive scheduling algorithm. Consider the example in Fig. 2 of a task graph with six tasks  $t_1, t_2, t_3, t_4, t_5,$  and  $t_6$ . Task 4 depends on task1, task2, task3 and task 6 depends on tasks4 and task5. there is an edge between two tasks  $t_1$  and  $t_2$  means that a communication cost should be calculated. Calculating minimum, average and maximum and the computation time for task  $t_1$  on processor  $P_i$  and assuming all the variables of SPM between two tasks  $t_1$  and  $t_2$  i.e. communication cost should be calculated. The computation time for task  $t_1$  on processor  $P_i$  assumes all the variable SPM depends on task1, task2, task3 and task6 depends on tasks4 and task5. There is an edge between two tasks  $t_1$  and  $t_2$  i.e. a communication cost should be calculated.  $1/n$  of the available SPM budget is assigned to  $P_i$ . Where  $n$  is the number of processors, and no SPM is assigned to  $P_i$ . Task  $t_5$  will not be scheduled at this point based on its ALAP value. First tasks  $t_1$  and  $t_2$  will be mapped to the two available processors  $P_i$  and  $P_j$ . The scheduling algorithm will map  $t_3$  to  $P_j$  as it is free before  $P_i$  since the computation time of  $t_2$  is less than of  $t_1$ , in a similar fashion, the scheduling algorithm. Fig. 2 shows the results of the common practice of partitioning the available SPM memory equally between the two processors  $P_1$  and  $P_2$ . Equally partitioned SPM reduces the computation time of the whole application to 25.96s. Nonequal partitioned SPM reduces the computation time of the whole application to 20.589s. Our integrated approach reduces the computation time to 17.79s. The computation time is reduced by dividing the SPM in any ratio. From the scheduled task, task  $T_4$  can only start after  $P_2$  is done executing task  $T_3$ . The issue now is to try to reduce the dead time between tasks  $T_1$  and  $T_4$  imposed by the computation time for tasks  $T_2$  and  $T_3$ . Minimizing the dead time, by locating more SPM budget to processor  $P_2$  reduces the computation time of task  $T_2$  and  $T_3$ . Tasks to which SPM memory is allocated to processor  $P_2$  then the computation time for task 1 will jump to 17 and the results for the minimum start time of  $T_4$  will increase from 16 to 17. Increase can be avoided by allocating some SPM memory to  $P_1$ , so the execution time will be balanced to end of task  $T_3$ . The problem with the previous schedule is that it allocated  $T_3$  to the same processor  $P_2$  that is scheduled to execute  $T_2$ . The elasticity of a task is defined as the extent to which this task can benefit from a larger SPM. Elasticity is defined as the extent to which the computation cost of task on  $P_i$  may decrease as the SPM budget of  $P_i$  is increased from the current budget to size, where size is the maximum amount of SPM budget available in model. Equation (1) defines elasticity of task  $T_1$  where  $cur$  is the computation time of the task under the current memory budget. The elasticity of task  $T_1$  is a measure of the room for computation time reduction of  $T_1$  with more SPM budget.

$$Elasticity(T_i) = \frac{Cur - Min}{Cur} \tag{1}$$

A bigger value of elasticity means that the computation time of  $T_1$  is more amenable for reduction with the increase in the SPM allocated to that task. Elasticity is a number between 0 and 1 for uniformity.

### VI. EXPERIMENTAL RESULT

Our heuristic starts with profiling the application to extract important information. Using the profiling data, the embedded application will be divided into tasks with a necessary data communication between the two tasks imposing a certain kind of dependence. From Fig. 3, for each task  $T_i$  and processor  $P_j$ , we calculate the number of variables, the size of the variables, and Minimum values.

