A Priority-Based Budget Scheduler with Conservative Dataflow Model

Marcel Steine[1], Marco Bekooij[2], Maarten Wiggers[1]
[1]Eindhoven University of Technology, Eindhoven, The Netherlands
[2]NXP Semiconductors, Eindhoven, The Netherlands
m.steine@tue.nl

Abstract

Currently, the guaranteed throughput of a stream processing application, mapped on a multi-processor system, can be computed with a conservative dataflow model, if only time division multiplex (TDM) schedulers are applied. A TDM scheduler is a budget scheduler. Budget schedulers can be characterized by two parameters: budget and replenishment interval.

This paper introduces a priority-based budget scheduler (PBS), which is a budget scheduler that additionally associates a priority with every task. PBS improves the guaranteed minimum throughput of a stream processing application compared to TDM, given the same amount of resources.

We construct a conservative dataflow model for a task scheduled by PBS. This dataflow model generalizes previous work, because it is valid for a sequence of execution times instead of one execution time per task which results in an improved accuracy of the model. Given this dataflow model, we can compute the guaranteed minimum throughput of the task graph that implements the stream processing application.

Experiments confirm that a significantly higher guaranteed minimum throughput of the task graph can be obtained with PBS instead of TDM schedulers and that a conservative bound on the guaranteed throughput of the task graph can be computed with a dataflow model. Furthermore, our bound on the guaranteed throughput of the task graph is accurate, if the buffer capacities in the task graph do not affect the guaranteed throughput.

1 Introduction

Modern multimedia systems, such as in-car entertainment systems and smart-phones, offer an increasing amount of functionalities to their end-users by simultaneously executing a number of real-time stream processing applications. For performance reasons, this processing is done by embedded multi-processor systems.

The applications are implemented by task graphs that consist of tasks that communicate over first-in-first-out (FIFO) buffers. To prevent buffer under-run and overflow, each task is only allowed to start executing if there is sufficient data and space in the buffers for the task to finish without having to wait for additional input data or output space. This required amount of data and space might be data-dependent, which potentially results in an a-periodic execution of the task. Resources are typically shared between multiple tasks to reduce costs. Tasks can share a resource with one or more tasks from other applications of which there might be no knowledge about execution times or execution rates.

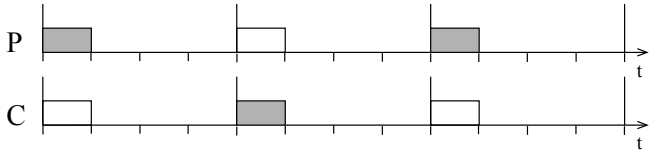

Figure 1. Worst-case schedule for TDM

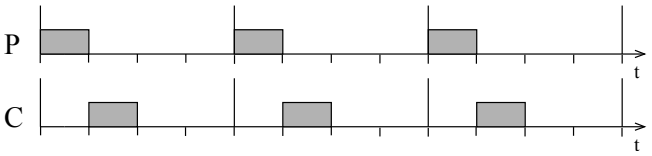

Figure 2. Worst-case schedule for PBS

Applications typically have a minimum throughput constraint. If budget schedulers are applied, then the minimum throughput of an application is independent of the execution times and execution rates of tasks of other applications due to the guaranteed minimum amount of time for every task to use the resource. Therefore, the use of budget schedulers simplifies analysis of the minimum throughput and improves the robustness of a system.

In [15] it has been shown that guarantees about the minimum throughput of an application can be given with dataflow analysis techniques, provided that budget schedulers are applied that can be characterized by a minimum time budget that is available in a maximum replenishment interval. A task can only be scheduled when it has budget remaining. An example of such a budget scheduler is a time division multiplex (TDM) scheduler.

In this paper, we introduce a priority-based budget scheduler (PBS) that in addition to minimum time budget and maximum replenishment interval has a third parameter, which is the task priority. Given the same amount of resources, PBS improves the guaranteed throughput of a stream processing application compared to budget schedulers that schedule tasks based on two parameters. For PBS, we derive a conservative dataflow model. This dataflow model is a generalization of the model presented in [15], because it is also applicable for budget schedulers that are characterized by two parameters and it is valid for a (cyclic) sequence of (worst case) execution times per task. Inclusion

978-0-7695-3782-5/09 $25.00 © 2009 IEEE
DOI 10.1109/DSD.2009.148

of a sequence of task execution times instead of a single execution time results in an improved accuracy of the analysis results. Furthermore, this dataflow model is also applicable for tasks which execute a-periodically.

TDM gives every scheduled task a guaranteed time slice in which the task can use the resource before the next task is scheduled. With PBS, we take the same approach. However, while TDM schedules tasks in a fixed sequence, PBS dynamically schedules tasks within a replenishment interval based on (1) the availability of sufficient input data and output space and (2) the task priorities.

The worst-case schedules in terms of throughput shown in Figure 1 and Figure 2 provide an example in which PBS improves the guaranteed throughput compared to TDM. Assume we want to compute the guaranteed throughput of a producer-consumer task graph that consists of a producing (P) and consuming (C) task that are executed on different processors. For this simple example both the producing and the consuming task have an execution time of 1 time unit. Their minimum guaranteed budget is also 1 time unit which is equal to the length of one time slice. The tasks produce or consume one container per execution. The FIFO buffer has a capacity of one container. Both schedulers allocate one slice per replenishment interval of 4 time units for either the producing or consuming task. There are three remaining time units which can be used by the other tasks using the same processor as the producer or consumer. The boxes indicate the location of the slices in the replenishment intervals, and these boxes are shaded in case a task executes.

In case both processors use a TDM scheduler, the worst-case schedules for the application are shown in Figure 1. The consuming task executes one replenishment interval after the production of a container because the worst-case situation is that the containers are produced at the end of the time slice of the consumer. The producing task must wait until the consuming task has finished its execution because the FIFO buffer has a capacity of one container. The alignment of slices as in Figure 1 cannot be prevented to occur in a system with non-synchronized clocks per processor [8].

In case both processors use a PBS scheduler, the worst-case schedules for the application are shown in Figure 2. Instead of assuming that the consuming task has just used its complete budget when the producing task finishes as is needed for TDM, with PBS the consuming task can dynamically be scheduled such that it can use its processor time after the producing task and the current task running on the processor are finished. As a consequence, space becomes available earlier, which enables the producing task to already execute in the next replenishment interval. If the time slices for the other tasks sharing the processor are also 1 time unit, the consuming task can execute immediately after the producing task finishes.

The amount of processing resources allocated to a task is described by the budget ratio, which is the minimum budget available in a maximum replenishment interval. In this example, the budget ratio is one time unit per 4 time units for both the producer and consumer. However, even though the budget ratio and buffer capacity is the same in both cases of this example, PBS schedulers, in this example, double the guaranteed throughput compared to TDM schedulers.

The outline of this paper is as follows. We first discuss related work in Section 2. Then in Section 3, we describe the PBS scheduler in more detail. In Section 4, we derive a conservative dataflow model for tasks scheduled by PBS. Our PBS implementation is discussed in Section 5. In Section 6, the accuracy of the dataflow model and the improvement in throughput by using PBS compared to TDM are evaluated through experiments. We discuss future work in Section 7 and conclude in Section 8.

2 Related Work

Much research has been done on schedulers and the analysis of the tasks to be scheduled. The currently available schedulers differ in the type of information needed for the scheduler to be useful and to allow analysis of the throughput of an application. Classical scheduling approaches such as Earliest Deadline First [5] and Rate Monotonic [9] need information about execution times and execution rates of all tasks sharing the same resource to allow analysis. In the domain of stream processing applications this information might not always be available. Some tasks might only be characterised by an average or estimated execution time, for which no useful bound on the execution time can be given. Due to possible data-dependent input and output behaviour of a task, we might not be able to predict arrival times within our application even if external events are periodic. Therefore, there might be no useful bound on the execution rate of a task. With guaranteeing a minimum budget in a maximum replenishment interval, budget schedulers provide resource virtualization which allows to quarantee upper-bounds on the temporal behaviour of applications independent of knowledge about tasks of other applications. This technique of guaranteeing a minimum budget in a maximum interval as implemented by a budget scheduler is also known as resource reservation [1] or capacity reservation [11]. Several budget schedulers are known, such as the Time Division Multiplex (TDM) scheduler or schedulers using Constant Bandwidth Server (CBS) [1] or Processor Capacity Reserves (PCR) [11].

A design time analysis of the throughput of an application can only be made if the budget ratio of tasks can be guaranteed at design time. If task switching overhead cannot be neglected, the maximum task switching overhead that can occur during a replenishment interval must be known at design time. For budget schedulers based on Constant Bandwidth Server [1] or Processor Capacity Reserves [11], the maximum number of task switches depends on the best-case execution times of the tasks. No useful bound on these best-case execution times might be available. For TDM, as well as for PBS as introduced in this paper, a tight bound on the maximum number of task switches in a replenishment interval is known at design time, which

is independent of the execution times or execution rates of tasks.

The dataflow model presented in [15] uses a latency and rate parameter, based on the latency-rate model as introduced in [14], to capture the worst-case behaviour of a task scheduled by a TDM scheduler. Given this dataflow model, the guaranteed throughput of a stream processing application can be derived with exact algorithms [4, 6] or with approximation algorithms [16]. These approximation algorithms have a polynomial computational complexity. Dataflow analysis has a number of attractive properties. First, dataflow analysis can deal with arbitrary cyclic dependencies. This allows the task graph to contain cyclic data dependencies and allows the capacity of the buffers to affect the throughput. As a consequence, a tradeoff can be made between buffer capacity and budget ratio. Second, in [17] it is shown that dataflow analysis techniques are applicable for tasks that produce or consume a data-dependent number of containers. Third, dataflow analysis techniques accurately compute a guaranteed throughput of the task graph that is independent of the best-case execution times of the tasks. Other analysis techniques [3, 7, 10] typically have difficulties with arbitrary cyclic dependencies, data-dependent execution rates of tasks, and rely on best-case execution times.

3 Priority-based budget scheduler

This section describes PBS in detail without considering task switching and synchronization overhead as this is implementation dependent. The overhead of our implementation is discussed in Section 5.

Similarly to TDM, PBS guarantees every scheduled task a minimum time slice in which it is allowed to use the resource before the next task is scheduled. Every task has a minimum guaranteed budget in a maximum replenishment interval, where the budget is the sum of its time slices.

In addition to TDM, priorities can be assigned to tasks when using PBS. In this paper we consider tasks that are scheduled by PBS to have either a low or a high priority. Of the tasks scheduled by one PBS scheduler there can be maximally one high priority task.

The sequence in which the low priority tasks are scheduled is fixed at design time, similarly as is done for all tasks scheduled by TDM. As the budget can consist out of multiple slices a task can be scheduled multiple times in this sequence. For the high priority task, the partitioning of budget into time slices is done at run-time. Before a low priority task is scheduled, first the high priority task is scheduled, if it has remaining budget. The high priority task executes until there is no sufficient data and space available in the buffers to finish its execution, i.e. until it is no longer *enabled*, or until it has no remaining budget, after which the next task is scheduled. The length of the time slices for the high priority task therefore depends on its enabling and available budget, while for the low priority tasks this length is fixed at design-time.

After all low priority tasks have received their budget, the high priority task is again scheduled, if it has remaining budget. As soon as the high priority task slice ends, the budget of each task is replenished and a new replenishment interval starts. Note that, in contrast to TDM, the replenishment interval of PBS is not necessarily always equal to the maximum replenishment interval. It might be possible that it is less then the maximum if the high priority task does not require (all of its) budget during the replenishment interval.

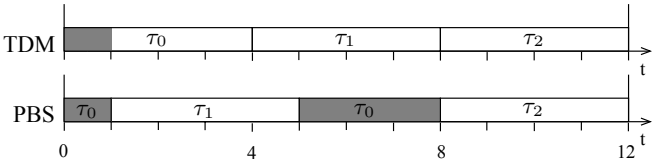

Figure 3. Comparison of TDM and PBS schedule

The procedure of selecting the next task allowed to use the resource as used by PBS can be described by the pseudo-code of Listing 1.

Listing 1 Determine next task to schedule with PBS

P_t : previous task using the resource
HP : high priority task
B_{hp} : remaining budget for HP
B_{max} : maximum budget for HP
$LP[i], 0 \leq i < n$: list of n low priority task slices
N_{lp} : pointer the next LP task slice to schedule

```
1:  if P_t ≠ HP and B_hp > 0 then
2:      next = HP
3:  else if N_lp < n then
4:      next = LP[N_lp]
5:      N_lp = N_lp + 1
6:  else {end of replenishment interval}
7:      next = HP
8:      N_lp = 0
9:      B_hp = B_max
10: end if
11: return next
```

In Figure 3 a comparison is made between a TDM and PBS schedule for one particular replenishment interval of 12 time units in which three task are scheduled. Every task has a budget of 4 time units. Assume that in the replenishment interval task τ_0 is enabled two times, at time 0 and 4, with a corresponding execution time of 1 and 3 time units. The shaded parts of the time slices indicate when τ_0 is executing. For TDM the budget for task τ_0 can only be used during a time slice which is fixed at design time. Task τ_0 can execute during the first part of this time slice due to the first enabling. The rest of the time slice is 'wasted' as the task is not enabled during the remainder of the time slice. For the second enabling budget is only available in the next replenishment interval. With using PBS to schedule these three tasks and setting τ_0 to be the high priority task the schedule will be constructed such that at time 0 the high priority task τ_0 can directly start executing for 1 time unit after which

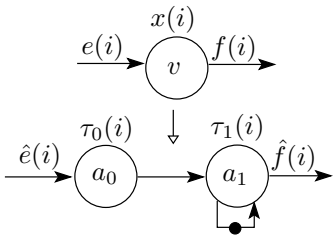

Figure 4. Modelling a task by two dataflow actors

the next task is scheduled. As τ_0 has remaining budget, τ_0 can start as soon as a new task switch is made after the second enabling. If we compare the TDM and PBS schedule we see that with PBS we can start the second execution of τ_0 earlier than when using TDM. This can have a positive influence on the throughput as seen in the example used in the introduction.

It is easy to see that PBS is a generalization of a TDM scheduler because a PBS and TDM scheduler have the same behaviour if all tasks have a low priority.

