ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

An Efficient Heuristic to Task Scheduling and Memory Partitioning for Multiprocessor System-on-Chip

Maria Glenny.P, Suganyadevi.K

Abstract*— MPSoC architecture uses multiple processors which provide the flexibility to meet the performance needs of multimedia applications. Almost all embedded systems employ software-managed memories known as scratch-pad memories (SPM). SPMs are software controlled and hence the execution time of applications can be accurately predicted. Scheduling the tasks and memory partitioning are two different portions in an embedded application. Scheduling the tasks and memory partitioning are two different portions in an embedded application. Such a decoupled approach increases the execution time. In this paper we present an integrated approach to task scheduling and SPM partitioning to further reduce the execution time of embedded applications by round robin scheduling. This proposed technique is applied to real-time benchmark applications to show the significant improvement of our integrated approach. Simple scalar simulator is used to show some of the profiling data's.

Index Terms— **Memory partitioning, Multiprocessor system-on- chip , Round robin scheduling ,Scratchpad memory, Simple Scalar Simulator, Task scheduling**.

I. INTRODUCTION

A Multiprocessor System-on-Chip (MPSoC) is a system-on-a-chip (SoC) which uses multiple processors usually targeted for embedded applications. MPSoC contain multiple heterogeneous processing elements, a memory hierarchy and I/O components and all these components are linked to each other by interconnect. The performance needs of multimedia applications and telecommunication are met by these architectures.

Nowadays, the increase in memory access speed has failed to keep up with the increase in processor speed. Hence the memory access latency a major issue in scheduling embedded applications on embedded systems. An MPSoC is an attractive solution to the increasing complexity and size of embedded applications. Execution time predictability is a critical 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 usually of limited size.

The tasks can be scheduled on different processors and then memory is partitioned for each task. The computation time of each task depends on the amount of SPM allocated to the processor executing this task. The problem of task scheduling and memory allocation on MPSoC's 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 technique may prevent reducing the computation time of the whole application. These two steps can be integrated to improve the performance.

The remainder of this paper is organized as follows. Section II presents the related work of this paper. Section III briefly describes the problem formulation .Section IV gives the motivation for this paper. Section V presents the integrated task scheduling and memory partitioning and Section VI describes the round robin scheduling. Section VII presents Experimental results and Section VIII draws the conclusion for this paper.

II. RELATED WORK

Many 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 [17] addressed the problem of decomposing (partitioning) on-chip memory space across parallel processors and allocating data across memory components in an integrated 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 [16] presented a compiler strategy to

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

optimize data accesses in regular array-intensive applications running on embedded multiprocessor environments. S.-R. Kuang, C.-Y. Chen and R.-Z. Liao [7] presented "An integer linear programming (ILP)" based approach for integrated hardware/software (HW/SW) partitioning and pipelined scheduling of embedded systems for multimedia applications. The objective of this is to minimize the total component cost and the number of pipeline stages subject to the throughput constraint on the pipelined architecture. R. Neimann and P. P. Marwedel [5] described a new approach to hardware/software partitioning using integer programming (IP). The partitioning approach described in this paper supports interfacing and hardware sharing and multi-processor systems.

V. Suhendra, C. Raghavan, and T. Mitra [18] proposed, an integrated task scheduling, mapping, SPM partitioning, and data allocation based on Integer Linear Programming (ILP) formulation. For performance optimization, An application-specific flexible partitioning of the on-chip SPM budget among the processors is critical. Moreover, SPM partitioning and scheduling the tasks of an application on to the processors and 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.

III. PROBLEM FORMULATION

A. ARCHITECTURAL MODEL

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

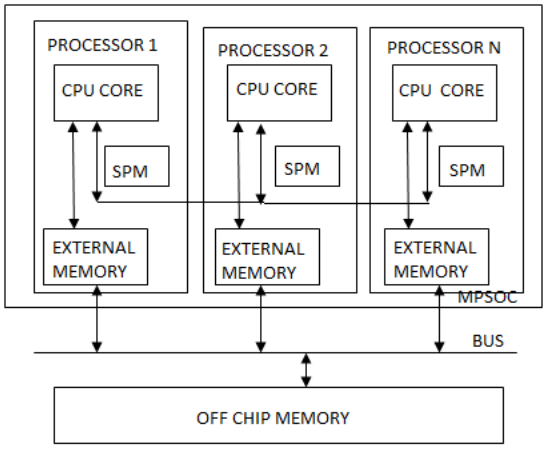

Fig I: MPSoC with virtually shared SPM

B. TASK DEPENDENCE GRAPH

Embedded applications usually consist of computation blocks, which are treated as tasks. There are usually dependences between tasks (blocks) that should be respected in the schedule. Our problem formulation is based on a task dependence graph (TDG).

