JOINT DESIGN AND IMPLEMENTATION OF PROCESSOR SCHEDULE AND CALL ADMISSION CONTROL ON A SOFTWARE DEFINED RADIO TESTBED

Tiejun Zhang (Software Radio Lab, School of ECE, Georgia Tech, Atlanta, GA , USA, Tjzhang@ece.gatech.edu)
Thomas Pratt(Software Radio Lab, School of ECE, Georgia Tech, Atlanta, GA , USA)

ABSTRACT

This paper regards a software defined radio (SDR) basestation (BS) as a multiprocessor system (MPS) and the processing for the traffic from subscriber stations (SS) as tasks, and considers scheduling of tasks to make the best use of the processors. It also considers call admission control (CAC) based on the available processing resources. An implementation method in a software defined radio test environment is also proposed.

1. INTRODUCTION

This paper proposes a joint design method and an implementation structure for processor scheduling and call admission control on a SDR testbed at the software radio lab, Georgia Institute of Technology. The scenario under consideration involves a BS for wireless multimedia communications with voice and data traffic. It is assumed that limited digital signal processing (DSP) resources are used to carry out the signal processing. In our study, the BS is regarded as an MPS, and the processing for the traffic from each SS is regarded as a task. In this paper, we first address the issue of scheduling tasks on processors. Next, we describe how to combine the processor scheduling with CAC. The BS CAC is used to decide whether or not the SS's request for admission will be accepted by the BS. For CAC design, many factors should be considered, for example, the wireless resources (bandwidth etc.) that the system has.

This work is supported by Georgia electronic design center(GEDC)

However, in this paper, we only consider the CAC design from the standpoint of available processing resources. The last sections discuss the design and implementation of the CAC.

2. PROCESSOR SCHEDULE ALGORITHM

The scheduling algorithm described in this section is based on the algorithm proposed by the first author in [1]. The following definitions are given:

1. The processors in our system are denoted as P={ P_1 , P_2 , … P_m }, m is the number of the processors
2. The tasks scheduled to an MPS are defined as {X, Y, r_i , d_i , Q_r }, where:

X={ T_1 , T_2 ,… T_n } is the set of tasks to be executed;

Y is the relations between tasks, the algorithm proposed assumes all tasks are independent;

r_i denotes the execution time of T_i , which is common to each processor

d_i is the deadline of task T_i , , where $1 \leqslant i \leqslant n$.

Q_r is the priority level of a task, Two priority levels are defined: level A and B, with level A designating a higher priority task than level B.

The following performance measures are defined:

1. The mean waiting time *W,* where the waiting time w_i of a task T_i is defined as the time that the task spends in the system before it is executed. Formally:

$$W=(\sum_{i=1}^{n} w_i)/n$$

2. The schedule length, which is defined as the maximum completion time

$C=\max\{c_i\}, 1\leqslant i\leqslant n$

where $c_i = r_i + w_i$, $1\leqslant i\leqslant n$.

The optimal value of C_{opt} is defined as:

$$C_{opt} = SI(n)/m, \quad \text{where } SI(n)= (\sum_{i=1}^{n} r_i)$$

C obtained from the proposed scheduling algorithm (presented below) is usually worse than C_{opt}, and offers a relative performance given by the following relation[1]:
$C/C_{ont} \leq 2m/(m+1)$.

SCHEDULING ALGORITHM

The scheduling approach of this algorithm may be understood as follows: Each task has its own priority level: A or B; tasks with level A must be scheduled to enough processors to ensure they are completed before the designated deadline; the remaining processors are scheduled to tasks with level B. If the quantity of processors is sufficient, all the tasks with level B can be scheduled to ensure they are finished before their deadline; otherwise, only a portion of tasks can be finished before the deadline, and some tasks must be discarded. The discarded tasks are selected randomly. The following assumptions are made:

1. The number of processors for the MPS is denoted as m .
2. Tasks with the same level have the same deadline.
3. n_1, d(A) is the number and deadline of tasks with level A, respectively
4. n_2, *d(B)* is the number and deadline of tasks with level B, respectively
5. d(A)<d(B)
6. Suppose all the tasks arrive at time S1(shown in fig.1), and scheduling is determined at this moment.

All the scheduled tasks will enter execution queues, where each processor is assumed to have an execution queue. Some processors (named as non dedicated processors) process tasks with level A before time S2 (shown in fig.1), then process tasks with level B; the other processors (named as dedicated processors) are dedicated to process tasks with level B.

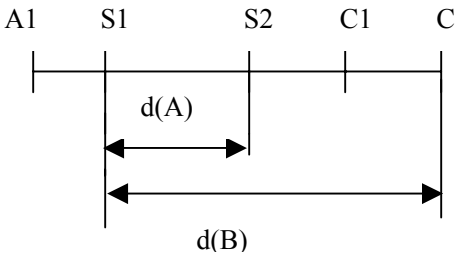

Fig.1 The timing for the task scheduling

The steps of the proposed algorithm are as follows:

(1) Determine the number of processors, m_1 for tasks with level A, where m_1 can be derived from the following relations:

$$C_{opt}(A)= SI(n_1)/m_1, \text{ where } SI(n_1)= (\sum_{i=1}^{n_1} r_i)$$

$(2* m_1/(m_1+1))* C_{opt}(A) \leqslant d(A)$,

$C(A)\leq d(A)$

Refer to [1] for the method of obtaining C(A).