With PBS we dynamically schedule tasks in the replenishment interval which influence the worst-case waiting time of the tasks. The waiting time of a task is defined as the difference between the enabling time and start time of a task that has remaining budget at this enabling time. This waiting time of a task can influence the throughput of an application. For TDM this worst-case waiting time is the same for all tasks and is equal to the replenishment interval minus the tasks budget. For PBS the waiting time of all tasks is also known at design time as all tasks sharing the same resource have a limited and known influence on each others waiting time. The worst-case waiting time of the high priority task is reduced to the maximum slice length of the low priority tasks, while the worst-case waiting time of a low priority task is increased to the maximum length of the replenishment interval plus the budget of the high priority task minus the budget of the low priority task. As the worst-case waiting time of the high priority task depends on the maximum length of the low priority slices, reduction of the maximum slice length of the low priority tasks reduces the worst-case waiting time of the high priority task, which potentially results in a further increase of throughput. This can, however, come at the cost of an increased number of task switches. This is because the maximum number of task switches in a replenishment interval is $2N+1$, where N is the number of low priority slices.

4 Dataflow model for PBS

In this section we present a dataflow model which captures the worst-case behaviour of an application of which tasks are scheduled on a processor using PBS.

Figure 4 depicts a task v and its model, which is a dataflow component consisting of dataflow actors a_0 and a_1. The production and consumption of containers by task v is modelled by the production and consumption of tokens in the dataflow model. The enabling condition of task v is equal to the firing rule of actor a_0. Actor a_0 produces one token per firing and actor a_1 fires, if there is one token available at both inputs.

For the task, let $e(i)$ be the time at which the i-th execution of the task is enabled, and let $f(i)$ be the finishing time of the i-th execution of the task. The execution time of the i-th execution of the the task is defined as $x(i)$. The execution time $x(i)$ is the maximum time between the enabling time $e(i)$ and finish of the i-th execution in case this task is the only task executed on the processor.

For the dataflow component, let $\hat{e}(i)$ be the enabling time of actor a_0 and let $\hat{f}(i)$ be the time at which actor a_1 finishes and produces its output tokens. The variables $\tau_0(i)$ and $\tau_1(i)$ represent the time between enabling and production of tokens by respectively actor a_0 and actor a_1 for the i-th firing.

A dataflow component is a conservative model of a task scheduled on a processor if Equation (1) holds.

$$e(i) \leq \hat{e}(i) \Rightarrow f(i) \leq \hat{f}(i) \tag{1}$$

In [15] it has been shown that if all tasks in an application are modelled by conservative dataflow components, i.e. Equation (1) holds, then the minimum throughput of the application can be derived analytically with dataflow analysis techniques. Another option to determine the throughput is to observe the worst-case finishing times of tokens with a dataflow simulator [2], which produces exact results.

During self-timed execution, actors fire as soon as they are enabled. The finishing times of the dataflow component during self-timed execution is expressed with Equation (2).

$$\hat{f}(i) = \max(\hat{e}(i) + \tau_0(i), \hat{f}(i-1)) + \tau_1(i) \tag{2}$$

In the next two sub-sections, we derive expressions for $\tau_0(i)$ and $\tau_1(i)$ for the high and low priority tasks such that Equation (1) holds for each dataflow component. In this derivation T denotes the maximum replenishment interval and B the minimum guaranteed budget.

4.1 Dataflow model for the high priority task

In this section, we derive $\tau_0(i)$ and $\tau_1(i)$ for the dataflow component corresponding to the high priority task, such that Equation (1) holds. To achieve this, we start this section by showing that $f^w(i)$ as given by Equation (3) is an upper bound on the actual finishing time of the i-th execution $f(i)$ of the high priority task i.e $f(i) \leq f^w(i)$. In this equation, W_h is the worst-case waiting time of the high priority task, which is equal to the maximum slice length of the low priority tasks. Subsequently, we derive the appropriate values for $\tau_0(i)$ and $\tau_1(i)$ such that $e(i) \leq \hat{e}(i) \Rightarrow f(i) \leq f^w(i) \leq \hat{f}(i)$ and therefore Equation (1) holds.

$$f^w(i) = \max(e(i) + W_h, f^w(i-1)) + \frac{T \cdot x(i)}{B} \tag{3}$$

In Lemma 1, we first determine the guaranteed remaining budget for the i-th execution of the high priority task in

the replenishment interval in which the worst-case finish of the high priority task $f^w(i-1)$ occurs. Then a case distinction is made based on the start of the i-th execution relative to this $f^w(i-1)$. We assume that $f^w(i-1)$ occurs in the m-th replenishment interval. We define b_r as the last budget replenishment moment before $f^w(i-1)$ and b_e as the first budget replenishment moment after $f^w(i-1)$.

Lemma 1. *If $b_e - f^w(i-1) = \frac{T \cdot Y}{B}$ then the budget available for the i-th execution of the high priority task during the m-th interval is at least Y.*

Proof. Assume $f^w(i-1) - b_r = \frac{T \cdot X}{B}$ and $b_e - f^w(i-1) = \frac{T \cdot Y}{B}$. Because $\frac{T \cdot X}{B} + \frac{T \cdot Y}{B} \leq T$ it holds that $X + Y \leq B$. By making use of induction we can conclude from the definition of $f^w(i)$ that if $f^w(i-1) - b_r = \frac{T \cdot X}{B}$ at most X budget is used in the m-th replenishment interval by firings previous to firing i. From this we can conclude that the remaining budget during the m-th interval is $B - X \geq Y$. □

Theorem 1. *The bound $f^w(i)$ as given in Equation* (3) *is an upper bound on the finish time of the i-th execution of the high priority task, i.e. $f(i) \leq f^w(i)$.*

Proof. We consider three cases based on the start of the i-th execution $s(i)$.