A TDG is a directed acyclic graph which represents the key computation blocks (tasks) of an application nodes and the communication between the tasks as edges. A task can be mapped to any of the processing cores. Therefore associated with each tasks T are the execution times corresponding to running the task T on each of the processing cores.

In case of homogeneous cores there will be only one execution time associated with each task. An edge from task T to T' represents the data transfer between the tasks. . Since the processors in our architectural model can be heterogeneous, the execution time of each task depends on the processor to which this task is mapped as well as the SPM memory allocated to that processor .A TDG is shown in Figure. II.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

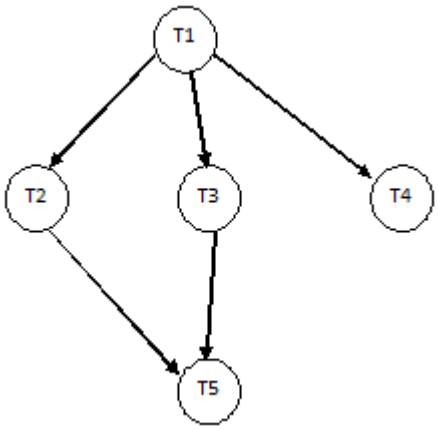

Fig II: Task Dependence Graph (TDG)

C. PROBLEM DEFINITION

Given an embedded application consisting of *t* tasks, an MPSoC architectural model, and an SPM budget, the problem statement is described below.

- *Scheduling*: Schedule the e tasks on the available processors.
- *Partitioning:* Partition the SPM memory among the processors.
- *Assigning:* Assign the scheduled tasks with allocated memory to each processor.
- Find the execution time of the tasks.

The goal is to minimize the execution time in cycles of the embedded application on the MPSoC architectural model.

D. MOTIVATION

Most works so far have treated task scheduling and memory partitioning as two decoupled steps that are performed by first scheduling the tasks onto processors and then partitioning the available memory among processors. Unfortunately, task scheduling and memory partitioning are interdependent and they should be integrated in one step in order to get high quality schedules.

Unlike current approaches that treat task scheduling and memory partitioning as two separate problems, we solve these two problems in an integrated fashion. This paper deals with developing effective heuristic for the task scheduling and memory partitioning problem for a multiprocessor system- on-chip where a single application is using the MPSoC at a time. These two steps are performed in an integrated fashion where the private on-chip memory budget allocated to a processor is decided as tasks are mapped to this processor.

The computation time of a task depends on the processor to which it is mapped, as well as on the SPM memory available for that task. Therefore, task scheduling should take into consideration the varying computation time of a task based on the processor and the SPM budget.

E. TASK SCHEDULING AND MEMORY PARTITIONING

A good heuristic for task scheduling and memory partitioning should take into consideration the varying execution time of a task throughout the process of building the schedule.

Consider the example in Figure. II of a task graph with six tasks, T_1, T_2, T_3, T_4, and T_5, Task T_2, T_3, depends on tasks T_1, and T_5, depends on tasks T_2, and T_3. The edge between two tasks T_i and T_j indicate that a communication cost should be accounted for provided that these two tasks are allocated to two different processors.

In this paper, the tasks can be scheduling with no scratch pad memories (SPM).The execution time is more when there is no SPM. SPM budget will be equally divided between processors P_1 and P_2 regardless of what tasks are mapped to what processors and this equally partitioned SPM reduces the computation time of the whole application.

The available SPM can be divided between the two processors in any ratio to further reduce this application's computation time. Such decoupled approaches increase the execution time. Our heuristic can reduce the computation time as it integrates task scheduling and memory allocation into one step.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

Using profiling of the tasks in the embedded application, Min, Avg, and Max values are calculated for each task on each of the available heterogeneous processors.