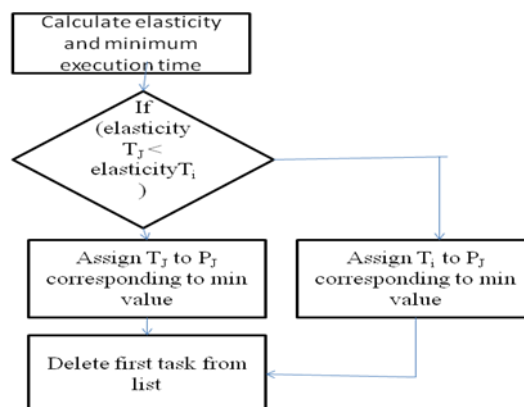


Fig. 4. Flowchart for calculating elasticity and minimum execution time for each task from list

All these values are computed based on profiling and the ASAP values for all tasks are calculated based on the average values. The minimum start time of a task  $T_i$  on processor  $P_j$ ,  $START\_TIME(T_i, P_j)$ , is equal to the maximum of the end time of processor  $P_j$ ,  $END\_TIME(P_j)$ , and the corresponding communication time. Two independent tasks mapped to the same processor will have zero communication cost. For lame benchmark, the overhead time provided that the predicted end computation time ( $PEC(P_j)$ ) of this processor is  $\beta\%$  less than that of  $P_j$ . Equation (2) represents the calculation of PEC. The PEC of a processor is clearly related to the elasticity of the tasks scheduled on that processor.

$$PEC(P_i) = END_{TIME}(P_i) - \sum_{T_i \in P_i} \left( Cur(T_i) \frac{Cur(T_i)}{1 + elasticity(T_i)} \right) \quad (2)$$

The operating system used for the execution of the five tasks is Linux operating system. The benchmark used is LAME benchmark. This type of benchmark is used for audio application. In Linux the operating system used is Fedora 12.0. It is a leading edge, free and open source operating system that continues to deliver innovative features to many users, with a new release about every six months. It helps to have optimized performance and faster updates. Three input files are considered for LAME benchmark. Three files are considered as three audio files of size 8 KB, 14 KB and 22 KB. The execution time, priority and elasticity for these input files are calculated. Scratch memory address lines are the ones not assigned by cache address lines.

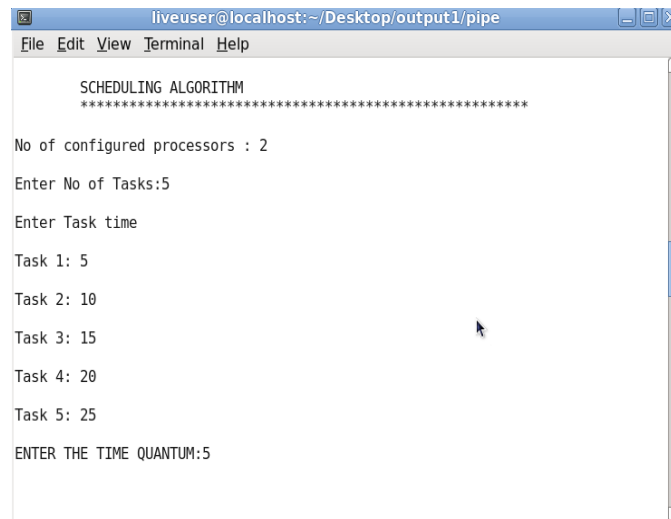


Fig. 5. Result of Round robin scheduling algorithm

From Fig. 5, five tasks are given the time period with number of configured processors mentioned. From the results remaining quantum values for each task is calculated.

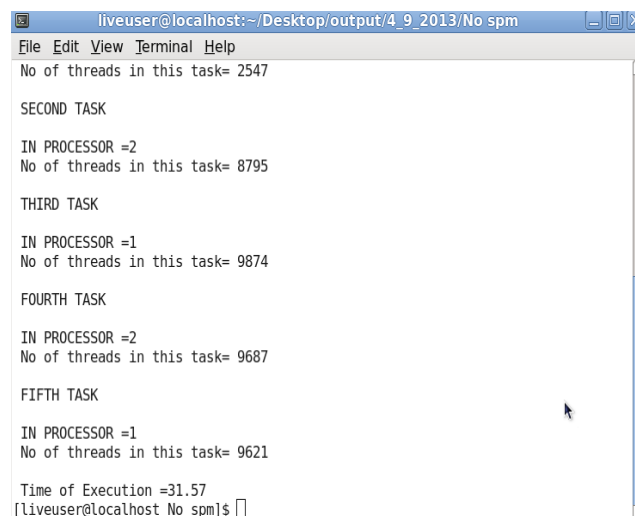


Fig. 6. Simulation result of calculating computation time with no scratch pad memory

Fig. 6. shows the simulation result of calculating computation time with no scratch pad memory .This shows the usage of hardware controlled memory, cache memory when used in embedded application, executes the computation at 31.57s. It is high when compared to the usage of software controlled memory. Elasticity value is 0.96832. Based on this elasticity execution time is reduced.

```

liveuser@localhost:~/Desktop/output1/equal
File Edit View Terminal Help
[liveuser@localhost ~]$ cd Desktop
[liveuser@localhost Desktop]$ ls
liveinst.desktop output1 output first
[liveuser@localhost Desktop]$ cd output first
bash: cd: output: No such file or directory
[liveuser@localhost Desktop]$ cd output
bash: cd: output: No such file or directory
[liveuser@localhost Desktop]$ ls
liveinst.desktop output1 output first
[liveuser@localhost Desktop]$ cd output1
[liveuser@localhost output1]$ ls
equal ILP integrated lame-simulation non equal pipe
[liveuser@localhost output1]$ cd equal/
[liveuser@localhost equal]$ ls
equal equal.c equal.c~ Makefile round.c
[liveuser@localhost equal]$ ./equal
Task Scheduling and Memory partitioning with equal partitioning
Generation of task

No.of processors available =2
FIRST TASK

IN PROCESSOR =1
Scratch pad Number=46433
    
```

Fig. 7. Simulation result of calculating computation time with equal scratch pad memory

Fig. 7, shows the simulation result of calculating computation time with equal partition of scratch pad memory. This shows the usage of software controlled memory, scratch pad memory when used in embedded application, executes the computation at 25.96s. Elasticity value is 0.9614.

```

liveuser@localhost:~/Desktop/output1/non equal
File Edit View Terminal Help

IN PROCESSOR =1
Scratch pad Number=65001
spm size = 200
Time of Execution =25.96[liveuser@localhost equal]$ cd ..
[liveuser@localhost output1]$ ls
equal ILP integrated lame-simulation non equal pipe
[liveuser@localhost output1]$ cd non\ equal/
[liveuser@localhost non equal]$ ls
Makefile nonequal non equal.c round.c
[liveuser@localhost non equal]$ ./nonequal
Task Scheduling and Memory partitioning with non equal partitioning
Generation of task

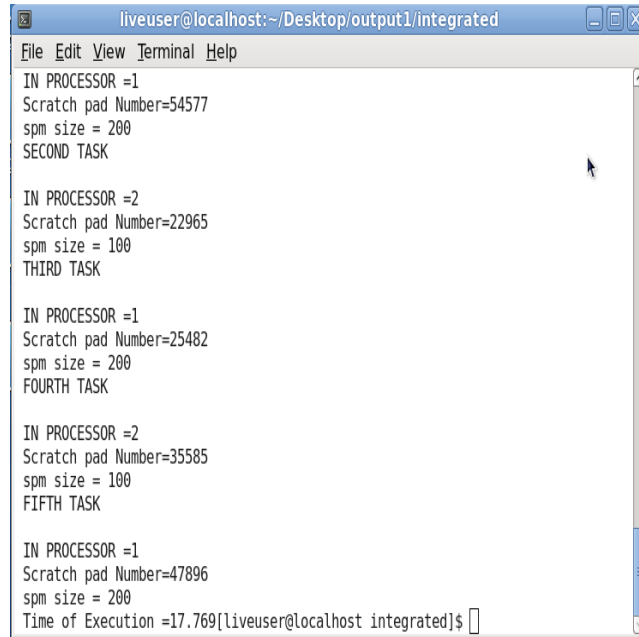
No.of processors available =2
FIRST TASK

IN PROCESSOR =1
Scratch pad Number=45789
spm size = 200
SECOND TASK

IN PROCESSOR =2
Scratch pad Number=42365
    
```

Fig. 8. Simulation result of calculating computation time with nonequal scratch pad memory

