

Dynamic Scheduling of Real-Time Tasks on Multicore Architectures

Thomas Megel, Vincent David, Damien Chabrol
 CEA LIST, Embedded Real Time Systems Laboratory
 Point Courrier 94, Gif-sur-Yvette, F-91191 France
 Email: Firstname.Lastname@cea.fr

Christian Fraboul
 IRIT - ENSEEIHT, Toulouse University
 2, rue Camichel 31000 Toulouse - France
 Email: Christian.Fraboul@enseeiht.fr

Abstract—We present a new dynamic scheduling on multicore architectures. This is an improvement of the Optimal Finish Time (OFT) scheduler introduced by Lemerre[7] reducing preemptions. Our result is compared with other schedulers and we show that our algorithm can handle with more general scheduling problems.

I. INTRODUCTION

We are interested in embedded multiprocessor architecture for real-time systems, *i.e.* dealing with safety-critical systems, for industrial purposes such as avionics, automotive or nuclear industry or domotics. We present a dynamic optimal scheduling for real-time tasks which means that if the task set is feasible, our scheduling meets all deadlines. It applies to tasks with same start time and deadline, periodic and time-triggered tasks and can handle with non-predictable tasks thanks to its *dynamic (online)* behaviour and *optimal finish time* characteristic. After some definitions in section 2, we present our optimal algorithm in section 3 and conclude in section 4 on ongoing works.

II. DEFINITIONS

We consider a system task Γ composed of several tasks. A task T releases several finite or infinite consecutive *jobs*. Each job J is characterized by $J.r$ its release time, $J.d$ its relative deadline and $J.e$ its worst-case execution time. *Task parallelism* allows jobs to be executed in parallel on different cores/processors. We use in a first step tasks with same start time and deadline (*i.e.* same $J.r$ and $J.d \forall J$).

Reconfiguration This is an operation changing the task system currently executed. It includes *global reconfiguration* (migration) for distributed systems, when there is a memory transfer from one memory to another, to execute a job in a different node (core, processor). We call *local reconfiguration* in shared memory systems, when job is just running in a different processor/core.

III. OPTIMAL SMP REAL-TIME SCHEDULING

Given a set Q of N independant jobs with same release time and deadline on M identical processors sharing

the same memory. We propose a real-time scheduler based on the Lemerre's[7] algorithm with constant time complexity: intra-job parallelism is forbidden here, pre-emption and local reconfiguration are allowed but their costs are not considered.

A. Algorithm description

Q is a deque containing jobs ordered by increasing durations, T_T is the interval duration and we suppose that $N > M$. The first part is a setup phase to set the M smallest jobs of Q on the M processors and remove them from Q . The jobs currently in execution remain active until its ending or until the biggest jobs of Q become urgent (*i.e.* when its laxity are null: see for example job J_8 at $t=1$, J_7 at $t=3$ in Figure 1). We exclusively reserved processors to them by stopping the biggest jobs currently executing (they return then in the first places of Q). This is the main difference between Lemerre's algorithm which preferred to stop the smallest jobs currently executing and migrate them on others processors. We do less operations and reduce preemptions by a factor 2. Our algorithm builds the schedule incrementally, determining the next preemption instant. The next instant is calculated by the minimum between remaining time of the smallest job in execution and laxity of the biggest job (the first value is 1 in example of Figure 1). If there are no urgent jobs, when a job is ending, we replace it by the next first waiting job from Q (see job J_4 replace J_1 at $t=1.5$ in Figure 1).

B. Scheduling example

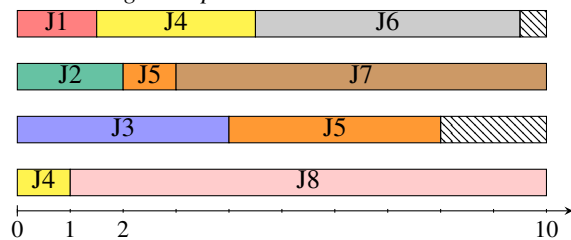


Figure 1: an example of an execution of our scheduler for 8 jobs ($J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8$) sorted by increasing

durations, respectively (1.5,2,4,4,5,5,7,9) (the sorting is made *off-line*) on 4 identical multiprocessors sharing the same memory (there are 2 preemptions and 2 local reconfigurations). Job duration is given by its *weight* multiplying by the interval duration ($T_T=10$).

Property Reconfiguration and preemption only happen when a local laxity of the biggest job is null, but at each time we use exclusively a processor for it, so there is at most $M-1$ preemptions and local reconfigurations. If there are more urgent jobs, it means that the task system was unfeasible.

Algo.	Preemptions	Online complexity	Alloc.
McN.[8]	$\leq M-1$	$O(1)$	static
Level[6]	$\leq N^2(M-1)$	$O(MN^2)$	dynamic
SCH[5]	$\leq M-1$	$O(N+M\log M)$	dynamic
Lem.[7]	$\leq 2M-2$	$O(1)$	dynamic
IZL	$\leq M-1$	$O(1)$	dynamic

Table 1: Optimal scheduling comparative (tasks with same release time and deadline on identical processors; alloc(ation): static, dynamic), our scheduler is IZL (*Incremental Zero Laxity*).