(1). If $b_e \leq s(i)$ then the budget available for the i-th execution of the task is fully replenished and available. Therefore, $f(i) \leq s(i) + x(i) + \lfloor \frac{x(i)}{B} \rfloor (T - B) \leq s(i) + \frac{T \cdot x(i)}{B}$. In this case we know that $s(i) \leq e(i) + W_h$, so $f(i) \leq e(i) + W_h + \frac{T \cdot x(i)}{B}$.

(2). If $f^w(i-1) \leq s(i) \leq b_e$ then we assume that x_1 budget is used before b_e and x_2 budget is used after b_e, $x(i) = x_1 + x_2$. We define $\frac{T \cdot Y}{B} = b_e - f^w(i-1)$. Now we can conclude by Lemma 1 that $x_1 \geq Y$ and therefore $x_2 \leq x(i) - Y$. As the budget is fully replenished at moment b_e we can conclude that $f(i) \leq b_e + \frac{T \cdot (x(i) - Y)}{B}$ for the same reason as the previous case. As $b_e - s(i) \leq \frac{T \cdot Y}{B}$ it holds that $f(i) \leq s(i) + \frac{T \cdot Y}{B} + \frac{T \cdot (x(i) - Y)}{B} \leq s(i) + \frac{T \cdot x(i)}{B}$. In this case we know that $s(i) \leq e(i) + W_h$, so $f(i) \leq e(i) + W_h + \frac{T \cdot x(i)}{B}$.

(3). If $s(i) \leq f^w(i-1)$ we assume the worst-case start $s(i) = f^w(i-1)$ to conclude by the same reasoning as the previous case that $f(i) \leq f^w(i-1) + \frac{T \cdot x(i)}{B}$. This latest possible start is the worst-case situation as earlier starts cannot finish later.

By combining the three cases we can conclude that $f(i) \leq \max(e(i) + W_h, f^w(i-1)) + \frac{T \cdot x(i)}{B} = f^w(i)$ and therefore this theorem holds. □

Given Theorem 1 we can use induction to conclude that $\tau_0(i)$ and $\tau_1(i)$ of Equation (2) can be chosen W_h and $\frac{T \cdot x(i)}{B}$ respectively, such that $e(i) \leq \hat{e}(i) \Rightarrow f(i) \leq f^w(i) \leq \hat{f}(i)$ and therefore Equation (1) holds.

Inclusion of $x(i)$ in Equation (2) enables to model a sequence of execution times, which is not possible with [15].

4.2 Dataflow model for the low priority task

In this section we derive expressions for $\tau_0(i)$ and $\tau_1(i)$ for the low priority tasks such that Equation (1) holds for the corresponding dataflow components.

Let W_l be the worst-case waiting time of a low priority task. This waiting time is equal to the maximum length of the replenishment interval plus the budget of the high priority task minus the budget of the low priority task.

Using the same type of reasoning as for the high priority tasks we conclude that $\tau_0(i)$ and $\tau_1(i)$ of Equation (2) can be chosen W_l and $\frac{T \cdot x(i)}{B}$ respectively, such that $e(i) \leq \hat{e}(i) \Rightarrow f(i) \leq f^w(i) \leq \hat{f}(i)$ and therefore Equation (1) holds.

As PBS is a generalization of a TDM scheduler the analysis of this section can also be applied to a task scheduled by a TDM scheduler. The worst-case waiting time W_t of a task scheduled by a TDM scheduler is equal to the maximum replenishment interval minus the budget of the task, which is lower then W_l as no extra waiting time is introduced by a high priority task. A task scheduled by TDM can then conservatively be modeled by taken $\tau_0(i)$ equal to this W_t while $\tau_1(i)$ remains $\frac{T \cdot x(i)}{B}$.

The increase of W_l compared to W_t is the cost of the reduced W_h. An increase in waiting time does not reduce the minimum throughput of an application as long as actors that correspond with the low priority tasks do not belong to critical cycles in the dataflow graph. These critical cycles determine the minimum throughput [13]. Therefore, PBS can give us the opportunity to increase the throughput by decreasing the length of the critical cycle, at the expense of longer non-critical cycles, by carefully setting the priorities.

5 PBS implementation

In this section, we describe our PBS scheduler implementation and discuss how task switching and synchronization overhead are taken into account such that the dataflow model is conservative.

In our implementation, we use the polling based synchronization scheme that is described in [12]. In this scheme, the task that produces a container updates the administration of the FIFO buffer. The task that is waiting for data or space polls the buffer administration for the availability of the containers. We do not need to include this busy-waiting time in the execution times of the tasks [16], because its effect on the throughput of the task graph is taken into account when analysing the complete dataflow graph.

