HAL
open science

Using Discrete Geometry to model PFair scheduling algorithm for Real-Time systems applications

Gaëlle Largeteau-Skapin, Annie Choquet-Geniet, Abdoulaye Ouattara

To cite this version:

Gaëlle Largeteau-Skapin, Annie Choquet-Geniet, Abdoulaye Ouattara. Using Discrete Geometry to model PFair scheduling algorithm for Real-Time systems applications. 2009. hal-00381677

HAL Id: hal-00381677

https://hal.science/hal-00381677

Preprint submitted on 6 May 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Using Discrete Geometry to model PFair scheduling algorithm for Real-Time systems applications

XLIM-SIC Internal report 2009-001

April 12th, 2009.

Gaëlle Largeteau-Skapin(1), Annie Choquet-Geniet(2), and Abdoulaye Ouattara(3)

1. Laboratoire SIC, Université de Poitiers,
BP 30179 86962 Futuroscope Chasseneuil cédex, France
2. Laboratoire LISI, ENSMA
Téléport 2 - 1 avenue Clément Ader
BP 40109 86961 Futuroscope Chasseneuil cedex, France
3. Université Polytechnique de Bobo-Dioulasso
01 BP 1091 Bobo-Dioulasso 01, Burkina-Faso
glargeteau@sic.univ-poitiers.fr, ageniet@ensma.fr.

Abstract. In this paper, we focus on the use of discrete geometry for the sake of real-time modeling and analysis. We consider multiprocessor context, and we determine the geometrical characterization of PFair scheduling algorithms, which are known to be very performant strategies. A feasability test can then be deduced from the geometrical properties.
Keywords: discrete geometry, real-time, scheduling, model.

1 Introduction

Real-time applications are dedicated to the control of (critical) physical processes, e.g. nuclear plants, car ignition systems, planes, robots... The application interacts with the process by means of sensors and actuators. The general aim is to ensure the process safety. For that sake, not only algorithmic correctness is required but also temporal correctness: the dynamic of the application must be suited to the dynamic of the physical process. A real-time application is classicaly modeled as a set of periodic tasks, dedicated to control, e.g. temperature acquisition in a nuclear station, robot's trajectory computation, processing of information provided by a synchronous link... Tasks are submitted to firm

deadlines, which must be respected, otherwise the application may have erroneous behavior. For instance, a late computed value can be obsolete, and using it may be misleading, or even dangerous. The main challenge for systems designers is to guarantee that all deadlines will be met. That is the concern of scheduling.

We consider here multiprocessor architectures and global scheduling: tasks can run at any time on any processor, they are never definitively assigned to a processor, and may start on one processor and resume on another. We also assume parallelism to be forbidden: at any time, a task runs on at most on one processor. If tasks are independent, synchronous (they all are first released at the same time) with implicit deadline (deadlines are equal to periods), there exist optimal[1] scheduling algorithms, the PFair algorithms [4, 6]. But in larger contexts, where tasks are not indenpendent, it has been proven that there exists no optimal strategy [10]. The designer can then turn towards off-line scheduling. A feasible schedule is computed before run-time, and is stored in a table. Off-line scheduling is more powerful that on-line scheduling, since scheduling decisions are made according to the instantaneous state of the application for on-line strategies meanwhile they are based on a global knowledge of the application for off-line scheduling. These approaches are mostly model-oriented. Approaches based on finite automata and on Petri nets can be found in the litterature [11, 12]. We consider here an approach based on discrete geometry. Such an approach has a very lower complexity than the former approaches, as well for the model generation as for the model analysis.

In [13], we have defined a discrete geometrical model equivalent to the automata based model presented in [9]. In this model, each task is associated with a $2D$-discrete object that collects all valid states of the task, i.e. all the states *(cumulated processed execution time of the task, time)* belonging to a valid schedule. The shape of this 2D-discrete object depends only on time characteristics of the task. The execution of the task set is then modeled by a $n+1$-discrete object (where n is the number of tasks). This model is called Concurrency Model and is built using extrusion and intersection of task models. Ressource sharing is modeled using a nD-discrete space, extruded following the time direction and then cut out from the concurrency model (see figure 3). We have proven in [13] that there exists a feasible schedule on a m-processor architecture if and only if there exists a m-connected path in the $(n+1)$D-discrete model of the application.