We propose an improvement of Lemerre's[7] algorithm which allows to be as competitive as MacNaughton's one (*bin-packing* approach[8]) for static scheduling and better (in complexity) or similar (in number of preemptions) than Gonzalez's[5] and Horvath's[6] one for dynamic scheduling (see Table 1). We can use the same approach as in [7] to prove our algorithm.

C. Applications to more complex models

We use now the *implicit-deadline periodic* model: a job J_i arriving periodically is characterized by a period p_i , a deadline equivalent to the period and an execution time τ_i : $J_i = (\tau_i, p_i)$. We define the *hyperperiod* as the least common multiple of all jobs periods from the periodic task system Γ , $H_\Gamma = lcm(p_i, \forall J_i \in \Gamma)$. The *job utilization* is $u_i = \tau_i/p_i$.

Given a set of N periodic independent jobs on M identical processors sharing the same memory. We decompose the *hyperperiod* into intervals to schedule job sets. Each interval starts at the end of the last interval and finishes when the first job deadline is met, each interval begins (respectively ends) at *job boundaries*. To construct a list of jobs, it simplifies the problem by setting the *weight* of each released job to its utilization (done off-line, in $O(N)$ time). These *weights* are constant on each interval, only the duration of these intervals vary: we apply our algorithm on each interval which jobs have same release time and deadline.

Pfair class of algorithms[1][2][9] -dealing with periodic task model- has generally M preemptions per time quanta ($M * lcm(p_i) / \min(p_i)$ with BF[9]) and at best linear complexity in run-time (with PD^2 [1]). We propose

an interesting alternative to the Pfair scheduling class (our *online* complexity: $O(M)$ setup time then $O(1)$ per system call, preemptions: $M-1$ max. per interval).

These techniques are not exclusively used for periodic task model and can be extended to time-triggered tasks (such OASIS task model[3]) even if weights are different between two intervals. It is possible because the next boundary job is known (one cannot schedule on-line in multiprocessors architecture without *a priori* knowledge of the next jobs characteristics[4]).

IV. CONCLUSION AND FUTURE WORK

Our goal of our research is to propose real-time energy-efficient scheduling for embedded many-core architecture with more general recurring task model which can be more appropriate for industrial purpose and to take into account the new trends, for example using processor/core groups. The algorithm presented here aims to minimize preemptions in shared memory systems. We are also working on dynamic scheduling with *hierarchical* memory and allowing both *local* and *global reconfiguration*. We would like to define how to evaluate costs for both of them. Another way is to work on *thread parallelism*: it is a part of a job J_i executed simultaneously on several cores/processors sharing the same memory. It allows intra-job parallelism: it is commonly forbidden in real-time scheduling but the trend may be change with parallel computing (OpenMP, MPI). This is a new research field, and it would be interesting to handle with. Moreover, we would take into account fault-tolerance in critical systems, how to define an execution and architecture model allowing error detections including recovering techniques.

REFERENCES

- [1] J. Anderson and A. Srinivasan. *Mixed Pfair/ERfair scheduling of asynchronous periodic tasks*. In *Proceedings of the 13th Euromicro Conference on Real-time Systems*, pages 76-85, June 2001.
- [2] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. *Proportionate progress: a notion of fairness in resource allocation*. In *STOC'93: Proceedings of the 25th annual ACM symposium on Theory of computing*, pages 345-354, 1993.
- [3] D. Chabrol, V. David, C. Aussaguès, S. Louise and F. Daumas, *Deterministic distributed safety-critical real-time systems within the oasis approach*. In *17th IASTED PDCS'05*, 2005.
- [4] M.L. Dertouzos and A.K. Mok, *Multiprocessor on-line scheduling of hard-real-time tasks*. *IEEE Transactions on Software Engineering*, vol. 15, n12, pp. 1497-1506, 1989.
- [5] T. Gonzalez and S. Sahni. *Preemptive scheduling of uniform processor systems*. *J. ACM*, 25(1):92-101, 1978.
- [6] E. C. Horvath, S. Lam, and R. Sethi. *A level algorithm for preemptive scheduling*. *J. ACM*, 24(1):32-43, 1977.
- [7] M. Lemerre, V. David, C. Aussaguès and G. Vidal-Naquet, *Equivalence between schedule representations: theory and applications*. In *Proceedings of the 14th IEEE RTAS'08*, 2008.
- [8] R. McNaughton, *Scheduling with deadlines and loss functions*, In *Management Science*, 1959.
- [9] D. Zhu, D. Moss and R. Melhem. *Multiple-resource periodic scheduling problem: how much fairness is necessary?* In *RTSS'03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, page 142, 2003.