(2) Determine the task set$\{n_B\}$ for tasks with level B.

The task set$\{n_B\}$ is made up of two subsets:$\{n_{B1}\}$and $\{n_{B2}\}$. $\{n_{B1}\}$ is the set of tasks with level B entering dedicated processors at time S1(shown in fig.1), the number of dedicated processors is m_2=m- m_1. $\{n_{B2}\}$ is the set of tasks with level B entering non-dedicated processors at time S2. The selected task set must satisfy the following relations for n_{B1}:

$$C_{opt}(n_{B1}) = \mathrm{SI}(n_{B1})/m_2, \text{ where } \mathrm{SI}(n_{B1}) = (\sum_{i=1}^{n_{B1}} r_i)$$

$$(2* m_2/(m_2+1))* C_{opt}(n_{B1}) \leq d(B)$$

$$C(n_{B1}) \leq d(B)$$

Similarly for n_{B2}:

$$C_{opt}(n_{B2}) = \mathrm{SI}(n_{B2})/m_1,$$

$$\text{where } \mathrm{SI}(n_{B2}) = (\sum_{i=1}^{n_{B2}} (r_i + d(A)))$$

$$(2* m_1/(m_1+1))* C_{opt}(n_{B2}) \leq d(B),$$

$$C(n_{B2}) \leq d(B)$$

Refer to [1] for the method of obtaining $C(n_{B1})$ and $C(n_{B2})$.

By calculating C(A), $C(n_{B1})$ and $C(n_{B2})$, we can determine how many level A tasks and level B tasks can be accepted by the BS for a given number of processors.

3. CAC DESIGN

A CAC method is derived that is based on the availability of processing resources. As shown in Fig.1, suppose at time A1, several SS requests for admission have been received by BS. At this point, the BS will make a decision as to which SS tasks will be admitted based on the type of task (the service SS requests is regarded as a task to be processed) and the available processors. The BS uses the scheduling algorithm to set up a task/processor allocation table, and the unallocated tasks are refused. The BS sends the appropriate information to SS. At time S1, the data streams from each SS arrive and are routed to processors and processed by times S2 and C for level A and level B tasks, respectively. In the mean time, another batch of requests for admission have been received by time C1, at which time the above decision process repeats.

A design example is given as follows. Suppose:

1. There are 20 tasks with level A and 40 tasks with level B to be scheduled.
2. d(A)=10, r(A)=5, r(A) is the execution time for every task A.
3. d(B)=20, r(B)=10, r(B) is the execution time for every task B.

The relation between the available processors and the number of admitted tasks is shown in Table 1.

Table1 The relation between the number of processors and the number of admitted tasks

Tasks \ Processors	10	15
Task A	20	20
Task B	10	20

4. IMPLEMENTATION

The Georgia Tech testbed uses DSPs, POWERPCs and the realtime operating system VXWORKS. A structure to emulate BS scheduling and CAC is shown in fig.2.

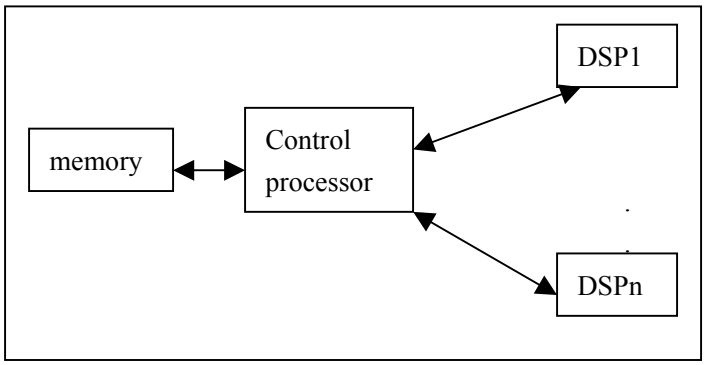

Fig.2 The structure of BS

The following processes are proposed to emulate the scheduling and CAC at the BS:

1. At time S1 (shown in Fig.1), all the task's data come into the memory, and the control processor (CP) maintains a memory address table for all the tasks.
2. The CP uses the Vxworks task manage function to set up the handler for each process according to task-processor allocation table. The CP also uses this table to configure some DSP processors as dedicated DSPs and the others as non-dedicated.
3. When a DSP finishes one task, it will send interrupt to CP, at which point the CP will send next task data to the DSP.

5. CONCLUSION

The joint design method for processor schedule and CAC proposed in this paper considers CAC from the point of processor resource utilization, a feasible implementation structure is given.

REFERENCES

[1] Zhang Tiejun, "A schedule algorithm for multiprocessor system used in WCDMA software radio base station " Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd , Volume: 3 , 6-9 May 2001,Page(s): 1889 -1891 vol.3