We define elasticity of a task as the extent to which this task can benefit from a larger SPM. In other words, we define elasticity as the extent to which the computation cost of a 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 our model. Elasticity of a task with respect to current and minimum computation time is given by,

$$\text{Elasticity}=\text{Cur}_i\text{-Min}_i/\text{Cur}_i \quad (1)$$

The Processor End Compilation (PEC) of a processor is related to the elasticity of the tasks scheduled on that processor. The PEC value provides the flexible essence of our heuristic as at each step the heuristic looks beyond the current SPM budgets distribution in its task mapping decision to an estimate of future distributions in future steps. In our heuristic, if assigning task Ti to either of two different processors will have the same increase value in the overall schedule time, we schedule Ti on the processor with the higher elasticity under the current SPM budget. The elasticity of a processor is the average value of the elasticity of the tasks scheduled on this processor.

$$\text{PEC}(P_k)=\text{End-time}(P_k)-\sum_{T_j \in P_k} [\text{Cur}(T_j)-(\text{Cur}(T_j)-1+\text{elasticity}(T_j)] \quad (2)$$

F. PIPELINED SCHEDULING

Most streaming applications such as multimedia and Digital signal processing applications are iterative in nature. The execution of the graph is evoked repeatedly for a stream of input data for these applications. Hence these applications are amenable to pipelined implementation. The pipelined implementation benefits from allowing multiple processors execute multiple iterations of the task graph at the same time. The goal of sequential implementation is to reduce 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. This type of scheduling decreases the time between the start times of two consecutive iterations of the task graph. The number of processors in the MPSoC system is equal to the maximum number of stages.

G. 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 extended 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 objective is to minimize the completion or end time of the last task.

H. ROUND ROBIN SCHEDULING

Round robin scheduling is a pre-emptive version of first-served scheduling. Each process in round robin scheduling is allowed to run for only a limited amount of time and this time interval is known as a time-slice or quantum.

The process is pre-empted if a process does not complete or get blocked because of an I/O operation within the time slice. In such a case, the time slice expires and. process gets pre-empted. This process which is pre-empted is placed at the back of the ready list where it must wait for the processes that were already on the list to cycle through the CPU. Figure.III shows the round robin scheduling stages.

Each task in a processor gets a brief chance to run if the quantum is short because it allows many processes to circulate through the processor quickly. By doing so, the iterative performance can be improved.

Context switch per overhead can be expressed as:

$$\textit{context switch overhead} = C / (Q+C) \quad (3)$$

Where Q is the length of the time-slice and C is the context switch time. Efficiency is increased when there is an increase in Q.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

I. ROUND ROBIN SCHEDULING ALGORITHM

- Start the process and Declare the array size
- After declaration, get the value and determine the number of elements to be inserted.
- With the elements inserted Set the time sharing system (quantum) with pre-emption
- The quantum is defined from 10 to 100ms
- Declare the queue as a circular and Make the CPU scheduler goes around the ready queue allocating CPU to each process.

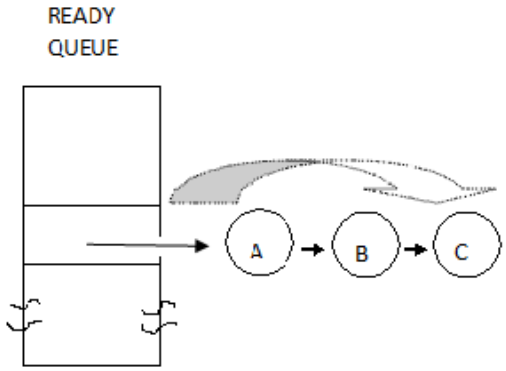

Fig III: Round Robin Scheduling

- The process is dispatched after quantum expires.
- The process release the CPU if the process have burst less than the time quantum
- The process is put into the tail of ready queue and the schedule select next process If the process have bust greater than quantum time.
- Display the results and Stop the process.

IV. EXPERIMENTAL RESULTS

We implemented five approaches to solve the task scheduling and memory allocation problem. They are mentioned below:

- Decoupled task scheduling and memory partitioning assuming equally partitioned SPM among alavailable processors TSMP−EQUAL;
- Decoupled task scheduling and memory partitioning with non-equally partitioned SPM among different processors, TSMP−ANY;
- Our integrated task scheduling and memory partitioning approach, TSMP−INTEG;
- Our heuristic with pipelining TSMP−PIPE;
- The optimal solution with pipelining based on the ILP formulation ILP−PIPE
- We used the Cjpeg real-life programs from the *Mediabench* and *MiBench*, as test benchmarks.

We used Simplescalar architectural simulation to profile the used benchmarks. Simplescalar can simulate the execution of an application on complex multiprocessor system on- chip architectures with different memory hierarchies. Through the simple scalar profiler, we extract the profiling data such as variable sizes and access frequencies and execution time of each task. The profiling is intended to: 1) divide each application into computation blocks referred to as

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

tasks; 2) find the computation times (Min, Avg, Max) for each task on each available processor in processor cycles; 3) find the number of variables; 4) the number of times each variable is used, freq; and 5) the size in bytes for each variable in the current application.

