Abstract: The mounting trend in current complex embedded systems is to position a multiprocessor system-on-chip. A MPSoC consists of multiple heterogeneous marching elements, a memory pecking order, and input/output items which are linked together by an on-chip interconnect structure. Such architecture provides the suppleness to meet the presentation requirements of multimedia applications while respecting the restraints on memory, price, sizing, clock time, and power. Many embedded systems employ software-managed memories known as scratch-pad memories. Different caches, SPMs are software controlled and hence the execution time of applications on such systems can be correctly predicted. Schedule the task of an embedded applications on the processors and partitioning the available SPM resources among these processors are two critical issues in such systems. Often, these are considered separately, such a decoupled approach may overlook better quality schedules. In this paper, we proposed an integrated approach to task scheduling and SPM partitioning to further reduce the execution time of embedded applications by using preemptive scheduling algorithm. Results on several real-life benchmarks show the significant improvement from our proposed technique.

Keywords: Memory partitioning, multiprocessor system-on-chip (MPSoC), scratch pad memory, preemptive scheduling (PS), task scheduling,
1. Introduction:

A Multiprocessor system–on-chip (MPSoC) is a system-on-a-chip (SoC) which uses multiple processors usually under fire for embedded applications. It is used by platform contain multiple heterogeneous processing elements, a memory hierarchy and I/O components. All these constituents are related to each other by on-chip interconnect. These architectures meet the operation needs of multi-media application and telecommunication architecture. As embedded systems become increasingly building complex, the increase in memory admittance speed has betrayed to keep up with the increase in processor speed. This the memory access response time a major issue in scheduling embedded applications on embedded systems. A MPSoC is an effective solution to the increasing complexity and size of embedded applications. Execution time predictability a vital issue for real-time embedded applications. MPSoC systems use software-controlled memories known as scratchpad memories (SPMs), which allow execution times to be predicted accurately Unfortunately, SPMs are expensive and hence they are commonly of limited size. The tasks can be scheduled on different processors and then memory is partitioned for each task. The computing time of each task looks on the amount of SPM apportioned to the processor executing this task. The problem of task scheduling and memory allocation on MPSoC is an NP-complete problem. Traditionally, these two steps are per-formed separately where tasks are usually scheduled first and the SPM budget is then partitioned among the processors. Such a decoupled approach may prevent reducing the computation time of the whole application.

The remainder of this paper is organized as Follows. Section II presents the related effort of this paper. Section III is problem formulation. Section IV briefly describes the Methodology Section V Task & TDG. Section VI presents task scheduling and memory partitioning Section VII describes the preemptive scheduling. Section VIII presents Experimental results and Section IX draws the conclusion for this paper.

2. Related Effort:

Various research groups have studied the Problem of task scheduling of applications on multiple processors where the objective is to Minimize the execution time. O. Ozturk and M.Kandemir [2] addressed the problem of decomposing (partitioning) on-chip memory space across parallel processors and allocating data across memory components an integrate
manner. The problem of integrated memory space partitioning and data allocation is addressed for chip multiprocessors and is achieved by two components: an optimizing compiler and an ILP (integer linear programming). M. Kandemir, J. Ramanujam, and A. Choudhury, [3] presented a compiler strategy to optimize data accesses in regular array-intensive applications running on embedded multiprocessor environments.


R. Neimann and P. P. Marwedel [7] described a new approach to HW/SW partitioning using integer programming (IP). The partitioning approach described in this paper works fully and supports multi-processor systems, interfacing and hardware sharing. V. Suhendra, C. Raghavan, and T. Mitra [6] proposed, an integrated task mapping, scheduling and SPM partitioning, and data allocation based on Integer Linear Programming (ILP) formulation. An application-specific flexible partitioning of the on-chip SPM work out among the processors is critical for performance. Moreover, scheduling the tasks of an application on to the processors and partitioning the SPM are inter-dependent even Though these steps are decoupled in the traditional design space exploration process. Hassan Salamy and J. Ramanujam [1] presented an integrated approach for task scheduling and memory partitioning to reduce the execution time.