Our aim is here to lay the basis for a geometrical characterization of PFair schedules which can be used for applications composed of dependent tasks with deadlines less than periods. This model can then be used to decide effectively if PFair scheduling is feasible in that extended context.

The paper is organized as follows: in section 2, we introduce basic notions and notations in discrete geometry and in real-time scheduling. In section 3, we present the modeling of PFair algorithms. In section 4, we present the model implementation within the geometrical software GRETA. The paper ends with some concluding remarks.

[1] A scheduling algorithm is said to be optimal if either it produces a feasible schedule, i.e. a schedule such that all deadlines are met, or there exists no feasible schedule.

2 Preliminaries

2.1 Basic notations in discrete geometry

The following notations correspond to those given by Cohen and Kaufman in [8], by Klette and Rosenfeld in [15] and those given by Andres in [3]. We provide only a short overview of these notions.

A **discrete** (resp. **Euclidean**) **point** is an element of $\mathbb{Z}^n$ (resp. $\mathbb{R}^n$). A **discrete** (resp. **Euclidean**) **object** is a set of discrete (resp. Euclidean) points. We note p_i the ith coordinate of a point p of $\mathbb{Z}^n$. Two discrete points p and q are k-neighbours (or k-connected), with $0 \leq k \leq n$, if $|p_i - q_i| \leq 1$ for $1 \leq i \leq n$, and $k \leq n - \sum_{i-1}^{n} |p_i - q_i|$. The set $D(a, b, \gamma, \omega) = \{(x, y) \in \mathbb{Z}^2, \gamma \leq ax + by < \gamma + \omega\}$ is called a digital straight line with slope a/b, lower bound γ and arithmetical width ω.

2.2 Basic notions of real-time scheduling

We consider multiprocessor systems. For any real x, $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x and $\lceil x \rceil$ the smallest integer greater than or equal to x.

The task model We consider applications composed of n independent periodic tasks $\tau_1(r_1, C_1, D_1, P_1)$, $\tau_2(r_2, C_2, D_2, P_2)$, ..., $\tau_n(r_n, C_n, D_n, P_n)$. Each task is submitted to hard temporal constraints. We adopt the classical modeling of tasks [14]. Each **periodic** task τ_i is characterized by four temporal parameters as described in figure 1: r_i is the first release date or offset; C_i the worst-case execution time; D_i the relative deadline, which corresponds to the maximal delay allowed between the release and the completion of any instance of the task; and P_i the period. Each task τ_i consists in an infinite set of instances (or jobs),

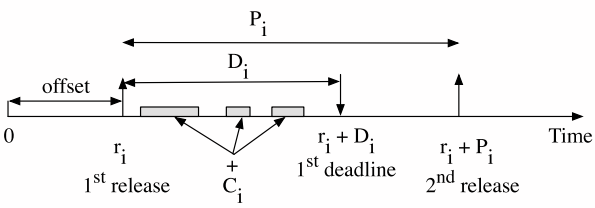

Fig. 1. Temporal modeling of a real time periodic task

released at times $r_i + k \times P_i$, with k $\in \mathbb{N}$. We assume that temporal parameters are known and determinist.
In the sequel, P denotes the **hyperperiod** of the system defined as $P = \text{lcm}(P_1,$

$P_2, \ldots, P_n$).
The processor's **utilization factor** characterizes the processor workload due to the application. It is defined by $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$. If U > m (m being the number of processors), the system is over-loaded and temporal faults cannot be avoided [7]. In the sequel, we suppose it to be less than m.
In the further, **slot** t denotes the time interval $[t, t+1)$. A task is said to be **scheduled at time** t when that one processor processes it during slot t.
A **schedule** is defined by $S : \mathbb{N} \times \{1, \ldots, n\} \rightarrow \{0, 1\}$ such that $\sum_{i=1}^{n} S(t, i) \leq m$.
We have $S(t, i) = 1 \Leftrightarrow \tau_i$ is scheduled at time t, for $i = 1 \ldots n$ and if $\sum_{i=1}^{n} S(t, i) = k < m$ then $(m - k)$ idle time units occur at time t. We also introduce S_i defined by

$$S_i(t) = \begin{cases} 1 \text{ if S(t, i)} = 1 \\ 0 \text{ else} \end{cases}$$