At the end of every time slice a task switch is made. The beginning of a task switch is indicated by a Fast Interrupt reQuest (FIQ) set by the scheduler. The high priority task can initiate to a task switch as soon as it is not enabled anymore, i.e., polling the buffer administration for sufficient data and space has failed, by using a SoftWare Interrupt (SWI) which gives back control to the scheduler to decide on the next task to schedule.

A task switch includes (1) saving the context of the current task, (2) saving the remaining budget in case the current task is the high priority task, (3) determining the next task to be scheduled, (4) restoring the context of the next task, (5) starting the budget accounting of the next task, and (6) starting the next task. An upper bound on the time needed for this task switch should be taken into account when constructing a conservative dataflow model.

We implemented our scheduler in assembly code for a model of the ARM7-TDMI core that does not have a cache. Two of these cores are connected to a dual ported memory in a cycle true simulator such that each core has a single cycle memory access latency. With this simulator we observed a maximum time needed for one task switch of 346 cycles, which is larger than the time needed for a task switch for a TDM implementation due to the slightly more complex decision that is made to determine the next task to schedule. For a TDM scheduler implemented on the ARM7-TDMI a maximum time of 249 cycles for one task switch is observed.

To obtain the guaranteed net budget of a task, the maximum task switching overhead should be subtracted from the gross budget of the task. This gross budget of a task is equal to the sum of its time slices in a replenishment interval. The maximum task switching overhead for a low priority task is equal to $M \times 346$ cycles, where M is the number of slices for the task and a maximum of 346 cycles are needed for one task switch. The maximum task switching overhead for a high priority task is equal to $(N + 1) \times 346$ cyles, where N is the number of low priority slices in a replenishment interval. Task switching overhead is correctly taken into account in the dataflow model if the budget B in Equation (3) is equal to the guaranteed net budget of the task, as this is the actual guaranteed minimum amount of time that the task can use the resource.

6 Experimental results

In this section, we present results measured with a cycle true simulator and compare them with results obtained by dataflow simulation.

In the experiments of this section we use a producer-consumer application for which both the producer and consumer are running on a different processor and have a high priority. Several other tasks are sharing a processor with the producer or consumer and are assigned a low priority. A measured upper bound on the execution time of the producer and consumer is 360790 clock cycles and 360530 clock cycles, respectively. For both the processors it holds that the gross budget for the high priority task is 4 Mcycles and the total gross budget for the low priority tasks is 32 Mcycles. The total gross budget for the low priority tasks is divided in 8 low priority slices with a length of 4 Mcycles.

Both the producer and consumer iterate through an infinite loop in which they do some processing as soon as polling the buffer administrations for sufficient data and space in the input and output buffers is successful. The time needed for this processing is bound by the worst-case execution time. The time needed for polling is not part of this execution time but is taken into account when analysing the complete dataflow graph, as discussed in the previous section. Finally, they produce containers with data or space. As soon as containers are produced by the consumer, a monitor in the cycle-true simulator will print the current time $f(i)$. By executing the dataflow model of the producer-consumer application in a dataflow simulator we obtain the values for the guaranteed worst-case finishing times $\hat{f}(i)$.

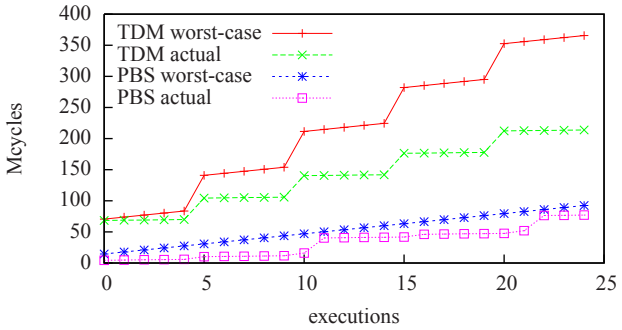

Figure 5. Finishing times of consumer for buffer capacity of 5 containers

In the first experiment, we verify whether the analytical results are conservative, evaluate the accuracy of the bounds and observe whether the use of PBS instead of TDM improves the guaranteed throughput for a producer-consumer application in which the tasks communicate one container per execution. In the second experiment, we evaluate the influence of the waiting time of the high priority task on the guaranteed throughput. In the third experiment, the producer has a sequence of execution times and we evaluate the increase of accuracy when the dataflow model includes the sequence of execution times compared to a single execution time. Finally, in the fourth experiment, we determine the accuracy of the bound for a producer which communicates a variable amount of containers per execution.

6.1 Experiment 1

Figure 5 shows the first 25 actual and worst-case finishing times for both PBS and TDM scheduling, with a buffer capacity of 5 containers and the same budget ratio for both tasks. From this figure, we can observe that the actual throughput using PBS scheduling is almost a factor 4 higher than using TDM given the same budget ratio for both tasks.

