Runtime Scheduling of Dynamic Task Graphs Communication with Embedded Multiprocessors

K.Gowthami, N.Kumaresan

Abstract-Multiprocessor mapping and scheduling algorithms have been extensively studied over the past few decades and havebeen tackled from different perspectives.Task scheduling is an essential aspect of parallel programming. Most heuristics for this NP-hard problem are based on asimple system model that assumes fully connected processors and concurrent interprocessor communication. Hence, contention for communication resources is not considered in task scheduling, yet it has a strong influence on the execution time of a parallel program. This paper investigates the incorporation of contention awareness into task scheduling. The proposed methodology is runtime scheduling which is designed to reduce the wastage of time during static scheduling. We have assumed heterogeneous processors with broadcast and point-to-point communication models and have presented online algorithms for them. Experimental results shows that dynamic scheduling provides better performance than static scheduling.

Index Terms—Static, dynamic, edge scheduling, Heterogeneous processors, Communication between tasks.

I. INTRODUCTION

MULTIPROCESSOR-Based embedded systems have becomequite widespread in recent times. Embedded systems such as personal handheld devices, set-top boxes, and miniprinters consist of complex applications with real-time performance requirements. For such applications, a multiprocessor architecture is getting increasingly preferred over a single processor solution to achieve the required performance. Each processor in a multi-processor system usually has its local memory for code storage and execution. A macro data-flow graph (usually represented as a directed acyclic graph (DAG)) is used to describe an application at a high level. Mapping and scheduling of macro data-flow Graphs to such multiprocessor architectures are a fundamental and challenging problem that is being addressed by many researchers. Several methods have been proposed for Specific applications and architecture models to minimize the application execution time. The increasing requirement of computing power has shifted embedded system designs from a single processor to multiple-processor-on-a-chip solutions. A Typical multiprocessor consists of a set of processors connected via a communication subsystem for exchanging data. A directed acyclic graph (DAG) is used to describe an Application at a high level. Scheduling of the nodes to the processors (task scheduling) and edges to the communication Channels (edge scheduling) with the objective of minimizing Overall execution time of the DAG is called the general Multiprocessor Scheduling problem. The objective of scheduling is to minimize the completion time of a parallel application by properly allocating the tasks to the processors. In a broad sense, the scheduling problem exists in two forms: *static* and *dynamic*. In static scheduling, which is usually done at compile time, the characteristics of a parallel program (such as task processing times, communication, data dependencies, and synchronization requirements) are known before program execution.

Manuscript received on July, 2012.

K.Gowthami, PG Student, Department of ECE, Anna University of Technology, Coimbatore, India,

N.Kumaresan Assistant Professor, Department of ECE, Anna University of Technology, Coimbatore, India.

II. TASK GRAPH EXAMPLE

Above task graph example shows four tasks with variable execution times. Among these tasks if anyone task completed its execution before its fixed execution time means the remaining time will be wasted to avoid that online scheduling was introduced. In this Dynamic Scheduling the execution time of the task was decided during execution of the task. .

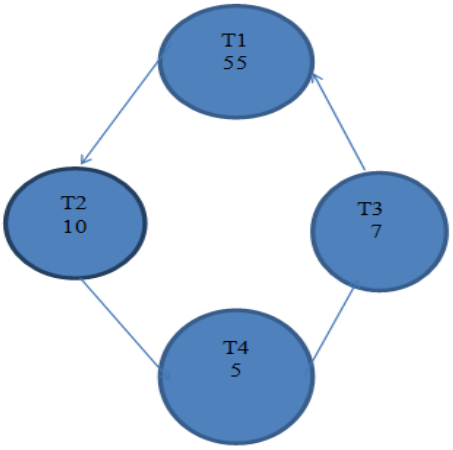

Fig2.1. An example task graph with variable execution times

III. ONLINE REMAPPING

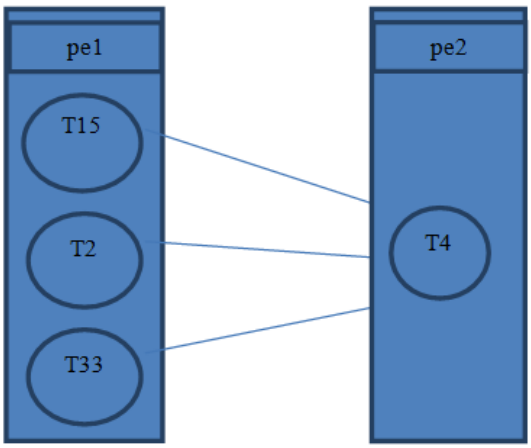

Fig 3.1 before Online Remapping

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

A. Online scheduling

Online scheduling is mostly used in real time applications to improve scheduling performance. Based upon the communication between the tasks there are two types of communications that is, Broadcast communication: In This type of communication each task in processor 1 delivers information to all the tasks in processor2. But this type of communication requires more channel for the communication.Point-to-Point communication: In point-to-point communication each task in processor1 delivers information to the task which defined by processor1.

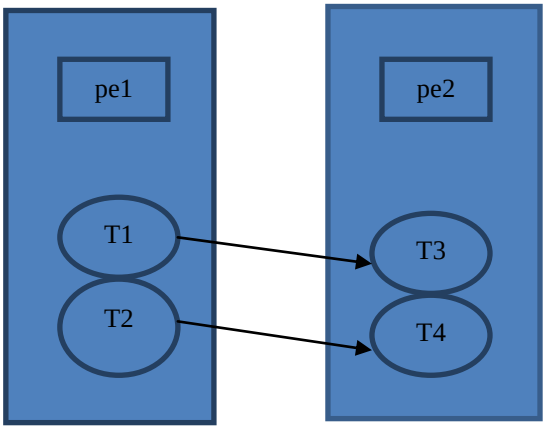

Fig 3.2 After online remapping

Fig 3.1 shows the static scheduling strategy that means execution time of the tasks was fixed. Each task executes in its specified time only. If any task completed its execution before or after its fixed execution time means the scheduling is not efficient.fig 3.2 shows after online remapping the execution time was improve with three units so online scheduling is provide better efficiency than static scheduling.

IV THE DAG SCHEDULING PROBLEM

The objective of DAG scheduling is to minimize the overall program finish-time byproper allocation of the tasks to the processors and arrangement of execution sequencing of the tasks. Scheduling is done in such a manner that the precedence constraints among the program tasks are preserved. The overall finish-time of a parallel program is commonly called the *schedule length* or *makes span*. Some variations to this goal have been suggested. For example, some researchers proposed algorithms to minimize the *mean flow-time* or *mean finish-time*, which is the average of the finish-times of all the program tasks. The Significance of the mean finish-time criterion is that minimizing it in the final schedule leads to the reduction of the mean number of unfinished tasks at each point in the schedule. Some other algorithms try to reduce the setup costs of the parallel processors. We focus on algorithms that minimize the schedule length.

A. The DAG Model

A parallel program can be represented by a directed acyclic graph (DAG),G =(V,E)where *V* is a set of *v* nodes and *E* is a set of *e* directed edges. A node in the DAG represents atask which in turn is a set of instructions which must be executed sequentially withoutpreemption in the same processor. The weight of a node $n_i$iscalled the computation costandis denoted by w (n_i). The edges in the DAG, each of which is denoted by (n_i ,n_j) , correspondto the communication messages and precedence constraints among the nodes. The weight ofAn edge is called the communication costof the edge and is denoted by (n_i,n_j).

B. Global Scheduler Model

The design of the global scheduler is essentially the core of the proposed methodology which either schedules a task communication at runtime. The global scheduler is notified by an event when there is a change in any of the processor states. The global scheduler's functionality can be modeled by the following two events.

1. Task completed event. A task completed event is generated by a processor when it completes execution of the assigned task on it. At this event, the global scheduler assigns the outgoing communications of the finishedtask to channels. Additionally, in the point-to-point communication model, it schedules the child tasks for a future time in a processor.
2. Free processor event. A free event is generated by a processor to notify the global scheduler when it is free and it has received a new communication. The proposed algorithm can overcome the deficiencies of these algorithms and have the following features

1. It assigns dynamic priorities to the nodes at each step based on the *dynamic critical path* (defined below) so that the schedule length can be reduced monotonically.
2. It changes the schedule on each processor *dynamically* in that the start times of the nodes are not fixed untilall nodes have been scheduled.
3. It selects a suitable processor for a node by looking ahead the potential start time of the node's *criticalchild* node on that processor.

C. Algorithm

Task Completed {Vi completes on p}
IfTr be Child if Vi€Vs else Childviwith exception of nodes based on the conditions evaluated at vi
While is not empty **do**
Let Tj be the first element in the ordering Φ (Tr)
Let C the channel in the bus B that provides
Minimum start time
IfVj is already scheduled on a processor
pe(Vj)then Schedule eij; pe(Vj) on C {schedule eijon Channel C intended for processor pe(Vj)}
Else
Let pe_kmin (PE Available time); (Vj,pe); {Select Channel for edge scheduling}Schedule e_{ij} on C from p tope_k {schedule e_{ij}onChannel C intended for processorpek}
Schedule Vj on pe_k
End if
Remove Vj from Vr
End while
End Event
EVENT - Free Processor Event {A processor peis free and a communication has arrived}
Let Vr be all the tasks that are ready on processor pe
Let v be the first element in the ordering of (Vr)
STv t {Schedule and Start v on pe} end

V EXPERIMENTALRESULTS

A. Simple ask generation output

Task Generator and Intergeneration Time blocks simulate the tasks that arrive at the shared processor. The amount of work or size of a task is set to 1 in the Set Task Size block. A new task waits in the FIFO Queue block until the Rate-Based Shared Processor subsystem can process the task. Each kind of task has a token generator subsystem in the Task Token Generators section of the model. The presence of the Start input port indicates that one kind of task depends on the completion of another kind of task. The output was simulated using the matlab tool.

