

Simultaneous Multistandard PHY Processing: A Low-Complexity Processor Resource Management Algorithm

Khoo Kiak Wei and Kambiz Homayounfar†

PHYBIT, INC.
10-10 Cendex Center, 120 Lower Delta Road, Singapore 169208
E-mail: khoo@phybit.com

†PHYBIT, INC.
AM Building 5F, 2-3-3 Higashi Gotanda, Shinagawa, Tokyo 141-0022, Japan
E-mail: kambiz@phybit.com

Abstract This paper presents a simple and efficient processor resource management algorithm for simultaneous execution of multiple air interfaces at the physical layer. Processor resources refer to available memory and cycles. Each physical layer task is called a load and characterized in terms of total memory and cycles it requires. The key challenges faced by our proposed method are twofold. First, the air interfaces are asynchronous. A new load can be presented to the processor at any time during its execution of a previous load. Second, obviously there are limits to available memory and cycles within a single-chip processor. How does the algorithm handle overload conditions? In this paper, we describe how these problems can be solved efficiently.

Key words Software-Defined Radio (SDR), Real-Time Operating System (RTOS), Physical Layer (PHY), Multi-standard, Scheduling Algorithms, Greedy Algorithms, Resource Allocation.

1 Introduction

Scheduling multiple tasks on a given processor is an NP-complete problem. This means that if we want to pack as many tasks as possible into an arbitrary processor, there is no known way to find the best combination in polynomial time. Hence, approximations are used and heuristics are put to work. Methods such as bin packing, strip packing, knapsack, rate-monotonic scheduling, earliest-deadline-first, [1]. are examples of the empirical approaches used in the RTOS scheduling processes. However, in situations when a processor is required to support a set of multi-rate wireless standards (such as WiMAX, 3G LTE, and WiFi) and must have the capability to process all the standards simultaneously, the problem becomes interesting. In these cases, the challenge is to find the best combination of the rates that each standard support and fits within the processor resource limits such as memory, cycles, and I/O bandwidth. None of the above-mentioned methods is applicable as they are not able to guarantee the generation of a proper set of rates such that all the tasks can be scheduled to fit on processor. Most of the methods above will pack as many tasks as possible with the highest possible rate and drop or hold those tasks that cannot fit. In hard real-time systems, this is not acceptable. Given the choice between dropping a task (not executing at all), and lowering its rate, it seems effective to choose the latter. Fortunately, all new wireless standards are capable of operation at a plurality of physical layer rates. This

capability was originally introduced into PHY to enable operation under a variety of channel conditions. If the channel is good, run as fast as possible. When channel becomes poor, slow down.

Naturally, an optimal solution can be found by brute-force method. This approach checks all the possible combinations and selects the best. However, the number of combinations increases exponentially with the number of tasks (or supported rates), and the brute-force approach becomes impractical.

Table 1: Definitions for Processor Resource Management Algorithm

S	Available cycles
M	Available memory
I	Number of PHY tasks
J_i	Possible rates for task i
$\{r_{i,j}\}$	Cycles for task i when using bit-rate j
$\{m_{i,j}\}$	Memory sizes for task i when using bit-rate j
$\{b_i\}$	Best bit-rate for each task i

Note that the rows sets $\{r_{i,j}\}$ must be arranged in a decreasing order, i.e. the highest bit-rate with the highest cycle requirement is always chosen first. The columns of $\{r_{i,j}\}$ must be arranged in increasing order. As an example:

$$\{r_{i,j}\} = \begin{bmatrix} 9.3 & 5.5 & 1.3 & 0.9 \\ 7.5 & 5.2 & 2.1 & 0.7 \\ 13 & 8.5 & 4.0 & 1.3 \\ 14.2 & 6.3 & 5.7 & 2.2 \end{bmatrix} \quad (1)$$

The same sorting scheme applies to $\{m_{i,j}\}$. Once the lists are sorted, they will enter the Algorithm 1.1 to find the sub-optimal solution for the best rates for each task, denoted by $\{b_i\}$.

Algorithm 1.1 Simultaneous Multistandard Processor Resource Management Algorithm

```

1:  $b[] \leftarrow \{-1, -1, \dots, K\}$ ;
2:  $i \leftarrow 0$ ;
3:  $m_T \leftarrow 0$ ;
4:  $s_T \leftarrow 0$ ;
5: while ( $i < I$ ); do
6:    $j = b[i] + 1$ ;
7:    $\phi = F$ ;
8:   while ( $j < J_i$ ); do
9:     if ( $(m_T + m_{i,j}) < M$ ) & ( $(s_T + s_{i,j}) < S$ )
10:      then
11:         $m_T \leftarrow m_T + m_{i,j}$ ;
12:         $s_T \leftarrow s_T + s_{i,j}$ ;
13:         $b[i] = j$ ;
14:        break;
15:      end if
16:       $j \leftarrow j + 1$ ;
17:    end while
18:    if  $\phi$  then
19:       $i \leftarrow i + 1$ ;
20:    else
21:       $i \leftarrow i - 1$ ;
22:       $m_T = m_{i,b[i]}$ ;
23:       $s_T = s_{i,b[i]}$ ;
24:      if  $i < 0$  then
25:        error;
26:        break;
27:      end if
28:    end if
29:  end while

```

2 Processor Resource Management Algorithm

We propose a heuristic method to find the best combination of the tasks by a greedy algorithm. By greedy, we mean any algorithm that follows the problem-solving approach of making a locally optimum choice at each stage with the hope of finding the optimal point at the end. As such, it cannot guarantee finding the globally optimal solution. Greedy algorithms do not operate exhaustively on all the data as they make commitments to certain choices as early as possible. Yet they have been proven effective in a variety of task-scheduling, resource allocation, or packet scheduling problems because they

are computationally efficient and give good approximations to the optimum. These features are essential for low-cost, embedded RTOS applications.

3 Algorithm Operation

The outer while-loop iterates over the number of tasks and the inner while-loop iterates through the number of available bit-rates for a particular task. The set $b[]$ keeps track of the best combination found thus far. The variables m_T and s_T are the memory and cycle consumption respectively. For any bit-rate j of task i , if the condition $((m_T + m_{i,j}) < M) \& ((s_T + s_{i,j}) < S)$ is met, the bit-rate j will be recorded in the vector $b[]$. Then the algorithm moves on to the next task. If none of the bit-rates of task i can be allocated, the algorithm will backtrack to task $i - 1$ and begin from the last found bit-rate j . At the end of algorithm, the vector $b[]$ contains the bit-rate of every task that can fit into the processor specification.

4 Results

Generally, the complexity of the algorithm in the worst case is $O(I^J)$ where I is the number of task and $J = \sum_i J_i$ is the number of available bit-rates. However, due to the characteristics of the greedy algorithm, the complexity is always between the average case $O(IJ)$ and best case $O(I)$.

5 Conclusions

A simple task scheduling algorithm has been presented to fit simultaneous tasks into a software-defined physical layer processor.

References

- [1] A.M. K. Cheng, Real-Time Systems: Scheduling, Analysis, and Verification, Wiley Inter-Science, 2002.
- [2] A. Nillson, E. Tell, and D. Liu, Simultaneous multi-standard support in programmable baseband processors, IEEE PRIME, Otranto, Italy, June 2006.