Schedule design for multiprocessor systems

Prof. Larisa S. GLOBA
DSc. in Computer Engineering,
IEEE Professional Member,
Chair of Information-telecommunication Networks,
Institute of Telecommunication Systems
National Technical University of Ukraine “Kiev Polytechnic Institute”

Ph.D. Student Dmytro LYSENKO
Ph.D. Student in National Technical University of Ukraine „Kyiv Polytechinic Institute”.

Abstract

Efficiency of multiprocessor system usage is strongly dependent on methods of schedule design – the way of task distribution on each processor to decrease overall schedule time. This article is devoted to the part of this process - schedule design on example of software development for LTE and WiMAX base stations.

Keywords: schedule, parallel programming, multi-processor system, genetic algorithm.

1. Introduction

Active telecommunication devices growth aim developers to constant updating of existing software and development of new one. With the appearance of multi-processors computer systems and parallel programming this topic got more challenging.

Thus, the main task of this article is decreasing of the telecommunication software development time by automation of schedule design, errors finding, debug simplification and software workflow visualization. This approach based on concept of modified model driven software development was discussed in [4,5].

To increase telecommunication software development, the original method of feasible schedule design for system on chip (SOC) software is offered. Hardware platform is SOC with a few types of different processing units (further “resources”). The digital signal transformation algorithm is defined by tasks list with dependency information. The resource type and processing time are defined beforehand for each task.

Other tasks resolved during feasible schedule design are minimizing the overall latency, meeting the deadline, and maximizing usage of system resources.

2. SOC software framework

SOC software framework is shown on Error! Reference source not found.. The messages containing Data and Control information flow between Upper and Lower Subsystems. Application Programming Interface (API) Parser and Data Analysis and Partitioning(DAP) module is responsible for providing Scheduler, placed on system level, with Task Lists created from these messages. Scheduler is responsible for dispatching and posting Tasks from Tasks Lists to a proper Hardware Abstraction Layer (HAL) driver at the correct time. Therefore, the Task Scheduler’s job is to decide “what” and “when” a Task should be executed. In case of multiple resources of the same type, the Task Scheduler may determine the resource number on which a Task runs, or rely on the HAL driver to make this assignment.

When in real time mode, Scheduler can’t use resource-intensive scheduling algorithms and its debugging becomes a complex task. Therefore schedule analysis is performed on PC in offline mode. In this case scheduler simulator can use different levels of HAL layer emulations.

Online Scheduler, as shown on Error! Reference source not found., scans Task List for available task, selects suitable resource and assigns task to it, till Tasks List will become empty. The logging stage is added to this process to estimate designed schedule quality and simplify debug process. Log analysis performed on PC in offline mode and consists of two stages (see Error! Reference source not found.). After processing on the stage #1 Task List log information is sending to PC. On stage #2, after log analysis and making a decision about necessary changes on Task List, the device restarts and changes applied before processing Task List. After downloading new log information the conclusions about successfullness of applied changes should be made.
3. Mathematical model of computational process

The source data for the digital signal transformation algorithm, used during base station operation, consists of next information:

- Set of tasks $\mathcal{W}$ and specified dependency relation: for each task $c \in \mathcal{W}$ corresponds certain subset $\mathbf{P}_c$ of its dependents. Meanwhile, for some tasks, $\mathbf{P}_c = \emptyset$.
- For each task are assigned: non negative integer value of priority $\Pi_c$, resource type index $\mathbf{r}_c$, run time $t_c$, and unique key $K_{cs}$. This key used to avoid uncertainty during task sorting.
- Set of available in SOC resources is known beforehand and includes information about all resource type count $R = \{R_1, R_2, ..., R_t\}$.

A simplest way to visualize task list information is to put it as directed graph $G_2 = \langle M, N, T \rangle$ where $M$ - tasks, $N$ - dependency relation, $T$ - mapping of set $N$ to direct product $M \times M$; correlates each dependency $i \in N$ in ordered tasks pair, first of which is dependent on second. This graph $G_2$ can provide visual information about Task List topology and is used to check source data for loop absence.

In practice, task list visualization as $G_2$ graph sometimes becomes inconvenient. Due to “arrows mess” graph topology can’t be observed and it makes this visualization useless.

To avoid this, it was decided to represent Task List considering tasks as arcs. In this case task’s start and stop events will be nodes in the new graph $G_2$. New graph $G_3 = \langle M', N', T' \rangle$ creation needs more time and computational resources but this way of visualization is free of previous method disadvantages and much more usable.

4. Scheduling Algorithms

The current problem of the tasks assignment to specific processors is an NP-complete and only be solved with heuristics [2]. On the other hand real time mode processing resource-intensive algorithms can’t be used. So, the scheduler implements several allocations algorithms to minimize the possibility of not finding a schedule. The algorithms 1, 2 and 3 can be used in online mode; the algorithm 4 can be implemented on Personal Computer (PC) only due to its high computational resource requirements.

Algorithm #1: Simple.

This algorithm follows two step processes: first assigns the task to resources and second schedules the tasks on each processor using uniprocessor algorithms to guarantee minimum computational resources requirements for scheduling. The algorithm is run every time when a scheduler is invoked and a task is completed.

Algorithm #2: Critical path based.

Any delays for critical tasks are the cause of increasing overall schedule latency. To avoid this lets set task priorities grounds on current problem [3]. The algorithm scheme shown on Error! Reference source not found. is run every time when a scheduler is invoked and a task is completed.

Algorithm #3: Critical path based with resource reservation.

In some cases algorithm #2 can be mistaken because of no reservation resources mechanism for critical tasks. Adding such check to algorithm logic will reduce delay possibility for critical path but also will require additional computational resources.

Algorithm #4: Genetic

To found suboptimal solution the genetic algorithm was used for task assignment. It is a type of stochastic algorithms that involves heuristic methods using evolutionary mechanism to search and select suboptimal solutions. It is proved as good solution for the current problem [3]. The algorithm scheme shown on Error! Reference source not found...
initial task list. Adding on this stage to population individual based on algorithm #2 will reduce number of iteration to found suboptimal solution but for results visualization such individuals were not added when calculating.

The time of designed schedule is used as objective function (OF). This time was calculated using algorithm #1. So OF can be written as

$$
G \left( \varphi_{12} \varphi_{41} \ldots \varphi_{24}, r_{10} r_{11} \ldots, r_{40}, P_{10} P_{11} \ldots P_{40}, P'_{10} P'_{11} \ldots P'_{40} \right)
$$

where $G$ - task priorities, $r$ - task indexes, $P$ - dependencies, $P'$ - additional dependencies. In terms of objective function two genes types can be defined - task priority $G$ and additional dependencies $P'$. The set of these two genes makes chromosome.

To provide variability for population lets define mutation as changing on one gene within chromosome. Task that going to be added as new dependency should have upper level in multilevel graph, according to current task to avoid loops. Next function is used to provide parent selection during crossing stage: 80% of parent selected from 20 % individuals that have best OF, 10% from 10% individuals that have worst OF and 10% of others. During crossing combination of genes from both parents is used to produce new individual. To avoid regression pedigree individuals maintains stage is added. Small number of individual with best value of OF is frozen and not be involved in mutation process. Experience has shown that number of such individual should be 0.05% from population size but not less than 1. Genetic algorithm stops when feasible solution is found or population degradation is detected. Also it can be limited by time or generations size.

After algorithm #4 has completed not only schedule parameters can be used as a result but the best individual chromosome with information about task priorities and additional dependencies can be also used. Applying this information on initial task list will cause decreasing schedule latency using algorithm #2 in online mode. This process is shown on Error! Reference source not found.

5. Outcome of experiment

Defined algorithms were implemented using “C” and “C#” languages as part of “Trancende 4G Base Station Processor” development process. The series of experiments were done for the algorithms efficiency analysis and setting up its base parameters such as mutation occurrence probability, population size, crossings number and pedigree individuals quantity.

Series of 32 tasks lists produced during WIMAX frame generation were used as test samples. In addition one task list was taken from LTE frame generation. This sample is notable by high paralleling abilities. All tasks list uses two types of resources and consists of from 97 to 1087 tasks and from 116 to 87317 dependencies.

On Error! Reference source not found. are shown computing results when number of first type resources are 2 and number of second type resources are 4 for algorithms #2, #3 an #4 against algorithm #1. For genetic algorithm population of 200 individuals was used and computation stopped on 150th generation.

Algorithms #2 and #3 always show the same or better results than algorithm #1. In most cases algorithm #3 shows the same result as #2 but sometimes deference can be up to 5%. For sample 4, genetic algorithm shows worse result, but with increasing population size to 1000 individuals and generations number to 500 the result becomes the same as for algorithm #2. See Error! Reference source not found. Best solution was found on 378th generation and stays the same for 2000 next generations. Similar result was received for other cases when algorithm #4 shows results worse than #2. In all cases population size and number of generation increase allows to get the same or better result against #2 and #3 algorithms. The significant results must be noted and it is shown by genetic algorithm for sample with good paralleling abilities (see sample 38) even for small population size. Eventually for all samples algorithm #4 shows the same or best result among others, but at the important parameter, computing time, is increasing. For example, for sample 4, schedule length 413 was found using algorithm #2 for less than 1 second but finding individual with the same objective function by algorithm #4 takes more than 3 hours.

On Error! Reference source not found. was shown best, worst and average result for population during all computing process. Population degradation can be found for this case on 170th generation. So, the situation when best objective function becomes equal to the average value for population can interpreted as the sign that further computation is not reasonable.
Computation performed for the cases with more resources number shows that with increasing of available resources number all algorithms shows better results and these results trend to critical path value. On the contrary, with decreasing number of available resources algorithm results trend to sum of all tasks length.

Experience has shown that next parameters should be chosen for efficiency of genetic algorithm: If tasks number in tasks list is \( N \) therefore population size should be \( N \approx 3 \), mutation number per generation \( N \approx 2 \), number of pair for crossing \( N \approx 0.6 \), number of descendants for each pair 2, and getting the result close to optimal will take \( N \approx 1.5 \) iterations.

6. Conclusions

For solving the task of schedule design suggested and implemented

- Mathematical models of computational process using node and arrow diagram methods.
- These models analysis and transformation methods for providing visualization of computational process.
- Four types of scheduling algorithms, three of which can be implemented in real time mode and one, which is resource intensive, can be used to found suboptimal solution in offline mode.

1. Implemented genetic algorithm, that allows finding schedule close to optimal and use task list parameters obtained during computation to make algorithms running in online mode, get the same result.

2. Experiments for more than 30 samples were performed for different number of available resources. Genetic algorithm parameters that help to take better results on shorter computing time found.

7. References