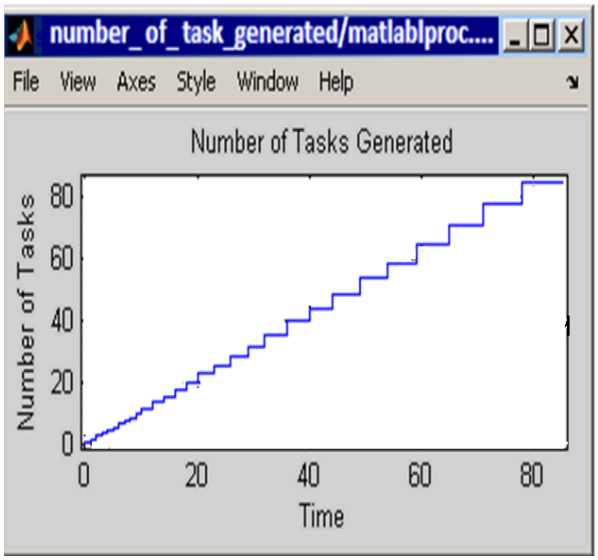

Fig 6.1 task generation

Fig6.1 shows the simple task generation example. In that when the number of task increases its overall execution time also increases. Tasks are created and simulated using the matlab. Fig 6.1 shows the task waiting service .Every processor has its own buffer to store the data's. When number tasks executes in parallel so there is contention between the tasks.

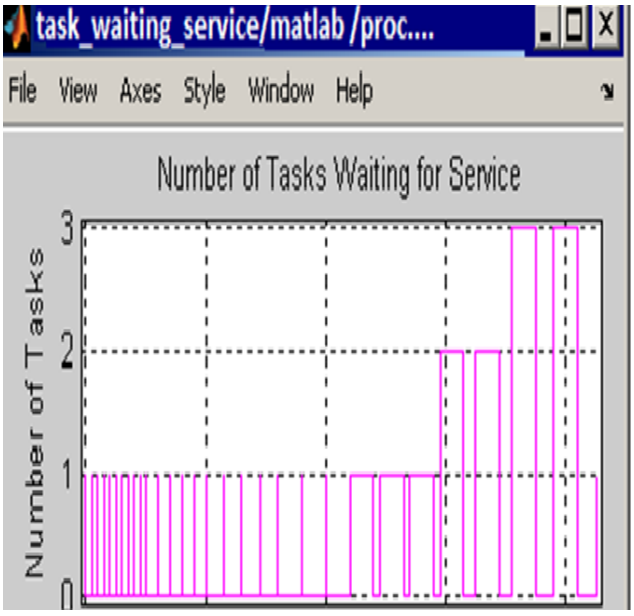

Fig6.1.1 number of tasks waiting for service

B. Comparison for static and dynamic scheduling

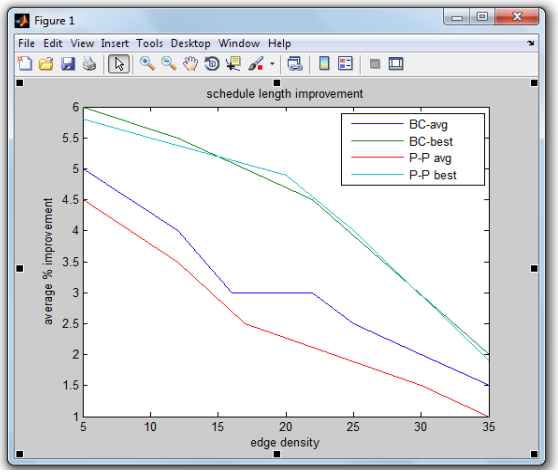

Fig6.2 comparison between static and dynamic scheduling

The schedule length generated by the proposed strategywas compared to the contention-aware static scheduler. Since this contention aware scheduler algorithm is not clearly defined for conditional task graphs, we made certain modifications to the static scheduler with the concepts of resource sharing given by Xie and Wolf [4] for comparison. If Li is the schedule length and L0 is the schedule length produced by the proposed strategy for the ith instance of total of I instances of a task graph G execution, the average case schedule improvement (Av Improve (%) is calculated as

$$\text{Avg improve} = \frac{\sum_{i=1}^{I} (L_i - L_i^{1} \times 100)/L_i}{I}$$

Online scheduling is mostly used in real time applications to improve scheduling performance.Upon the arrival of a new task or the departure of a completed task, the Scheduler subsystem recomputes the residual processing time for each task. Since the rated-based processor divides its resources equally among tasks, if there are two tasks in progress, the residual processing time for a task will be twice as long as the amount of unfinished work for that task.