And for any times t and t', and for any task τ_i, we define $W_i(t, t')$ as the **processed execution time** for task τ_i between time t and time t'. We have $W_i(t, t') = \sum_{u=t}^{t'-1} S_i(u)$.

2.3 Geometric model for real-time systems behaviors

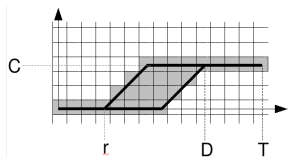

Fig. 2. Geometrical model $\Omega(\tau)$ of task $\tau(r = 3, C = 3, D = 7, T = 11)$.

As shown on figure 2, the single task model Γ_i is an object of 2D space: the (task processed execution time, time) space. The shape of this 2D-discrete object depends on the temporal parameters (r, C, D, T) of the task.

The model for the whole application $\Gamma = (\tau_i)_{i \in [1,n]}$ is the set

$$\Omega(\Gamma) = \{(x_1, ...x_n, t) \in \mathbb{Z}^{n+1}, \forall i \in [1, n], (x_i, t) \in \Gamma_i\}.$$

Diagrams (a) to (d) on figure 3 present the building steps for the concurrency model. It is obtained using extrusion and intersection operations.

Resource sharing between tasks τ_i and τ_j is modeled by a surface (noted $G(R)$ on figure 3) in the (x_i, x_j) plan which is then extruded following time

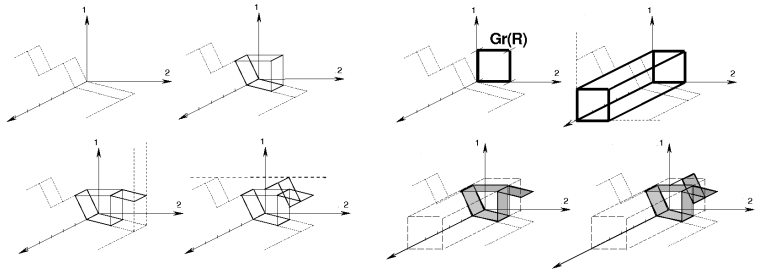

Fig. 3. building steps of a two task system geometric model.

direction. The result is finaly cut out from Γ_i to obtain $\Gamma_{i,R}$. Those steps are depicted by diagrams (e) to (h) on figure 3.

Feasability of a real-time system application relies on the connectivity of the resulting $(n+1)$D-object: if it is m-connected then there exists at least one valid schedule of the application on a m-processor architecture. If not, we can measure the distance to feasability by computing the distance between connected components. Please refer to [13] for a complete definition of the geometric model.

3 Discrete modelisation of PFair algorithm

In this part, we present a PFair algorithm and the associated discrete geometric model in the context of synchronous tasks with implicit deadlines($\forall i,\ r_i = 0$ and $D_i = T_i$). Then we model asynchronous task systems while relaxing the $r_i = 0$ constraint. Then we consider systems with deadlines $D_i \leq T_i$ and finally we take concurrency and resource sharing into account.

3.1 PFair scheduling algorithm

PFair scheduling strategies have been proposed in the general multiprocessor context, for which they are very efficient. The basic idea is that each task is processed at "regular rate". This means that at each time t, the number of processed slots $W_i(0,t)$ is proportionnal to t, with coefficient $u_i = \frac{C_i}{P_i}$. But, since the number of processed slots at time t must be integer, $u_i \times t$ is approximated by either $\lfloor u_i \times t \rfloor$ or $\lceil u_i \times t \rceil$.
This is formally expressed by the following definition:
A schedule is **PFair** iff we have:

$$\forall t \in \mathbb{N}, -1 < u_i \times t - \sum_{j=0}^{t-1} S_i(j) < 1$$

Figure 4 illustrates PFairness. For any task τ_i, the broken line W_i must remain strictly between both limit lines $W^- = u_i \times t - 1$ and $W^+ = u_i \times t + 1$.