Fig. 8 shows the simulation result of calculating computation time with non equal partition of scratch pad memory .This shows the usage of software controlled memory, scratch pad memory when used in embedded application, executes the computation at 20.589s. Elasticity value is 0.9514.



```

liveuser@localhost:~/Desktop/output1/integrated
File Edit View Terminal Help
IN PROCESSOR =1
Scratch pad Number=54577
spm size = 200
SECOND TASK

IN PROCESSOR =2
Scratch pad Number=22965
spm size = 100
THIRD TASK

IN PROCESSOR =1
Scratch pad Number=25482
spm size = 200
FOURTH TASK

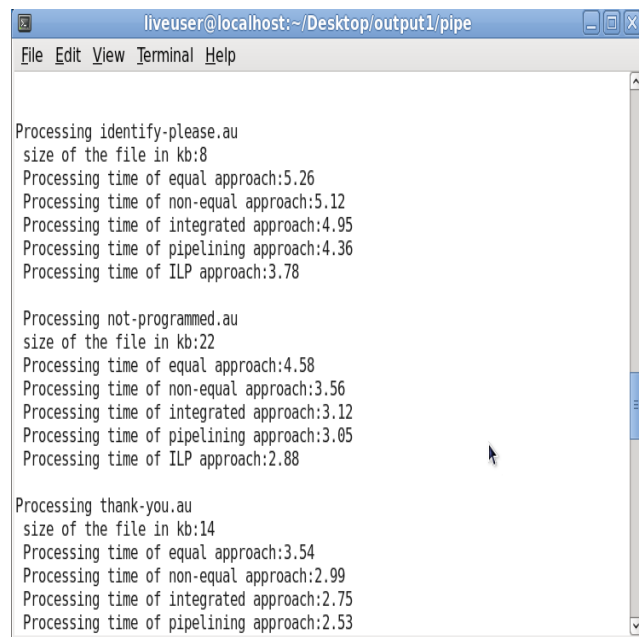
IN PROCESSOR =2
Scratch pad Number=35585
spm size = 100
FIFTH TASK

IN PROCESSOR =1
Scratch pad Number=47896
spm size = 200
Time of Execution =17.769[liveuser@localhost integrated]$

```

Fig. 9. Simulation result of calculating computation time with integrated approach

Fig. 9 shows the simulation result of calculating computation time with integrated approach of effective memory partitioning and scheduling. This shows the usage of software controlled memory, scratch pad memory when used in embedded application, executes the computation at 17.769s. Elasticity value is 0.9437.



```

liveuser@localhost:~/Desktop/output1/pipe
File Edit View Terminal Help

Processing identify-please.au
size of the file in kb:8
Processing time of equal approach:5.26
Processing time of non-equal approach:5.12
Processing time of integrated approach:4.95
Processing time of pipelining approach:4.36
Processing time of ILP approach:3.78

Processing not-programmed.au
size of the file in kb:22
Processing time of equal approach:4.58
Processing time of non-equal approach:3.56
Processing time of integrated approach:3.12
Processing time of pipelining approach:3.05
Processing time of ILP approach:2.88

Processing thank-you.au
size of the file in kb:14
Processing time of equal approach:3.54
Processing time of non-equal approach:2.99
Processing time of integrated approach:2.75
Processing time of pipelining approach:2.53

```

Fig. 10. Processing audio files of 8kb, 14kb and 22kb with lame benchmarks

Fig. 10 shows the simulation result of calculating computation time processing audio files. Three audio files of 8kb, 22kb and 14kb are processed. 8kb executes for five approaches with minimum execution time. ILP (Integer linear programming) takes less computation time when compared to other approach.

**Table I: CPU burst time of process by round robin algorithm**

S.NO	PROCESS NAME	CPU BURST TIME(IN Secs)	PRIORITY	ELASTICITY
1	NON EQUAL	25.96	5	0.96832
2	EQUAL	20.589	4	0.9614
3	INTEGRATED APPROACH	17.79	3	0.9514
4	PIPELINING	17.10	2	0.9437
5	ILP	16.94	1	0.940

Above table explains the CPU burst time for execution of each approach. The approach included are non equal partitioning, equal partitioning of SPM memory, integrated approach with effective pipelining and memory partitioning, integer linear programming.