3. Problem Formulation:

Architectural Model: we focus on embedded single chip multiprocessor architecture as shown in Figure 1. The architecture contains multi-cores on chip and this cores can be homogeneous or heterogeneous. The processor cores communicate with the sharing off chip memory via bus. The architecture use scratchpad memory, which is fast SRAM managed by software. In single-chip multiprocessor setting each processor core can access its private SPM as well as SPMs of other processors. Such a SPM is known as virtually shared scratchpad memory or VS-SPM.

3.1. Problem Identification:

Given an embedded application consisting of tasks, an MPSoC architectural model, and an SPM budget, the problem statement is described below.
i. Schedule the tasks on the available processors.
ii. Partition the SPM memory among the processor.
iii. Assign the schedule the task with allocated memory to each processor. find the execution time of the tasks.

![Figure 1: MPSoC with virtually shared SPM](image)

The main aim is to reduce the execution time in cycles of the embedded application on the MPSoC architectural model.

4. Methodology:

The embedded application is given to the MPSoC that consists of multiple processors. The application is then divided into number of tasks. These tasks are scheduled and the memory should be partitioned among the processors. And these are implemented in preemptive algorithm and find execution time.
5. Tasks & TDG:

Embedded applications usually belong of Computation blocks, which are handled as tasks. An application programme is divided into tasks. Tasks are the various operations in the application. Whenever a program is executed the operating system develop a new task for it. The task is like an envelope for program. The state information of a task is represented by the task states such as idle, running, ready and blocked states. There are usually dependences between tasks that should be observed in the schedule. The problem formulation is based on a task dependence graph (TDG). A TDG is a directed acyclic graph with weighted edges where each vertex represents a task in the embedded application.

Figure 2: Block diagram of proposed Technique
In there are six tasks are obtained. Task T4 depends on tasks T1, T2 and T3, and task T6 depends on tasks T4 and T5. Any time there is an edge between two tasks Ti and Tj means that a provided that these two tasks are allocated to two different processors. Tasks T1, T2, T3, and T5 are ready to be scheduled in our example. Task T5 will not be scheduled at this point based on its communication cost should be accounted for ALAP value. Thus, first tasks T1 and T2 will be mapped to the two available processors P1 and P2. Fig.2. An example TDG The scheduling algorithm will map T3 to P2 as it is free before P1 Consider the example task graph shown below with six tasks, T1, T2, T3, T4, T5, and since the computation time of T2 is less than that of T1. In a similar fashion, the scheduling algorithm will assign tasks T4 and T6 to processor P1 whereas task T5 will be mapped to processor P2. From the task schedule, it has seen that task T4 can only start after P2 is done executing task T3. The issue now is to try to reduce the dead time between tasks T1 and T4 imposed by the computation time for tasks T2 and T3. To minimize this dead time, techniques usually allocate more SPM budget to processor P2 to reduce the computation time of tasks T2 and T3.

6. Task Scheduling And Scratch Pad Memory Partitioning:

Five approaches can be implemented to solve the task scheduling and memory allocation problem on MPSoC systems, namely:
6.1. Decoupled TSMP On Equal Partitioned SPM:

The results are partitioning the available SPM memory equally between the two processors. With such a criterion, the available SPM budget will be equally divided between processors P1 and P2 regardless of what tasks are mapped to what processors. Equally partitioned SPM reduces the computation time of the whole application.

![Figure 4: Schedule on equal partitioning memory](image)

6.2. Decoupled TSMP On Non Equal Partitioned SPM:

To further reduce this application’s computation time, the available SPM can be divided between the two processors in any ratio. From the task schedule, we can see that task T4 can only start after P2 is done executing task T3. The issue now is to try to reduce the dead time between tasks T1 and T4 imposed by the computation time for tasks T2 and T3. To minimize this dead time, techniques usually allocate more SPM budget to processor P2 to reduce the computation time of tasks T2 and T3.

![Figure 5: Schedule on non-equal partitioning Memory](image)

6.3. Integrated TSMP:

The problem with the previous schedule is that it allocated T3 to the same processor P2 that is scheduled to execute T2.
This choice is the reason for the dead time in the schedule as T2 cannot benefit much from more SPM memory. A good heuristic should take these values into consideration where a better choice for T3 is to be scheduled on P1 with all available SPM memory being allocated to this processor, and the result is a schedule with the minimal end time.