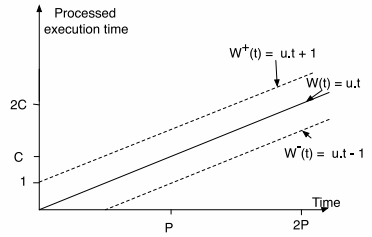

Fig. 4. PFair execution of a task: the execution curve must be located between both dotted lines

Figure 5 shows examples of PFair and non PFair behaviors.

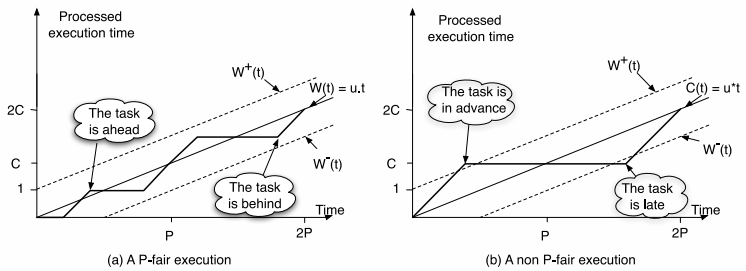

Fig. 5. (a) a PFair execution and (b) a non PFair executions, and different status of the task

At any time t, a task is said to be:

- **ahead** if $W_i(t)$ is above the ideal line $W_i(t) = u_i \times t$. It has been processed a little bit more than in the ideal case. We have: $u_i \times t - \sum_{j=0}^{t-1} S_i(j) < 0$.
- **punctual** if it has been processed for exactly $u_i \times t$ slots. We have: $u_i \times t - \sum_{j=0}^{t-1} S_i(j) = 0$.
- **behind** if $W_i(t)$ is under the ideal line $W_i(t) = u_i \times t$. It has been processed a little bit less than in the ideal case. We have: $u_i \times t - \sum_{j=0}^{t-1} S_i(j) > 0$.

PFair strategies follow the global frame described below.

1. The task set is partitioned into three sets.
   - the **Urgent** set collects all the behind tasks which would be late (under the lower bound) if they were not processed at time t. These tasks must be processed at time t, else the PFairness condition would be violated.

- the **Tnegru** set collects the ahead tasks which would be in advance (over the upper bound) if they were processed at time t. These tasks must not be processed at time t, else the PFairness condition would be violated.
- the **Contending** set collects the other tasks: the PFairness won't be violated neither if they are processed nor if they are not.

2. Urgent tasks are processed.
3. Contending tasks are sorted. The m - $\mid Urgent(t) \mid$ first contending tasks are processed[2].

Several PFair versions have been proposed in the litterature (PF, PD and PD^2 [6, 5, 1, 2]). These algorihms differ in the way they select tasks to process among the contending tasks. Moreover, in PD^2, Tnegru tasks may also be processed, provided some processors are available. It is a conservative version of PFair strategies. These scheduling strategies are very efficient, as stated in theorem 1.

Theorem 1 *[6] The scheduling algorithms PF, PD and PD^2 are optimal for systems of periodic synchronous independent tasks with implicit deadlines (deadlines are equal to periods) in multiprocessor context. Moreover, the system is feasible if and only if $U \leq m$ where m is the number of processors.*

For multiprocessor systems, Pfair scheduling strategies are the only known optimal strategies.

3.2 PFair modeling

We define the PFair geometrical model for a task $\tau(r, C, D, T)$ as the set of PFair states belonging to a PFair schedule.

Definition 1. *The PFair geometric model of a task τ is the set:*

$$V(\tau) = \{(t, y) \in \mathbb{N} \text{ such that } \exists \text{ a PFair schedule } S,\ y = \textstyle\sum_{k \in [0,t]} S_i(k)\}$$

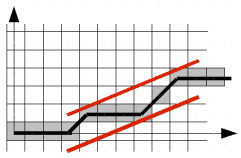

Fig. 6. A PFair schedule of task $\tau(r = 3, C = 3, D = 7, T = 11)$.