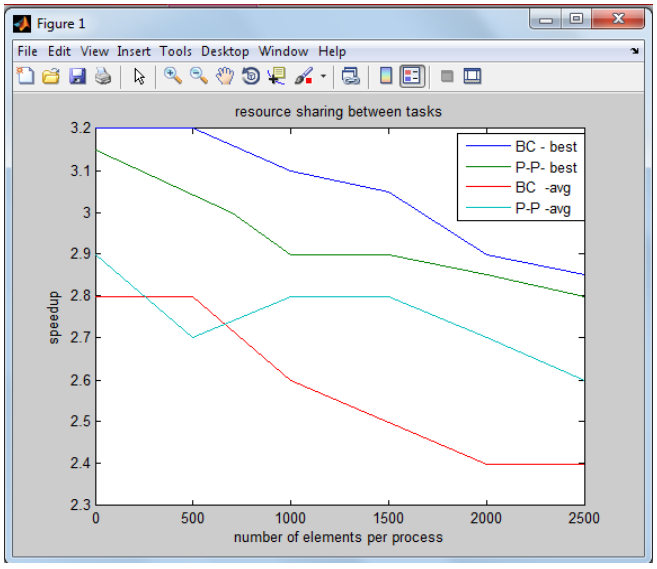

Fig6.2.1 Resource sharing between tasks

VI CONCLUSION

The proposed scheduling strategy schedules tasks to processors at runtime based on the execution behavior and performs the edge scheduling to handle contention. Experimental results show that the proposed solution provides better average-schedule length at runtime over the static schedulers for task graphs with unpredictable execution behavior. The algorithm presented in this paper may be suitably extended for heterogeneous processor models. We presented an efficient algorithm for generating schedules based upon a compact task graph representation. The scheduling heuristic exploits the potential for interleaving the execution of tasks on a single processor and overlapping their execution of tasks on multiple processors to maximize resource utilization. This approach is time efficient because it does not perform unnecessary splitting of larger tasks into smaller tasks. Using the basic multiprocessor scheduling algorithm we also developed a distributed on-line scheduling algorithm.

A. Future work

The proposed algorithm in its present form assumes a network of fully connected processors but can be generalized to other networks such as hypercube, mesh, etc. In order to accomplish that, the procedure for computing the start times of nodes on the processors will need to be modified and it will need to take into account the hop distances of the processors holding the parent nodes.

REFERENCES

1. A. Feldmann, , J. Sgall, M.-Y. Kao,and S.-H. Teng, “Optimal Online Scheduling of Parallel Jobs with Dependencies,” Proc. 25th Ann. ACM Symp. Theory of Computing (STOC ’93) pp. 642-651, 1993.
2. **A.** K. Agrawala, D. Moss, S. Noh, B. Trihn, and .MultipleResource Allocation for Multiprocessor Distributed Real-Time Systems.In Workshop on Parallel and Dist.Real-Time Syst., April 1993.
3. C. Hou , J. Yang, X. Ma, and Z. Yao, “A Static Multiprocessor Scheduling Algorithm for Arbitrary Directed Task Graphs in Uncertain Environments,” Proc. Eighth Int’l Conf. Algorithms and Architectures for Parallel Processing (ICA3PP ’08), pp. 18-29, 2008.
4. E. A. Lee, G. C. Sig, “Scheduling to Account forInterprocessor Communication Within Interconnection- Constrained Processor Network,” 1990 InternationalConference on Parallel Processing, Vol. 1,pp. 9-17, August 1990.
5. H. El-Rewini and T. G. Lewis, “Scheduling ParallelProgram Tasks onto Arbitrary Target Machines,” Journalof Parallel and Distributed Computing, Vol. 9, No. 2, pp.138-153, June 1990.
6. J. Jung,.K. Shin, M. Cha, M. Jang, , W. Yoon, and S. Choi, “TaskScheduling Algorithm Using Minimized Duplications in Homogeneous Systems,” J. Parallel Distributed Computing, vol. 68, no. 8, pp. 1146-1156, 2008.