Table I: Comparison Chart

TSMP Based on	Execution Time(ms) for	
	Static Scheduling	Round Robin Scheduling
Equal	516.94	98.8
Non-Equal	28.58	20.78
Integrated	19.54	17.78
Integrated Approach With Pipelining	16.48	10
Pipelining based on ILP Formulation	14.26	5

We tested the benchmark, assuming a multiprocessor system on chip of two processors and a scratch pad memory with size that varies between 4 kB and 4 MB. We assume 100-cycle latency for off-chip memory access compared to 1-cycle latency for the SPM on-chip memory. We tested the benchmark under three SPM budgets chosen based on the size of the benchmark. The choice 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.

The columns in Table: I show the comparison between Static and Round Robin Scheduling for TSMP–EQ, TSMP–ANY, and TSMP–INTEG.

The improvement greatly depends on the structure of the embedded application. TSMP–ANY improved over TSMP–EQ from little improvement close to dramatic improvement.

Such improvements show that static memory allocation that is, partitioning the SPM budget equally among the processors limits the effectiveness of SPM memories as it does not consider the characteristics of the tasks assigned to a processor in its memory partitioning decision.

Our integrated approach for task scheduling and memory partitioning, TSMP–INTEG, further improved the results over the decoupled approach, TSMP–ANY. TSMP–INTEG improved over TSMP–ANY from little improvement close to dramatic improvement. This improvement 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.

The pipelining results emphasize the fact that such embedded applications can benefit significantly from pipelining. The pipeline cost is the computation time needed for one pipeline stage. As expected, our embedded applications greatly benefit from pipelining as the execution time is decreased on average compared to TSMP–INTEG. In order to show the effectiveness of our task scheduling/memory partitioning heuristic, INTEG, we compared it to an optimal integer linear formulation (ILP) based on the ILP formulation of this problem ILP–PIPE.

We tested our heuristic on the Cjpeg benchmark. Figure. IV show the results achieved by our heuristic when considering a system with four processors and an SPM budget ranging from 512K to 4Min a real time Cjpeg benchmark.

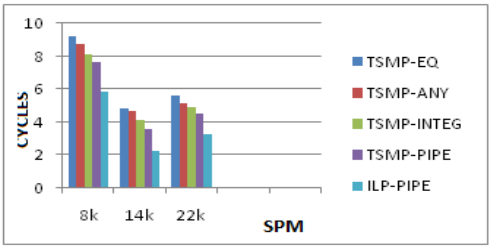

Fig IV: Results for cjpeg benchmark

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

VIII. CONCLUSION

In this paper, we presented an effective heuristic that integrates task scheduling and memory partitioning of embedded applications on system-on-chip with scratchpad memory. Our integrated approach using round robin scheduling significantly improved the results over static scheduling. This approach can be further enhanced using another scheduling algorithm.

REFERENCES

[1] Hassan Salamy and J. Ramanujam, "An effective solution to task scheduling and memory partitioning for Multi-processor system-on-chip" IEEE transactions on computer-aided design of integrated circuits and systems, vol. 31, no. 5, may 2012.

[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] 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.

[4] G. D. Micheli, R. Ernst, and W. Wolf, Readings in Hardware/Software\Co-Design. San Francisco, CA: Morgan Kaufmann, 2002.

[5] R. Neimann and P. Marwedel, "Hardware/software partitioning using integer programming," in Proc. DATE 1996,pp.473–480.

[6] K. S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," IEEE Trans. Very Large Scale Integr., vol. 10, no. 3, pp. 193–208, Jun. 2002.

[7] 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.

[8] 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.

[9] P. Panda, N. Dutt, and A. Nicolau, Memory Issues in Embedded Systems- on-Chip: Optimization and Exploration Dordrecht, The Netherlands: Kluwer, 1999.

[10] P. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. Off-chip memory: The data partitioning problem in embedded processors-bases systems," ACM Trans. Des. Automat. Electron. Syst., vol. 5, no. 3, pp. 682–704, Jul 2000.

[11] A. Dominguez, S. Udayakumaran, 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.

[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.

[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] IBM. ILOG CPLEX 8.1: Reference Manual, ILOG,Inc[Online].Available: http://www.ilog.com/products/cplex.

[19] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Media Bench: A tool for evaluating and synthesizing multimedia and communications systems," in Proc. IEEE Int. Symp. Micro architecture, Dec. 1997, pp. 330–335.

[20] 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.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

[21] T. Austin, E. Larson, and D. Ernst, "Simple Scalar: An infrastructure for computer system modeling," IEEE Compute., vol. 35, no. 2, pp. 59–67, Feb. 2002.