Figure 6 presents the PFair execution schedule (_ _ _ p _ _ _ p p _ _ _ _ _) for task $\tau(3, 3, 7, 11)$ where p stands for processing and _ for idle. In the following, we define this set on a more geometrical way.

[2] $\mid A \mid$ denotes the cardinality of set A.

3.3 Synchronous task systems with implicit deadlines

The discrete model of the PFair behaviors of a real-time task is composed of all states the application can reach under PFair execution. This set consists of all discrete points close enough (at a distance less than 1) to the line $y = (C/T)t$.

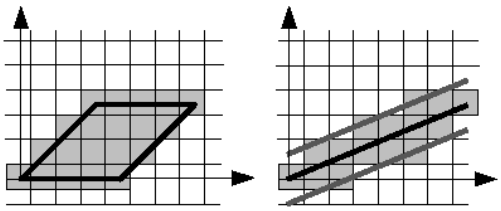

Fig. 7. Geometrical model Γ and PFair state set V for task $\tau(r = 0, C = 3, D = 7, T = 7)$.

We can see on figure 7 the geometrical model Γ of task $\tau(r = 0, C = 3, D = 7, T = 7)$ and the subset V of Γ composed of τ PFair states. Let us state $y = \sum_{j=0}^{t-1} S_i(j)$. From PFairness definition, we have $\forall t \in \mathbb{N}, -1 < \frac{C_i}{T_i} \times t - y < 1$, so we get: $\forall t \in \mathbb{N}, -T_i < C_i \times t - y \times T_i < T_i$. We therefore define the set V geometricaly as:

Definition 2. *The set of discrete PFair states for a task $\tau(0, C, D = T, T)$ is defined by:*

$$V(\tau) = \{(t, y) \in \mathbb{Z}^2, -T < Ct - Dy < T\}$$

Since PFair states belong to feasible schedules, we also have the following property:

$$V(\tau) \subset \Omega(\tau)$$

Asynchronous task systems with implicit deadlines. In this context, a task cannot be processed before its first release. Therefore, the only states available before this time are those with $y = 0$. After release, the execution must follow the line $y = \frac{C}{D} \times t + r$. Thus we give the following definition:

Definition 3. *The set of discrete PFair states for a task $\tau(r \neq 0, C, D = T, T)$ is:*

$$V = \left\{ (t, y) \in \mathbb{Z}^2, \begin{array}{l} \forall t < r, y = 0, \\ \forall t \geq rC \times r - T < Ct - Ty < C \times r + T \end{array} \right\}.$$

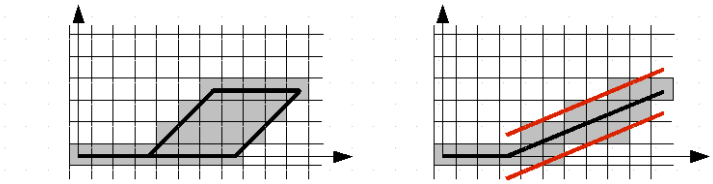

Fig. 8. Geometrical model and PFair-state set V for task $(r = 3, C = 3, D = 7, T = 7)$.

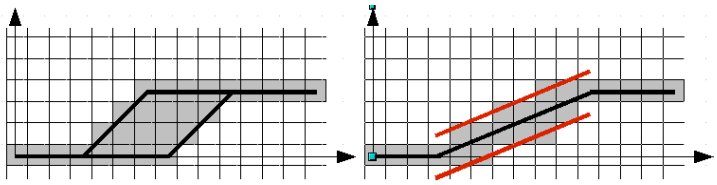

Fig. 9. Geometrical model and PFair-state set V for task $(r = 3, C = 3, D = 7, T = 11)$.