6.4. Integration Of TSMP With:

6.4.1. Pipeling:

Pipeline scheduling allows tasks of different embedded application instances to be scheduled at each stage of the pipeline. Such a schedule does not necessarily decrease the computation time of one instance of embedded application, but rather it decreases the time between the start times of two consecutive iteration of task graph. Here the pipelined concept is implemented by storing the result of previous task in to the memory while current task is executing. This further reduces the computation time.

6.4.2. Pipeline Scheduling:

Most streaming applications such as multimedia and Digital signal processing applications are iterative in nature. For these applications, the execution of the graph is evoked repeatedly for a stream of input data Hence these applications are amenable to pipeline. Implementation benefits from allowing multiple processors execute multiple iterations of the task graph at the same time. The objective for sequential implementation is to minimize the execution time of the single iteration of the task graph. Pipeline scheduling benefits from allowing tasks from different embedded application instances to be scheduled at each stage of the pipeline. Such a schedule does not necessarily decrease the computation time of one instance of embedded application, but rather it decreases the time between the start times of two consecutive iterations of the task graph. The objective is to decrease the pipeline stage time interval, as after filling up the pipeline an instance execution of the application is performed each
pipeline stage. The maximum number of stages is equally to the number of processors in the MPSoC system.

6.5. Integer Linear Programming (ILP) Formulation:

The optimal solution with pipelining is based on the ILP formulation to further reduce the execution time. We first formulate the scheduling of tasks on multiple processors. This formulation is then extend to handle the pipeline scheduling. Finally, we formulate SPM partitioning and data allocation and integrate it with the formulation of task scheduling.

The objective of ILP is to minimize the critical path through the task graph. That is, the aim is to minimize the completion or end time of the last task.

7. Preemptive Scheduling:

This is preemptive in priority based scheduling algorithm. This scheduling algorithm it comes under a event driven algorithm in RTOS. In this scheduling Each system processor if given a priority, then the scheduling has done according to the priority of each process. A high priority job can get CPU whereas lower priority job has to wait and Priority scheduling is necessarily a form of preemptive scheduling where priority is the basic of preemption. In most real-time kernels use preemptive priority-based scheduling by default. In this figure .7 with this type of scheduling, the task that get to run in any point is the task with the highest priority among all other tasks ready to run in the system.

![Figure 7: Preemptive priority-based scheduling.](image)
Real-time kernels generally support 256 priority levels, in which 0 the highest and 255 the lowest. In some kernels appoint the priorities in reverse order, where 255 the highest and 0 the lowest. Regardless the concepts are basically are equal. With a preemptive priority based scheduler each task has a priority and the highest priority task runs first. If a task with priority higher than the current task become ready to run and the kernel immediately saves the current task’s context in its TCB and switches to the higher-priority task. In fig 7. task 1 is preempted by higher-priority task 2 which is then preempted by task 3. When task 3 complete, task 2 resumes; similarwise, when task 2 completes, task 1 resumes. Although tasks are assign a priority when they are created, a tasks priority can be changed dynamically using kernel given calls. The ability to change task priorities dynamically allow an embedded application the flexibility to adjust to external event as they obtained, creating a correct real-time, responsive system. Note however that abuse of this capability can lead to priority inversion, deadlock and eventual system failure.

8. Experimental Result:

We implemented five approaches to solve the task scheduling and scratch pad memory partitioning for on MPSoC systems, namely:

i. Decoupled task scheduling and scratch pad memory with equal partitioning.
ii. Decoupled task scheduling and scratch pad memory with non-equal partitioning.
iii. Task scheduling and scratch pad memory both are integrated with partitioning.
iv. Integration with pipelining TSMP−PIPE.
v. The effective solution with pipelining based on the ILP formulation ILP−PIPE.

We used the Lame benchmark (MP3 Player) real-life programs from the Mediabench and MiBench, as test benchmarks.

8.1. Experimental Setup In Hardware Implementation:

In hardware using ARM7Processor in LPC2148 Controller using IAR workbench tool suite.