From Figure 5, we can also observe that the computed worst-case finishing times and the actual finishing time diverge for TDM scheduling while this is not the case for PBS scheduling. The reason is that for this experiment the throughput for PBS scheduling is completely determined by the execution times of the tasks while for TDM scheduling also the buffer capacity influences the throughput. Using

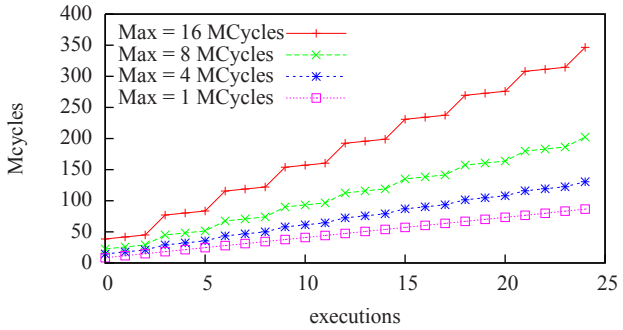

Figure 6. Finishing times of consumer for buffer capacity of 3 containers

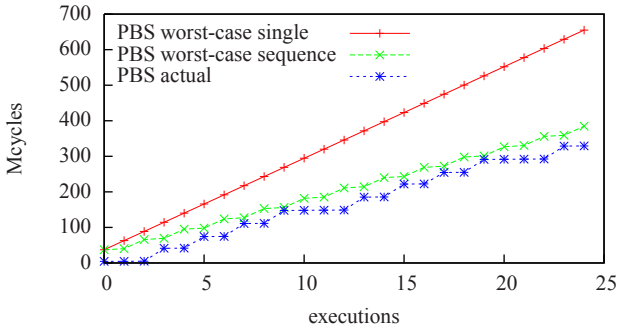

Figure 7. Finishing times of consumer for alternating execution times

dataflow analysis to compute a buffer capacity that does not influence the guaranteed throughput results in a sufficient buffer capacity of 22 containers, in case of TDM. Our simulations confirm that, for this capacity, the bound on the maximum finish times, in case of TDM, no longer diverges from the actual finishing times. Furthermore, with a buffer capacity of 22 containers TDM has the same throughput as when PBS with a buffer capacity of 5 containers.

In this experiment, the task switching overhead for PBS scheduling is maximal 17 task switches per replenishment interval, while for TDM scheduling the task switching overhead is 9 task switches. The task switching overhead in this experiment is less than 0.02% and 0.01% for PBS and TDM scheduling, respectively.

For this experiment, the computation of the upper bounds on finishing times by our dataflow simulator requires a fraction of a second, while one simulation run of the cycle-true simulator requires more than 10 minutes.

6.2 Experiment 2

In the previous experiment it is assumed that the largest low priority slice is 4 Mcycles. The worst-case waiting-time of the high priority task depends on this largest low priority slice. Therefore, there is an extra trade-off between the maximum length of the low priority slices and the number of task switches. In this experiment the influence on the throughput of the producer-consumer job for different maximum low priority slice lengths is determined.

Figure 6 shows the worst-case finishing times of the first 25 executions for maximum slice lengths of the low priority tasks between 1 and 16 Mcycles. In all cases the buffer capacity is equal to 3 containers. The results show that a higher throughput can be obtained if we reduce the slice lengths of the low priority tasks, at the expense of extra task switches.

6.3 Experiment 3

In this paper a model is introduced that can include tasks with a (cyclic) sequence of execution times. It is expected that this has a positive influence on the accuracy of the model compared to a model that can only include one worst-case execution time per task.

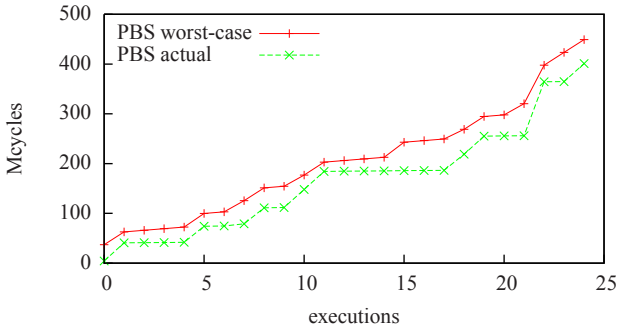

Figure 8. Finishing times of consumer for variable production quanta

In this experiment, we evaluate the accuracy of the model using a cyclic sequence of (worst-case) execution times and compare it to using just one (worst-case) execution time. For the producing task we now have a cyclic sequence of execution times. In an alternating fashion, subsequent executions have an upper bound on their execution time of 2860780 and 368790 cycles. The capacity of the buffer is 6 containers.

Figure 7 shows the finishing times of the first 25 executions as observed in the cycle-true simulator. Furthermore, it shows the worst-case finishing times for the dataflow model with only one worst-case execution time and the worst-case finishing times for the dataflow model with a sequence of worst-case execution times taken into account.

The results show that the model with a sequence of worst-case execution times is conservative and produces a more accurate result than a model in which just one execution time is taken into account.

6.4 Experiment 4

While the previous experiments assumed a static enabling condition for the producer and consumer, the model introduced in this paper is valid for every enabling condition. In this experiment dataflow analysis is applied to a producer-consumer application in which the producer has a more complex enabling condition. Instead of producing one container in every execution as was done in the previ-

ous experiments, the data producing task produces between zero and four containers in every execution. We increased the execution time of the data producing task to have one upper bound of 2861750 cycles. The capacity of the buffer is 5 containers.