Task systems with relative deadlines. When the relative deadline is strictly less than the period, the task must complete execution before deadline and no execution can arise between D and T. The task must be processed during C slots between r and $r + D$. So the execution rate is here equal to $\frac{C}{D}$, and the task is idle before r and between D and T. We thus have:

Definition 4. *The set of discrete PFair states for a task $\tau(r \neq 0, C, D \leq T, T)$ on its first period is defined by:*

$$V_1 = \left\{ (t, y) \in \mathbb{Z}^2, \begin{array}{c} \forall t \leq r, y = 0, \\ \forall t \in [r, D[, C \times r - D < Ct - Dy < C \times r + D, \\ \forall t \in [D, T[, y = C \end{array} \right\}.$$

The next period starts with the next job release at T. We therefore represent the PFair behaviors during time interval $[0,\ k \times T]$ using union of lines $(y = \frac{C}{D} \times t + (i-1) \times T)_{1 \leq i \leq k}$ digitisation. We state $r_0 = r$, $r_i = (i-1) \times T$ and $D_i = r_i + D$.

Definition 5. *The set of discrete PFair states for a task $\tau(r \neq 0, C, D \leq T, T)$ on its ith period $(i > 1)$ is defined by:*

$$V_i = \left\{ (t, y) \in \mathbb{Z}^2, \begin{array}{l} \forall t \in [r_i, D_i[, C \times r_i - D < Ct - Dy < C \times r_i + D\}, \\ \forall t \in [D_i, r_{i+1}[, y = C \times i \end{array} \right\}.$$

The complete set is therefore:

Definition 6. *The set of discrete PFair states for a task* $\tau(r \neq 0, C, D \leq T, T)$ *on its k first periods is:*

$$V = \bigcup_{1 \leq i \leq k} (V_i).$$

Concurrency and resource sharing. The model for PFair states of the whole system is obtained using extrusion of each PFair model of each task followed by their intersection [13], we get the following set:

Definition 7. *The set of discrete PFair states of the task system* $(\tau_i)_{i \in [1,n]}$ *is the set*

$$V_\Gamma = \{(t, y_1, ...y_n) \in \mathbb{Z}^{n+1} \text{ such that } \forall i \in [1, n], (t, y_i) \in V_{\tau_i}\}$$

The feasability of a system is determined by the connectivity of the final object. If there exists an m-connected path (set of discrete m-neighbourg points of the object such that there exists one and only one point for each t coordinate) through the object then there exists a schedule which guarantees that all deadlines are met.

The geometric modeling of the PFair on-line algorithm allows several analysis: we can decide if there exists a PFair schedule, for a given number of processors m, by searching a m-connected path in the model. If such a m-connected path does not exist, we can assure that there exists a resource sharing problem (see theoreme 1). Since the model integrates both PFainess and resource management,we can also determine whether there exists a schedule that both is PFair and respects resource scharing constraints for a given number of processors. If no such schedule exists, it can be possible to find a connected path that is partly PFair before and after the resource sharing problem and thus bypasses the ressource sharing impossible states. It can be useful in order to propose a bi-modal schedule: PFair mixed with a specific non-Pfair behavior which is compatible with the resources management. We can therefore also compute distance to PFairness of a schedule for a m processor architecture. This distance is defined as the length of the shortest m-connected path linking the last PFair state before the resource sharing problem and the first PFair state after.

4 implementation

The software GRETA has been developped 4 years ago. This software computes the geometrical model of a task system where tasks can be asynchronous, with relative deadlines ($D < T$). The ressources sharing is also implemented.

The PFair model is computed in the same way as the general model, we just have to add a further test on each point to decide if it belongs to the model or if it does not. On figure 10, we can see the PFair model of a two task system. In the obtained discrete object, a Pfair schedule can be computed as a list of states (voxels)(see light grey voxels on figure 10).