## VII. CONCLUSION

In this paper, we presented an effective heuristic that integrates effective task scheduling and memory partitioning with varying ratios of embedded applications on multiprocessor system on chip with scratch pad memory. Although this project has addressed some problem, there are some issues related to what has been done and are still opened to be solved. The analysis in this project considers that priorities of the tasks are assigned before performing the analysis. However, a non covered issue in this project is to find an optimal priority assignment for the tasks in the task dependence graph. Our effective integrated approach overcome the disadvantage of widely used decoupled approach and significantly improves the results since the appropriate partitioning of SPM spaces among different processors depends on the tasks scheduled on each of those processors

## REFERENCES

- [1] L. Benini, D. Bertozzi, A. Guerri, and M. Milano, "Allocation and scheduling for MPSOC via decomposition and no-good generation," in *Proc. IJCAI*, 2005, pp. 107–121.
- [2] Y.-K. Kwok and I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms," *J. Parallel Distributed Comput.*, vol. 59, no. 3, pp. 381–422, Dec. 1999.
- [3] Y. Cho, N.-E. Zergainoh, S. Yoo, A. Jerraya, and K. Choi, "Scheduling with accurate communication delay model and scheduler implementation for multiprocessor system-on-chip," *Des. Automat. Embedded Syst.*, vol. 11, nos. 2–3, pp. 167–191, 2007.
- [4] P. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 5, no. 3, pp. 682–704, Jul. 2000.
- [5] O. Avissar, R. Barua, and D. Stewart, "An optimal memory allocation scheme for scratch-pad-based embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 1, no. 1, pp. 6–26, Nov. 2002.
- [6] A. Dominguez, S. Udayakumar, and R. Barua, "Heap data allocation to scratch-pad memory in embedded systems," *J. Embedded Comput.*, vol. 1, no. 4, pp. 521–540, Dec. 2005.
- [7] G. D. Micheli, R. Ernst, and W. Wolf, *Readings in Hardware/Software Co-Design*. San Francisco, CA: Morgan Kaufmann, 2002.
- [8] S.-R. Kuang, C.-Y. Chen, and R.-Z. Liao, "Partitioning and pipelined scheduling of embedded systems using integer linear programming," in *Proc. ICPADS*, 2005, pp. 37–41. M. Young, *The Technical Writer's Handbook*. Mill Valley, CA: University Science, 1989.
- [9] P. Panda, N. Dutt, and A. Nicolau, *Memory Issues in Embedded System-on-Chip: Optimization and Exploration*. Dordrecht, The Netherlands: Kluwer, 1999.
- [10] Z. Ma, C. Wong, S. Himpe, E. Delfosse, F. Catthoor, J. Vounckx, and G. Deconinck, "Task concurrency analysis and exploration of visual texture decoder on a heterogeneous platform," in *Proc. IEEE Workshop Signal Process. Syst.*, Aug. 2003, pp. 245–250.
- [11] R. Neimann and P. Marwedel, "Hardware/software partitioning using integer programming," in *Proc. DATE*, 1996, pp. 473–480. K. Elissa, "Title of paper if known," unpublished.
- [12] J. Sjodin and C. V. Platen, "Storage allocation for embedded processors," in *Proc. Int. Conf. CASES*, Nov. 2001, pp. 15–23.
- [13] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction," in *Proc. DATE*, 2002, pp. 409–415.
- [14] F. Angiolini, L. Benini, and A. Caprara, "Polynomial-time algorithm for on-chip scratchpad memory partitioning," in *Proc. Int. Conf. CASES*, 2003, pp. 318–326. M.
- [15] M. Kandemir, J. Ramanujam, and A. Choudhury, "Exploiting shared scratch pad memory space in embedded multiprocessor systems," in *Proc. DAC*, 2002, pp. 219–224.
- [16] O. Ozturk and M. Kandemir, "An integer linear programming based approach to simultaneous memory space partitioning and data allocation for chip multiprocessors," in *Proc. IEEE ISVLSI*, Mar. 2006, p. 6.
- [17] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated scratchpad memory optimization and task scheduling for MPSOC architecture," in *Proc. Int. Conf. CASES*, 2006, pp. 401–410.
- [18] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. IEEE Int. Symp. Microarchitecture*, Dec. 1997, pp. 330–335.
- [19] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE 4th Annu. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.