Figure 8 shows the first 25 actual and worst-case finishing times for PBS. The results show that the model for a task with variable production quanta is conservative and produces an accurate result.

7 Future work

Currently, the PBS scheduler we consider in this paper has two priority levels of which there is one high priority task which is allowed to initiate a task switch. While this already gives a potential improvement compared to TDM, it might be possible to allow a broader trade-off between worst-case waiting times by introducing more priority levels. When determining a conservative dataflow component for the tasks scheduled by these variants it is expected that the same proof can be used as is currently used for the variant with one high priority task, only the waiting times for the tasks change depending on the number of other tasks that can potentially influence their waiting time. More research and experiments with different variants of PBS might be part of future research.

In this paper we use a dataflow model which models a task by two dataflow actors as is currently done in literature. Further improvements in the accuracy of the dataflow model might be possible by increasing the number of actors in the dataflow component. Proofs of the corresponding values for the actors are needed and experiments should be done to determine the possible benefits of such an extended dataflow model.

Furthermore, as PBS can be applied for any kind of shared resource, determining the impact of using PBS instead of TDM for different types of shared resources, such as interconnect or memories, might be an interesting direction for future work.

8 Conclusion

In this paper, we introduce the priority-based budget scheduler (PBS), which improves the guaranteed throughput of stream processing applications compared to a time division multiplex (TDM) scheduler, given the same amount of resources.

Furthermore, we derive a conservative dataflow model of a task scheduled by PBS. With this dataflow model the guaranteed throughput can be computed even if the application contains cyclic dependencies. The same dataflow model can also conservatively model a task that is scheduled by TDM and is, furthermore, suitable for a sequence of execution times per task, which improves the accuracy of the analysis results compared to a single worst-case execution time per task.

For PBS a tight bound on the number of task switches can be computed at design time, which is independent of the execution times of the tasks. For our PBS implementation, we found the worst-case cost of a task switch. Together, this allows us to derive a conservative dataflow model that takes task switching overhead into account.

Experiments confirm that PBS scheduling can result in a significantly higher throughput than TDM scheduling. A comparison between measured and analytical results confirms that the dataflow model is conservative, which indicates that the task switch costs are correctly taken into account. Our experiments show that the analytical bounds are accurate, if the guaranteed throughput is completely determined by the execution times of the tasks and not influenced by the buffer capacities. The dataflow model is shown to be able to include a (cyclic) sequence of execution times and can leverage this additional information to obtain more accurate analysis results compared to only a single execution time. Furthermore, the presented dataflow model still computes accurate and conservative estimates of the guaranteed throughput in case tasks execute a-periodically.

References

[1] L. Abeni and G. Buttazzo. Resource Reservation in Dynamic Real-Time Systems. *Real-Time Systems*, 27(2):123–167, 2004.

[2] M. Bekooij, S. Parmar, and J. van Meerbergen. Performance guarantees by simulation of process networks. In *SCOPES*, pages 10–19, 2005.

[3] R. L. Cruz. A Calculus for Network Delay: Part II: Network Analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991.

[4] A. Dasdan. Experimental Analysis of the Fastest Optimum Cycle Ratio and Mean Algorithms. *ACM Transactions on Design Automation of Embedded Systems*, 9(4):385–418, October 2004.

[5] M. Dertouzos. Control Robotics: The procedural control of physical processes. *Information Processing*, 74:807–813, 1974.

[6] A. Ghamarian et.al. Throughput Analysis of Synchronous Data Flow Graphs. *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pages 25–36, June 2006.

[7] M. Jersak, K. Richter, and R. Ernst. Performance Analysis of Complex Embedded Sytems. *International Journal of Embedded Sytems*, 1(1–2):33–49, 2005.

[8] M. Krsti, E. Grass, F. K. Grkaynak, and P. Vivet. Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook. *IEEE Design and Test of Computers*, 24(5):430–441, 2007.

[9] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time environment. *J. ACM*, 28:101–155, 2004.

[10] A. Maxiaguine et.al. Tuning SoC Platforms for Multimedia Processing: Identifying Limits and Tradeoffs. In *CODES+ISSS*, pages 128–133, 2004.

[11] C. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 90–99, May 1994.

[12] A. Nieuwland et. al. C-HEAP: A Heterogeneous Multi-Processor Architecture Template and Scalable and Flexible Protocol for the Design of Embedded Signal Processing Systems. *Design Automation for Embedded Systems*, 7(3), 2002.

[13] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000.

[14] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, 1998.

[15] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *SCOPES*, pages 11–22, 2007.

[16] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Buffer Capacity Computation for Throughput Constrained Streaming Applications with Data-Dependent Inter-Task Communication. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 183–194, 2008.

[17] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Computation of Buffer Capacities for Throughput Constrained and Data Dependent Inter-Task Communication. In *DATE*, pages 640–645, 2008.