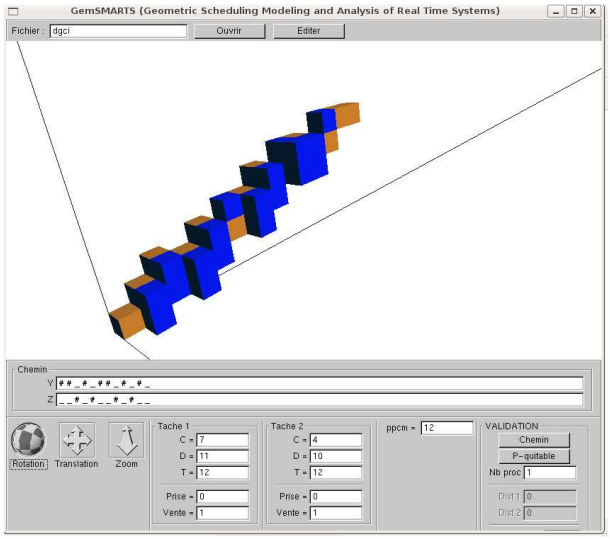

Fig. 10. GRETA result of PFair modelling for a two task system.

5 Conclusion

PFair scheduling is a very powerful strategy for real-time scheduling in multiprocessor context. It has been proven to be optimal for synchronous, independent task systems, with implicit deadlines. It could thus be of interest to use it for larger context. Besides, the discrete geometrical approach is really effective for deciding feasability and finding feasible schedules. We have here paired both notions. We have presented the geometrical characterization of PFair behaviors of a real-time application. We have then explained how to take resources management constraints into account in our model. The geometrical modeling has been implemented within the software GRETA.

Next step will be the extraction of as Pfair as possible schedules. Furthermore, it could be interesting to try to derive a resource management protocol which is as close as possible from PFair behavior. A work in progress is the definition of quality criteria based on geometrical properties (mainly distances) that allow to quantify the "PFairness" of a given schedule.

References

1. J. Anderson, A. Block, and A. Srinivasan. Pfair scheduling : Beyond periodic task sytems. In *Proceedings of the* 12th *Euromicro Conference on Real-Time Systems*, pages 35–43. Chapman and Hall, 2000.

2. J. Anderson, P. Holman, and A. Srinivasan. Fair scheduling of real time tasks on multiprocessors. *Handbook of scheduling : Algorithms, Models and Performance analysis*, pages 31.1–31.21, 2004.
3. E. Andres, R. Menon, R. Acharya, and C Sibata. Discrete linear objects in dimension n : The standard model. *Graphical Models*, 65:92–111, 2003.
4. S. Baruah. Fairness in periodic real-time scheduling. In *Proceedings of the* 16th *IEEE Real-Time Systems Symposium*, pages 200–209, 1995.
5. S. Baruah, J. Gehrke, and C.G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the* 9th *International Parallel Processing Symposium*, pages 280–288, April 1995.
6. S.K. Baruah, N.K. Cohen, C.G. Plaxton, and D.A. Varvel. Proportionate progress : a notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.
7. G.C. Buttazzo. *Hard Real-Time Computing Systems.* Kluwer Academic Publishers, 1997.
8. D. Cohen-Or and A Kaufman. Fundamentals of surface voxelization. *Graphical Models and Image Processing*, 57(6):453–461, November 1995.
9. Geniet D. and Largeteau-Skapin G. Wcet free time analysis of hard real-time systems on multi-processors: a regular language-based model. *Thoretical Computer Science*, 388:26–52, 2007.
10. M. Dertouzos and A.K. Mok. Multiprocessor on-line scheduling of hard real-time tasks. *IEEE Transactions Software Engineering*, 12(15):1497–1506, 1989.
11. E. Grolleau and A. Choquet-Geniet. Off line computation of real time schedules by means of petri nets. In R. Boel and G. Stremersch, editors, *Discrete events systems*, pages 309–316. Kluwer Academic Publishers, 2000.
12. G. Largeteau, D. Geniet, and J.P. Dubernard. Validation of distributed periodic real-time systems using can protocol with finite automata. In *Proc. of* 5th *S.C.I.* I.I.I.S., 2001.
13. G. Largeteau-skapin, D. Geniet, and E. Andres. Discrete geometry applied in hard real-time systems validation. In *Discrete Geometry for Computer Imagery*, LNCS, pages 22–33, Szeged, Hungary, 2005.
14. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
15. A. Rosenfeld and R. Klette. Digital straightness. In *Int. Workshop on Combinatorial Image Analysis*, Electronic Notes in Theoretical Computer Science, pages 1–32, 2001.