This ARM (Advanced RISC Machine) is 16/32 bit controller. The LPC2148 microcontroller are based on a 32/16 bit ARM7TDMI-S CPU with real-time emulation and embedded trace support that combines the microcontroller with embedded high speed flash memory range from 32 kB to 512 kB. In 128-bit wide memory interface and unique accelerator architecture.
enable 32-bit code execution at the maximum clock rate. In critical code size applications the alternative 16-bit Thumb mode reduce code.

In this lame benchmark has choosing a three audio files .In according to their file SPM size like 8kb, 14kb, 22kb has to be dumped in controller and audio file has work in priority scheduling based that should be dumped in controller using IAR Workbench tool suite using mucos II kernel and finally execute the time cycle period in the LCD display.

In fig.8.shows the hardware setup of lame benchmark in ARM7Processor, it Can show the execution time period in LCD display. If we want implement in software means by using LINUX OS to simulate in Simple scalar architectural simulator .this Simple scalar can simulate the execution of an application on complex MPSoC architectures with various memory hierarchies. In the simple scalar profilerand we extract the profile data such as variable sizes and access frequencies and execution time of each task.

In table. 1 can compare the results of five approaches in lame benchmark application of execution time.

We analysis the benchmark under three SPM computes chosen based on the size of the benchmark. The option of SPM sizes for each benchmark is essential as too little SPM or too much SPM for a certain embedded application may not reflect the effectiveness of our heuristic. The off-chip memory size is assumed to be unlimited that is it can hold all the data variables needed by the embedded application.

Our integrated approach for task scheduling and memory partitioning, TSMP–INTEG, further well growing the results over the decoupled approach, TSMP–ANY. TSMP–INTEG improved over TSMP–ANY from small improvement near to dramatic improvement. This
betterment is due to the guidance that our integrated approach uses to partition the memory based on the fact that the SPM configuration of a certain processor depends on the tasks mapped to that processor.

<table>
<thead>
<tr>
<th>TSMP Based on</th>
<th>Execution time for (ms)</th>
<th>Memory Allocation (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>8kb</td>
</tr>
<tr>
<td>equal</td>
<td>5.58</td>
<td>4.54</td>
</tr>
<tr>
<td>non-equal</td>
<td>5.19</td>
<td>3.50</td>
</tr>
<tr>
<td>integrated</td>
<td>4.90</td>
<td>3.09</td>
</tr>
<tr>
<td>Integrated Approach</td>
<td>4.20</td>
<td>2.98</td>
</tr>
<tr>
<td>With Pipelining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelining based on</td>
<td>3.89</td>
<td>2.88</td>
</tr>
<tr>
<td>ILP Formulation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 1: Execution time period of lame benchmark*

The pipelining results stress the fact that such embedded applications can merits significantly from pipelining. The pipeline cost is the computation time needed for one pipeline stage. As expected, our embedded applications greatly advantage from pipelining as the execution time is reduce on average compared to TSMP–INTEG. In order to show the effectiveness of our task scheduling and memory partitioning heuristic, INTEG, we compared it to an optimal
integer linear formulation (ILP) based on the ILP formulation of this problem ILP-PIPE. This ILP-PIPE is a better solution compared to other four approaches.

We tested our heuristic on the lame Benchmark in preemptive scheduling based on fig. 8, which shows the results achieved by our heuristic when considering a system with two processors and an SPM budget ranging from 8K to 128k memory allocation. A real-time lame benchmark was executed in the execution time.

![Comparison Results of Lame benchmark](image)

**Figure 9: comparison Results of Lame benchmark**

9. **Conclusion:**

An effective approach was presented that integrates task scheduling and scratch pad memory partitioning of embedded applications on multiprocessor systems-on-chip based preemptive scheduling algorithm. Compared to the widely-used decoupled approach, this integrated approach using preemptive scheduling algorithm significantly improved the results, since the appropriate partitioning of SPM spaces among different processors depends on the tasks scheduled on each of those processors and vice versa. Thus, the reduction in the execution time of the tasks scheduled on the processors is obtained using various approaches such as equally partitioned SPM, non-equal partitioned SPM, integrated approach, and integrated approach with pipelining, ILP with pipelining.
References:


