Quantification du taux d'invalidité d'applications temps réel : une approche géométrique

Gaëlle Largeteau-Skapin, Dominique Geniet

Quantification du taux d'invalidité d'applications temps réel : une approche géométrique.

Gaëlle Largeteau, Dominique Geniet
Laboratoire d'Informatique Scientifique et Industrielle,
Université de Poitiers & ENSMA,
Téléport 2 - 1 avenue Clément Ader
BP 40109 86961 Futuroscope Chasseneuil cédex, France
tèl : 05-49-49-80-62.
fax : 05-49-49-80-64.
glargeteau@yahoo.fr, dominique.geniet@laposte.net

Résumé

Ce travail s'intègre dans un projet de conception d'outils d'aide à la spécification d'applications temps réel à contraintes strictes : il s'agit, lorsqu'une application est invalide, d'identifier les causes de l'invalidité et de proposer des solutions. Pour ce faire, nous étudions ici la définition de mesures d'invalidité. Nous nous basons sur un modèle géométrique pour les applications temps réel. Une mesure est définie à partir de ce modèle sur la base d'une distance géométrique. Les systèmes que nous traitons utilisent des ressources critiques et fonctionnent sur des architectures multiprocesseurs. Les tâches sont périodiques, non réentrantes, à départs différés.

Plan

Mots clés

temps réel, validation temporelle, multiprocesseurs, partage de ressources, géométrie.

1 Introduction

Une application temps réel est destinée à piloter un procédé et diffère d'un système classique par l'obligation qu'elle a de respecter des contraintes temporelles [Sta88]. Ces contraintes peuvent être strictes si leur non respect implique des conséquences graves au sens où, par exemple, des vies humaines sont en jeu. Dans ce cas, le système est *dur* [But97]. Si, au contraire, le non respect des échéances ne met pas en péril la vie du système, le système est *mou*. Dans notre étude, nous considérons uniquement les systèmes durs.

Un système temps réel est un ensemble de tâches codant chacune une réaction de l'application à un événement. Les systèmes que nous considérons utilisent des ressources et fonctionnent sur une architecture multiprocesseurs dans laquelle tous les processeurs sont identiques. Les tâches peuvent être préemptées et peuvent migrer. Chaque tâche τ_i est spécifiée par des caractéristiques temporelles : une date de réveil r_i, un délai critique D_i, une période T_i, et un temps d'exécution fixe C_i (temps d'exécution du pire cas, WCET [CP00]). Nous supposons qu'aucune tâche n'est réentrante ($\forall i, D_i \leq T_i$).

Valider une application temps réel consiste à prouver qu'elle peut fonctionner en régime permanent sans jamais commettre de faute temporelle. Les techniques de validation actuelles proposent une réponse binaire (oui/non) au problème de l'ordonnançabilité [CHO96][SSR98]. Nous souhaitons raffiner cette réponse dans le cas négatif : si une application est détectée comme *très* invalide, les modifications à apporter à sa spécification risque d'être importantes. Si, au contraire, l'application est *peu* invalide, un changement mineur de la spécification permet sans doute de valider l'application. Quantifier ainsi la non-ordonnançabilité permet de proposer au concepteur un supplément d'information qu'il peut exploiter pour concevoir son application, de façon à ce qu'elle corresponde au cahier des charges tout en étant valide du point de vue temporel.

Dans cette étude, nous associons à chaque tâche un objet dont la géométrie dépend uniquement de ses caractéristiques temporelles. L'objet associé à une application possède $n+1$ dimensions, n étant le nombre de tâches [LCG04]. La mesure que nous définissons est basée sur une distance géométrique.

Dans un premier temps, nous définissons le modèle géométrique. Nous définissons ensuite une mesure d'invalidité. Nous présentons enfin quelques résultats obtenus après exérimentations de cette mesure.

2 Mesure d'invalidité : objectifs et spécification

2.1 Objectifs

La validation d'application temps réel consiste à prouver l'existence d'une séquence d'ordonnancement en utilisant des tests d'ordonnançabilité ou une simulation. Lorsqu'une application est ordonnançable, les modèles hors-ligne fournissent souvent, de manière exhaustive, l'ensemble des ordonnancements possibles permettant ainsi une analyse plus poussée sur les caractéristiques secondaires telles que la régularité de la gigue, les temps de réponse, etc... Dans le cas d'une application non ordonnançable, le modèle est souvent vide (il n'existe aucun ordonnancement possible) et on ne peut rien dire de plus.

Une mesure d'invalidité est un outil permettant de quantifier l'effort nécessaire pour rendre l'application ordonnançable. Cette mesure peut ensuite être utilisée par le concepteur de l'application pour modifier la spécification de façon à ce que la configuration de tâches devienne ordonnançable. L'objet de notre travail est de définir une telle mesure puis de l'expérimenter.

La section 2.2 définit les principales propriétés que doit vérifier une mesure d'invalidité.

2.2 Spécification de la mesure

Une mesure est un réel positif, déstiné à quantifier une propriété donnée. Ici, nous voulons évaluer l'effort à fournir pour rendre une application ordonnançable. Nous associons donc la valeur 0 à une application valide et une valeur plus grande que 0 à une application invalide. Une mesure d'invalidité doit tenir compte du contexte d'ordonnancement. En effet, une application non ordonnançable sur un seul processeur peut l'être sur deux. Dans ce cas, la mesure d'invalidité pour un processeur doit être supérieure à 0 et la mesure pour deux processeurs doit être nulle. La mesure d'invalidité dépend donc du nombre de processeurs. Nous notons $\mathrm{M}_p(\Gamma)$ la mesure d'invalidité de la configuration Γ en considérant un contexte à p processeurs.

Nous notons $\mathcal{A}$ l'ensemble des configurations de tâches et nous notons $\mathcal{C}$ un contexte d'ordonnancement[1] à k processeurs. Nous notons $\mathcal{O}_C(\Gamma)$ l'ensemble des ordonnancements valides dans le contexte $\mathcal{C}$.

Une mesure d'invalidité définit une relation d'ordre *"au plus aussi valide que"* dans le contexte $\mathcal{C}$ (notée $\prec_{\mathcal{C}}$) sur l'ensemble $\mathcal{A}$: soient A et B $\in \mathcal{A}$,

$$A \prec_{\mathcal{C}} B \Leftrightarrow M_k(A) \leq M_k(B).$$

La première propriété d'une mesure d'invalidité est qu'elle doit être nulle pour

[1] Un contexte d'ordonnancement décrit l'architecture matérielle ainsi que le type de tâches étudiées

les applications valides et non nulle pour les applications invalides.

$$M_k(\Gamma) > 0 \Leftrightarrow \mathcal{O}_C(\Gamma) = \emptyset.$$
$$M_k(\Gamma) = 0 \Leftrightarrow \mathcal{O}_C(\Gamma) \neq \emptyset.$$

De plus, s'il n'existe aucune séquence valide pour deux processeurs, il n'en existe pas pour un seul processeur : l'application est *"moins"* ordonnançable sur un processeur que sur deux. En effet, les modifications à apporter dans ce cas sont d'une part celles que l'on aurait dû effectuer dans le cas bi-processeurs et d'autre part une augmentation du nombre de processeurs. Il y a donc plus de modifications à apporter dans le cas où on ne dispose que d'un processeur.

La mesure doit donc avoir la propriété suivante :

$$M_p(\Gamma) > 0 \Rightarrow \forall k < p,\ M_k(\Gamma) \geq M_p(\Gamma).$$

Une application ordonnançable sur deux processeurs l'est sur toute architecture comportant plus de deux processeurs. La mesure doit donc vérifier :

$$M_p(\Gamma) = 0 \Rightarrow \forall k > p,\ M_k(\Gamma) = 0.$$

3 Définitions des modèles de représentation

Nous souhaitons valider temporellement une application temps réel, i.e. savoir si toutes les tâches s'exécutent avant leur échéance. Nous avons donc besoin de connaître l'état d'avancement de chaque tâche composant l'application en fonction du temps.

3.1 Espace de représentation des états d'une tâche

L'état d'une instance de tâche est défini par son avancement x et le temps t qui s'est écoulé depuis son activation. Nous considérons donc, pour chaque tâche, un espace E à 2 dimensions : la première correspond au temps, la seconde à l'avancement de l'instance (temps CPU déjà consommé).

Lors de l'exécution d'une instance d'une tâche τ, celle-ci passe par un certain nombre d'états suivant qu'un processeur lui est attribué ou non. L'état initial e_0 de la première instance de cette tâche est l'état $(r_i, 0)$, son dernier état est l'état $(r_i + T, C)$. L'état final d'une instance est aussi l'état initial de l'instance suivante : l'état initial e_k de la $k^{\text{ème}}$ instance de la tâche est $(e_{k-1} + (T, C))$. Une exécution de la tâche est donc complètement définie par la suite des états atteints par ses instances successives, qui constitue une courbe de l'espace (temps,temps CPU). Cette courbe (et par abus de langage sa description analytique) sera appelée *"trajectoire de la tâche"*.

Lors de son exécution, la tâche est définie à chaque instant par son état. Elle ne peut être simultanément dans plusieurs états et son état ne peut pas

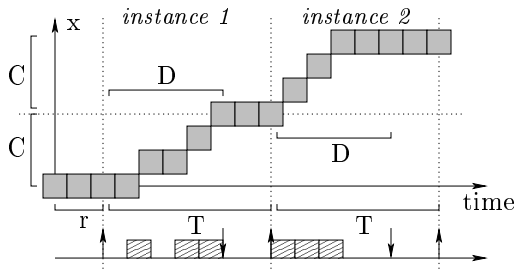


FIG. 1 – Une trajectoire de la tâche τ (r=2,C=3,D=5,T=7).

être indéfini. La trajectoire est donc une application. Cette application est croissante car soit la tâche s'exécute et son état d'avancement augmente, soit elle est suspendue et son état d'avancement reste le même. De plus, durant un intervalle $[t, t+\epsilon]$, la tâche ne peut pas avancer dans son exécution de plus de ϵ puisque c'est le temps maximum qui peut lui être alloué dans l'intervalle considéré. La tâche τ doit respecter sa spécification temporelle (r, C, D, T) : Sur la trajectoire d'une exécution de τ, ceci se traduit par un certain nombre de contraintes géométriques. Nous notons r_i=$r + i \times T$ le réveil de la $i^{\text{ème}}$ instance de la tâche et $r_i + D$= $r + i \times T + D$ son échéance.

Définition 1 *On appelle* ***trajectoire valide*** *de τ (r, C, D, T) toute trajectoire Tr_τ ayant les propriétés suivantes (voir figure 1) :*

$$
\begin{array}{lllll}
Tr_\tau V & : & \mathbb{N}^+ & \rightarrow & \mathbb{N}^+ \\
 & & Tr_\tau V([0,r]) & = & \{0\} \\
 & & Tr_\tau V([r_i, r_i + D]) & = & [(i-1) \times C, i \times C] \\
 & & Tr_\tau V([r_i + D, r_{i+1}]) & = & \{i \times C\}
\end{array}
$$

On appelle $\mathrm{TV}(\tau)$ l'ensemble des trajectoires valides pour τ. Chaque exécution de la tâche en dehors de l'intervalle $[r_i, r_i + D]$ constitue une faute temporelle, c'est pourquoi les applications $\mathrm{Tr}_\tau \mathrm{V}$ sont constantes sur les intervalles $[0, r]$ et $[r_i + D, r_{i+1}]$. La valeur $\mathrm{TrV}_\tau(r_i + D)$=$i \times C$ garantit l'exécution de la totalité du code de l'instance i avant son échéance $r_i + D$ et ce pour chaque instance i de la tâche. Il n'y a donc pas de faute temporelle dans la séquence d'exécution de τ associée à une trajectoire de cette forme. Le domaine formé des points des trajectoires valides est appelé "domaine de validité" de la tâche.

Définition 2 *Le domaine de validité $\Omega(\tau)$ d'une tâche τ est l'ensemble de ses états valides :*

$$\Omega(\tau) = \bigcup_{\psi \in TV(\tau)} \{(t, \psi(t))\}.$$

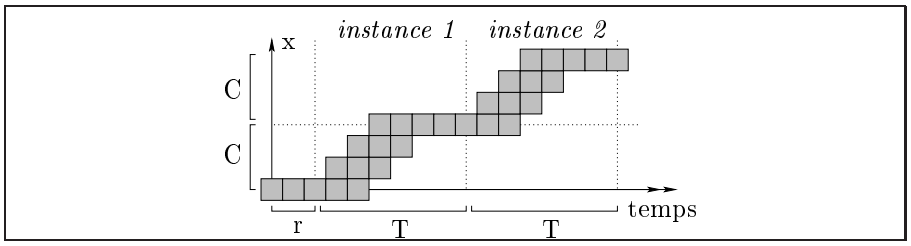


FIG. 2 – Espace 2-dimension pour la représentation de deux instances consécutives d'une tâche τ (r, C, D, T).

L'ensemble $\Omega(\tau)$ associé à τ est l'ensemble des points faisant partie d'une trajectoire valide de τ, i.e. chaque point est une représentation géométrique d'un état valide temporellement et atteignable de τ.

La figure 2 présente l'ensemble des états valides restreint aux deux premières instances d'une tâche.

3.2 Expression du parallélisme

L'espace de repésentation d'une application est en $n+1$ dimensions : une dimension pour l'avancement de chaque tâche et la dimension du temps commune à toutes les tâches. Chaque tâche de l'application est définie par un ensemble de points 2-D dans son plan (le plan (temps, τ_i)). Ces points sont complétés [LCG04] à $(n+1)$ dimensions (cf. figure 3).

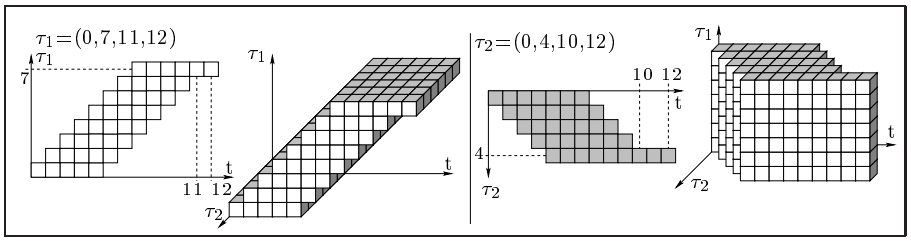


FIG. 3 – Modèles géométrique de deux tâches.

L'état du système Γ à un instant t est défini par l'état d'avancement de chacune des tâches τ_i à cet instant t. De plus, la validité d'un état à l'instant t du système dépend de la validité de l'état de chaque tâche de ce système à l'instant t.

Définition 3 *L'ensemble des états valides d'un système de tâches*

$\Gamma=(\tau_i)_{i\in[1,n]}$ *est l'ensemble* Ω *défini par :*

$$\Omega(\Gamma) = \{P = (t, x_1, ..., x_n)/\ (t, x_i) \in \Omega(\tau_i)\}$$

L'espace des points associés aux états du système est $n+1$-dimensionel. On appelle T($\Omega(\Gamma)$) l'ensemble des trajectoires valides pour Γ.

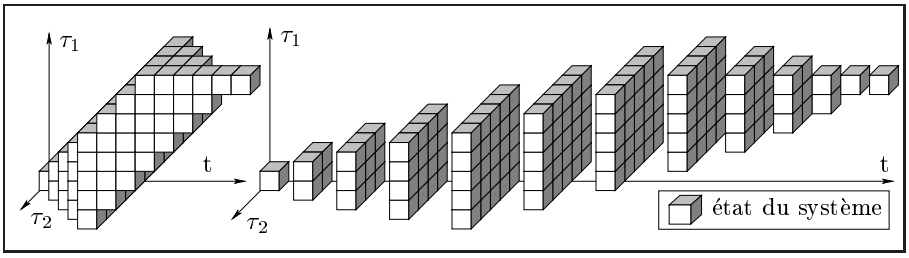


FIG. 4 – Modèle d'une application.

3.3 Partage de ressources

Nous nous intéressons maintenant à l'intégration du partage de ressource. Soit u_i l'ensemble des intervalles pour lesquels la tâche τ_i est en section critique (cf. figure 5).

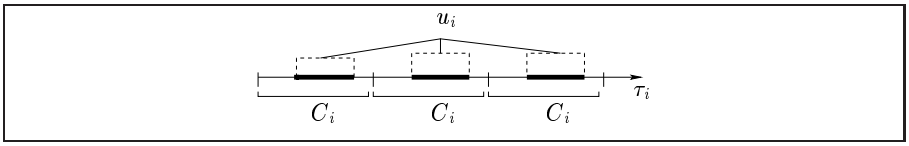


FIG. 5 – Ensemble des sections critiques d'une tâche.

Un état valide de l'application est tel qu'au plus une tâche est en section critique. Pour construire cet ensemble, nous utilisons la notion de domaine géométrique de la ressource R (cf. figure 6) :

Définition 4 *Le domaine de validité d'une application dont les tâches partagent une ressource R est*

$$\Omega_R(\Gamma) = \{(t, x_1, ...x_n),\ |\{x_i \in u_i\}| \leq 1\}$$

On appelle trajectoire R-valide toute trajectoire qui correspond à un ordonnancement compatible avec le respect de l'exclusion mutuelle pour l'accès à R. L'ensemble des trajectoires R-valides de notre application est :

$$T(\Omega_R(\Gamma)) = \{\psi \in T(\Omega(\Gamma))/\ \forall t \in \mathbb{N}^+,\ (t, \psi(t)) \in \Omega_R(\Gamma)\}$$

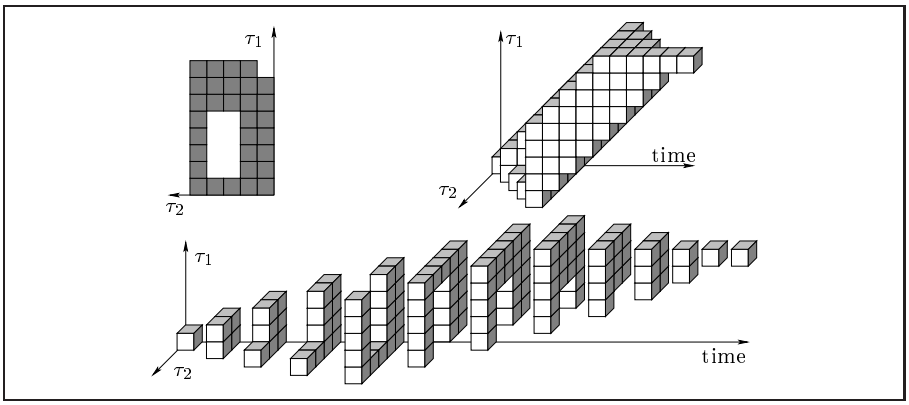


FIG. 6 – Modèle géométrique d'une application utilisant une ressource.

3.4 Partage de processeurs

Nous souhaitons maintenant tenir compte du partage de processeurs. Nous avons besoin de connaître, pour chaque trajectoire de $\mathrm{T}_R(\Gamma)$, le nombre de processeurs qui lui est nécessaire. Pour cela, nous définissons une notion de voisinage : si il faut exactement k processeurs pour passer d'un état de l'application, à l'instant t, à un état de l'application, à l'instant $t+1$, nous disons que les deux états (ou les deux points) sont k-voisins (cf. figure 7) :

Définition 5 *Deux états* $P = (t_p, p_1, ...p_n)$ *et* $Q = (t_q, q_1, ...q_n)$ *sont k-voisins si :* $t_q = t_p + 1$ *et* $\sum_{i \in [1,n]} |p_i - q_i| = k$

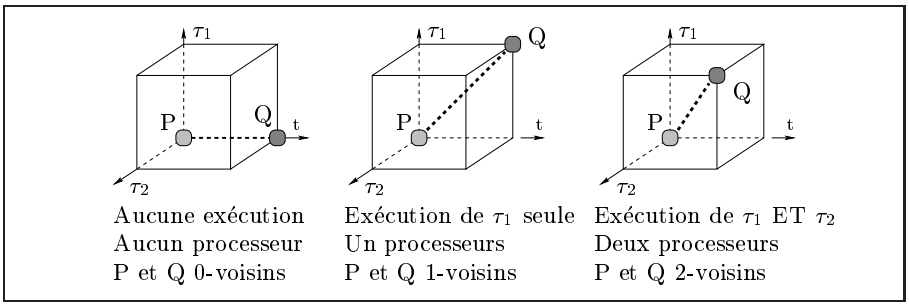


FIG. 7 – 0,1 et 2-voisinages.

Une trajectoire valide pour k processeurs sera dite k-continue. Une telle trajectoire est une suite de points dans laquelle deux points consécutifs sont au

plus k-voisins :

Définition 6 *Une trajectoire $\psi \in T_R(\Gamma)$ est k-continue si et seulement si $\forall i \in \mathbb{N}$:*

$$\left.\begin{array}{l}\psi(i) = (i, (x_i)_{i\in[1,n]}) \\ \psi(i+1) = ((i+1), (y_i)_{i\in[1,n]})\end{array}\right\} \Rightarrow \sum_{i\in[1,n]} |x_i - y_i| \leq k$$

Nous notons $\mathrm{T}_k(\Omega_R(\Gamma))$ l'ensemble des trajectoires k-continues de $\mathrm{T}_R(\Gamma)$.

Définition 7 *Un ensemble Ω_R est dit k-continu si il existe au moins une trajectoire $\psi \in T(\Omega_R(\Gamma))$ qui est k-continue.*

Conséquence : Si un ensemble Ω_R est k-continu alors il existe un ordonnancement temporellement valide de l'application sur k processeurs.

3.5 Décision d'ordonnançabilité

Un ordonnancement valide d'une application sur k processeurs est une trajectoire qui doit être au maximum k-continue. La décision d'ordonnançabilité repose donc sur l'existence d'une telle trajectoire. La décision d'ordonnançabilité pour une application $(\tau_i)_{i\in[1,n]}$ utilisant une ressource R et s'exécutant sur k processeurs est obtenue en évaluant le prédicat suivant : $T_k(\Omega_R(\Gamma)) \neq \emptyset$

4 Mesure d'invalidité

Même lorsqu'une application est invalide, l'objet géométrique qui lui est associé n'est pas vide. L'ensemble $\Omega_R(\Gamma)$ contient des états valides qui font ou non partie d'une trajectoire valide de Γ. Dans le cas où il n'existe pas de trajectoire valide, les états de $\Omega_R(\Gamma)$ font partie de trajectoires discontinues. Pour mesurer l'invalidité de l'application, nous mesurons les tailles des intervalles de discontinuité des trajectoires. Nous nous plaçons donc dans l'espace $\mathbb{N}^{n+1}$ correspondant à l'ensemble des états de commutation.

4.1 Composantes p-continues

Considérons une configuration Γ de n tâches, son domaine de validité $\Omega_R(\Gamma)$ et l'ensemble de ses trajectoires (valides et invalides au sens de la k-continuité) $T(\Omega_R(\Gamma))$. Nous utilisons, dans la suite, les notions suivantes, qui sont apparentées à celles de connexité et de composante connexe :

Définition 8 *Un sous-ensemble A d'états de $\Omega_R(\Gamma)$ est p-continu si : $\forall (P_1, P_2) \in A^2,\ \forall (t_1, t_2) \in \mathbb{N}$*

$$\left.\begin{array}{l}P_1 = (t_1, (x_i)_{i\in[1,n]}) \\ P_2 = (t_2, (x_i)_{i\in[1,n]}) \\ t_1 < t_2,\end{array}\right\} \begin{array}{l}\exists \psi \in T(\Omega_R(\Gamma)) \text{ } p\text{-continue sur } [t_1, t_2] \mid \\ \psi(t_1) = P_1, \psi(t_2) = P_2.\end{array}$$

Définition 9 *Une composante p-continue A de $\Omega_R(\Gamma)$ est un sous-ensemble p-continu maximal de $\Omega_R(\Gamma)$ (cf. figure 8) :*
$\forall a \in A, \exists b \in A | a$ *et* b *sont p-continus* $\wedge\ \forall c \notin A, \nexists d \in A | c$ *et* d *sont p-continus*

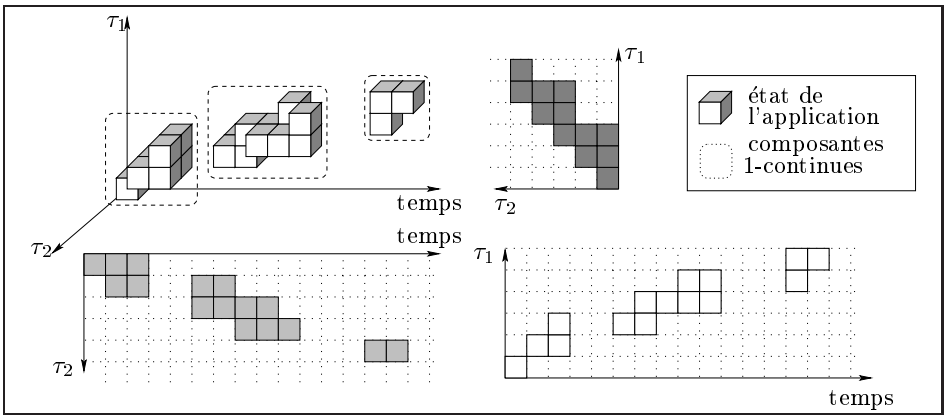


FIG. 8 – Composantes 1-continues.

Nous notons $\mathcal{X}^p$ l'ensemble des composantes p-continues de $\Omega_R(\Gamma)$. Nous avons la propriété suivante : $k < l \wedge \mathcal{X}^l = \{X_1, ..., X_r\} \wedge \mathcal{X}^k = \{Y_1, ..., Y_s\} \Rightarrow \forall j \in [1, s], \exists i \in [1, r] / Y_j \subset X_i$

Ceci découle directement de la définition de la p-continuité qui implique qu'une trajectoire k-continue sur un intervalle I est l-continue sur I pour tout $l < k$.

4.2 Distance

Nous associons la taille d'un intervalle de discontinuité d'une trajectoire à un *indicateur géométrique* entre deux composantes p-continues de $\Omega_R(\Gamma)$.

Soient P et Q deux états quelconques de l'espace $(t, (\tau_i)_{i\in[1,n]})$. Nous définissons δ_p comme le nombre d'unités de temps d'exécution minimum nécessaire, avec p processeurs, pour passer de l'état de l'application correspondant au point P à celui associé au point Q (cf. figure 9).

La fonction δ_p est définie de la façon suivante :

Définition 10 *Soient $P = (t_p, (p_i)_{i\in[1,n]})$ et $Q = (t_q, (y_i)_{i\in[1,n]})$ tels que $t_p < t_q$. La distance entre P et Q est :*

$$\delta_p \ : \ \mathbb{N}^{n+1} \times \mathbb{N}^{n+1} \rightarrow \mathbb{N}$$

$$\delta_p\,(P,Q) = max\Big(\Big\lceil \frac{\sum_{i=1}^{n} |p_i - q_i|}{p} \Big\rceil,\ max_{i\in[1,n]}|p_i - q_i|, |t_p - t_q|\Big)$$

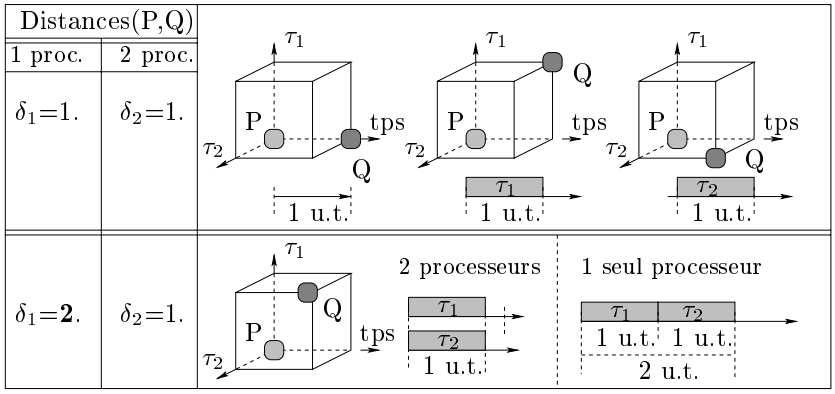


FIG. 9 – Distances entre $P = (t, p_1, ..., p_n)$ et $Q = (t+1, q_1, ..., q_n)$.

Le terme $\left\lceil \frac{\sum_{i=1}^{n} |p_i - q_i|}{p} \right\rceil$ correspond à une répartition au mieux de la charge sur les processeurs disponibles. Toutefois, une tâche τ_i doit être exécutée pendant $|p_i - q_i|$ unités de temps pour passer de l'état P à l'état Q. Les tâches n'étant pas réentrantes, la durée $\delta_p(P, Q)$ ne peut être inférieure à la plus grande durée d'exécution nécessaire à une tâche pour passer d'un état à l'autre. Nous avons ainsi le terme $max_{i\in[1,n]}|p_i - q_i|$. Le dernier terme, $|t_p - t_q|$, signifie que pour passer de l'état P à l'état Q on ne peut pas mettre moins de temps que le temps qui sépare les deux états.

Exemple : On considère un système $\Gamma=(\tau_i)_{i\in[1,6]}$ de six tâches. Soit $P = (t_p, (p_i)_{i\in[1,6]})$, et soit $Q = (t_q, (q_i)_{i\in[1,6]})$ deux états de l'application avec $t_p < t_q$.

P	t_p	p_1	p_2	p_3	p_4	p_5	p_6
Q	$t_p + 4$	$(p_1 + 2)$	$(p_2 + 3)$	$(p_3 + 1)$	$(p_4 + 4)$	p_5	$(p_6 + 3)$

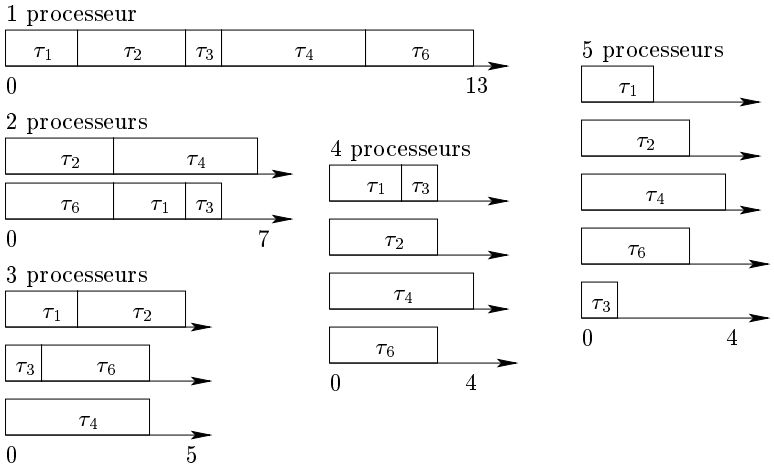
Les valeurs δ_p pour chaque nombre possible de processeurs sont :

$\delta_1(P,Q)=max\Big(13,4,4\Big)=13$ $\quad$ $\delta_2(P,Q)=max\Big(\lceil 6.5\rceil,4,4\Big)=7$

$\delta_3(P,Q)=max\Big(\lceil 4.33\rceil,4,4\Big)=5$ $\quad$ $\delta_4(P,Q)=max\Big(\lceil 3.25\rceil,4,4\Big)=4$

$\delta_5(P,Q)=max\Big(\lceil 2.6\rceil,4,4\Big)=4$ $\quad$ $\delta_6(P,Q)=max\Big(\lceil 2.16\rceil,4,4\Big)=4$

Ces valeurs peuvent être associées à des séquences d'exécution. Nous présentons un exemple de séquence d'exécution pour chaque δ_p :



Théorème 1 *δ_p est une distance, c-à-d. δ_p vérifie les axiomes suivants [L04] :*
- Séparation : $\delta_p(P,P) = 0$
- Symétrie : $\delta_p(P,Q) = \delta_p(Q,P)$
- Triangulaire : $\delta_p(P,R) \leq \delta_p(P,Q) + \delta_p(Q,R)$

Nous définissons la distance δ_p entre deux composantes p-continues de la façon suivante :

Définition 11 *La distance δ_p entre deux composantes p-continues X_i et X_j de $\Omega_R(\Gamma)$ est $\delta_p(X_i, X_j) = min_{P \in X_1, Q \in X_2}\big(\delta_p(P,Q)\big)$.*

Remarque : La distance δ_p entre deux composantes p-continues n'est jamais nulle car les composantes p-continues sont disjointes : une composante p-continue est un ensemble p-continu maximal. Lorsqu'une application est ordonnançable son domaine de validité est constitué d'une unique composante p-continue.

Ces différentes définitions permettent, dans la partie suivante, la définition d'une mesure d'invalidité.

4.3 Mesure d'invalidité

Nous définissons notre mesure d'invalidité M_p d'une configuration de tâches Γ grâce à la distance entre composantes p-continues. Le modèle géométrique d'une application invalide comporte plusieurs composantes p-continues. Nous disposons, pour chaque couple de composantes, de la distance δ_p. La mesure d'invalidité que nous définissons tient compte de cet ensemble de distances.

La mesure devant quantifier l'invalidité, nous avons choisi de considérer le plus grand intervalle de discontinuité. Cet intervalle étant le plus grand, nous sommes partis de l'hypothèse que lorsque, en modifiant la spécification, on aura réussi à combler ce *vide*, on aura par la même occasion comblé les *vides* plus petits et ainsi rendu l'application valide. La validation de cette hypothèse s'appuie, bien sûr, sur des expérimentations actuellement en cours.

Définition 12 *Soit $\mathcal{X}^p = \{X_1, ..., X_s\}$ l'ensemble des composantes p-continues de $\Omega_R(\Gamma)$. Alors :*

$$M_p(\Gamma) = \begin{cases} 0 & si\ |\mathcal{X}| = 1, \\ Max_{i,j \in [1,s]^2}(\delta_p(X_i, X_j)) & sinon. \end{cases}$$

5 Propriétés

Dans la partie précédente, nous avons défini une mesure de non ordonnançabilité M_p. Vérifions que cette mesure vérifie les propriétés spécifiées dans la section 2.2. Pour les démonstrations, nous notons $\mathcal{C}$ un contexte d'ordonnancement à p processeurs, et Γ une configuration de n tâches.

5.1 Propriétés de la mesure

Notre mesure devant quantifier l'invalidité, elle doit vérifier les propriétés suivantes :

- être nulle pour toutes les configurations ordonnançables ($M_p(\Gamma) = 0 \Leftrightarrow \mathcal{O}_\mathcal{C}(\Gamma) \neq \emptyset$) et non nulle pour les applications invalides ($M_p(\Gamma) > 0 \Leftrightarrow \mathcal{O}_\mathcal{C}(\Gamma) = \emptyset$).
- $M_p(\Gamma) = 0 \Rightarrow \forall k > p,\ M_k(\Gamma) = 0$, car une application ordonnançable dans un contexte comportant p processeurs l'est aussi lorsque le nombre de processeurs est plus grand.
- $M_p(\Gamma) > 0 \Rightarrow \forall k < p,\ M_k(\Gamma) \geq M_p(\Gamma)$, car une application non ordonnançable sur p processeurs ne l'est pas pour un nombre de processeurs plus petit.

Montrons donc que $M_p(\Gamma) = 0 \Leftrightarrow \mathcal{O}_\mathcal{C}(\Gamma) \neq \emptyset$.

Preuve : Si $M_p(\Gamma) = 0$ alors $|\mathcal{X}^p|$=1 et donc $\mathcal{X}^p = \{\Omega_R(\Gamma)\}$. Comme les éléments de $\mathcal{X}^p$ sont p-continus, $\Omega_R(\Gamma)$ est p-continu. Il existe donc une trajectoire p-continue correspondant à un ordonnancement valide de Γ sur p processeurs et donc $\mathcal{O}_\mathcal{C}(\Gamma) \neq \emptyset$.

Si $\mathcal{O}_\mathcal{C}(\Gamma) \neq \emptyset$ alors il existe une trajectoire ψ p-continue valide pour Γ. Comme ψ est p-continue, tous les points qu'elle définit font partie de la même composante p-continue. Or ψ est définie pour tout t de $\mathbb{N}$ donc il n'y a qu'une

seule composante p-continue. Dans ce cas, $|\mathcal{X}^p|$=1 et $M_p(\Gamma)$=0. □

De la même manière, on montre que $M_p(\Gamma) > 0 \Leftrightarrow \mathcal{O}_{\mathcal{C}}(\Gamma) = \emptyset$.
Preuve : Si $M_p(\Gamma) > 0$ alors $|\mathcal{X}^p| > 1$. Comme $\mathcal{X} = \{X_1, ..., X_s\}$ avec $\forall i \in [1, |\mathcal{X}^p|], \forall j \neq i, X_i \cap X_j = \emptyset$, il n'existe aucune trajectoire p-continue sur [0,+∞[et donc aucun ordonnancement valide ($\mathcal{O}_{\mathcal{C}}(\Gamma) = \emptyset$).

Si $\mathcal{O}_{\mathcal{C}}(\Gamma) = \emptyset$, alors il n'existe aucune trajectoire p-continue sur [0,+∞[. Soit ψ une trajectoire de Γ, ψ n'étant pas p-continue, il existe $I = [t_p, t_q]$ / ψ n'est pas p-continue sur I. On a donc P= $(t_p - 1, \psi(t_p - 1))$ et Q= $(t_q + 1, \psi(t_q + 1))$ qui ne sont pas p-continus. Suivant la définition des composantes p-continues, on a nécessairement au moins deux composantes p-continues X_1 et X_2 avec P ∈ X_1 et Q ∈ X_2. Et enfin $|\mathcal{X}^p| \geq 2$ et donc $M_p(\Gamma) > 0$. □

La mesure que nous avons définie correspond donc bien à la spécification : la mesure d'invalidité associée à une application ordonnançable est nulle.

Nous montrons donc que $M_p(\Gamma) = 0 \Rightarrow \forall k > p,\ M_k(\Gamma) = 0$.
Preuve : Supposons $M_p(\Gamma) = 0$; alors $|\mathcal{X}^p|$=1. Il existe donc une trajectoire $\psi \in T_p(\Omega_R(\Gamma))$ p-continue sur l'intervalle [0,+∞[. Comme toute trajectoire p-continue est k-continue pour tout $k > p$, ψ est aussi k-continue sur [0,+∞[. Il existe donc une seule composante k-continue : $|\mathcal{X}^k|$=1. Nous avons donc $\forall k > p,\ M_k(\Gamma) = 0$. □

Une application qui n'est pas ordonnançable sur p processeurs l'est encore moins lorsque l'on diminue le nombre de processeurs.

Montrons que $M_p(\Gamma) > 0 \Rightarrow \forall k < p,\ M_k(\Gamma) \geq M_p(\Gamma)$.
Preuve : Supposons $M_p(\Gamma) > 0$ alors $\mathcal{X}^p = \{X_1^p, ..., X_r^p\},\ r > 1$. Soit $k < p$, d'après la propriété des composantes continues, on a : $\mathcal{X}^k = \{X_1^k, ..., X_l^k\},\ l > 1$ et $\forall i \in [1, l], \exists j \in [1, r],\ X_i^k \subseteq X_j^p$. La distance maximale entre deux composantes k-continues est donc au minimum égale à la distance maximale entre deux composantes p-continues (cf. figure 10). Nous avons donc $M_k(\Gamma) \geq M_p(\Gamma)$.
□

5.2 Autres propriétés

On note Θ une trajectoire p-continue. Par définition des distances et de la mesure, on a : $M_p(\Gamma) = 0 \Rightarrow \forall t \in \mathbb{N},\ P = (t, \Theta(t)),\ Q = (t + 1, \Theta(t + 1)),\ \delta_p(P, Q) = 1$. Soit p=inf $\{k,\ M_k(\Gamma) = 0\}$, le plus petit nombre de processeurs pour lequel l'application est ordonnançable. Alors, on a la propriété

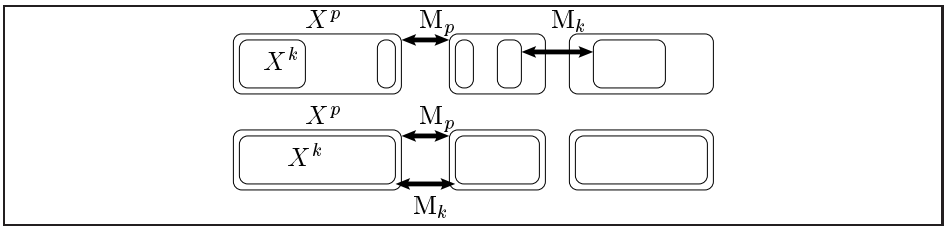


FIG. 10 – Mesures avec k et p processeurs.

suivante :

$$\forall k < p,\ M_k(\Gamma) = \left\lceil \frac{p}{k} \right\rceil .$$

Preuve : Soit p=inf $\{k,\ M_k(\Gamma) = 0\}$, et soit $k < p$, alors on considère tous les couples de points de la trajectoire Θ consécutifs P et Q tels que : $P = (t, \Theta(t))$ et $Q = (t+1, \Theta(t+1))$ soient k-discontinus.
Comme P et Q sont p-continus, on a :

$$\left.\begin{array}{l} P = (t, (p_i)_{i\in[1,n]}) \\ Q = (t+1, (q_i)_{i\in[1,n]}) \end{array}\right\} \Rightarrow \delta_p(P,Q) = 1.$$

Donc :

$$\left\lceil \frac{\sum_{i=1}^{n} |p_i - q_i|}{p} \right\rceil = 1.$$

Comme p=inf $\{k,\ M_k(\Gamma) = 0\}$, on a $\sum_{i=1}^{n} |p_i - q_i| = p$: en effet, si $\sum_{i=1}^{n} |p_i - q_i| = p - 1$ alors $M_{p-1}(\Gamma)$=0 et donc $p \neq inf\{k, M_k(\Gamma) = 0\}$.
On a alors, $\delta_k(P,Q) = max(\lceil \frac{p}{k} \rceil, 1, 1) = \lceil \frac{p}{k} \rceil$.
Et donc :
$\forall P = (t, \Theta(t)), Q = (t+1, \Theta(t+1)),\ \delta_k(P_1, P_2) = \lceil \frac{p}{k} \rceil (> 1)\ ou\ \delta_k(P,Q) = 1$, suivant que les points sont k-continus ou non.
On a alors $M_k(\Gamma) = Max_{t\in\mathbb{N}}(\delta_k((t, \Theta(t)), (t+1, \Theta(t+1))))$.
Et donc $M_k(\Gamma) = \lceil \frac{p}{k} \rceil$. □

Cette propriété permet de simplifier l'algorithme de calcul de la mesure.

6 Résultats et expérimentations

Le principal avantage du modèle géométrique est que l'objet construit est le même quelque soit le nombre de processeurs. La différence entre les contextes n'intervient que lorsque l'on considère les voisinages. On peut donc calculer les mesures pour tous les nombres de processeurs que l'on souhaite à partir du même objet. Ceci permet d'obtenir certains résultats intéressants : lorsque la

mesure est non nulle pour un nombre de processeurs égal au nombre de tâches, nous sommes sûrs qu'il y a un interblocage. En effet, chaque tâche ayant son propre processeur, il n'y a pas de conflit sur ce point et si l'application est invalide c'est donc un partage de ressource qui est en cause. De plus, lorsque le concepteur spécifie un nombre p de processeurs pour lequel la mesure est non nulle, nous pouvons calculer toutes les autres mesures et, s'il existe, on peut trouver un nombre de processeurs $k(> p)$ pour lequel la mesure est nulle. Le concepteur pourra soit modifier les paramètres de ses tâches, soit modifier son architecture en y ajoutant des processeurs.

L'expérimentation du calcul de cette mesure a un double objectif. D'une part montrer son homogénéité (lorsque la charge varie, la mesure doit varier de façon cohérente) et d'autre part, essayer de déterminer les règles de modification des paramètres qui peuvent permettre de rendre l'application valide. Ce deuxième objectif est en cours d'étude. Les premières expérimentations montrent d'une part que la mesure est bien homogène et d'autre part, qu'elle est sensible : lorsque la charge diminue faiblement (augmentation des période), la mesure diminue faiblement et lorsque la charge diminue brutalement (augmentation des délais critiques), la mesure diminue fortement.

7 Conclusion

Le modèle géométrique que nous avons défini permet de modéliser une application temps réel multiprocesseurs avec partage de ressource. La notion de k-continuité que nous avons introduite permet de décider de l'ordonnançabilité des applications sur des architectures données.

Nous avons défini une mesure d'invalidité sur la base d'une distance entre composantes p-continues. Cette mesure dépend du nombre de processeurs et correspond à la taille du plus grand intervalle de p-discontinuité.

Les premières expérimentations montrent que cette mesure a un comportement cohérent suivant les modification de la charge : dans la série de test effectués, nous avons constaté que, conformément à nos prévisions, les meilleures valeurs sont obtenues lorsque les tâches sont à échance sur requête (le délai critique est le plus grand possible).

Plusieurs perspectives sont envisageables. Dans un premier temps, il serait intéressant d'étudier plus systématiquement le comportement de notre mesure en fonction des modifications des paramètres temporels des tâches. La modification du taux d'utilisation intervient peu dans l'évolution de la mesure, alors que la diminution du taux de charge propose de bien meilleurs résultats. Une expériementation plus complète permettraient sans doute de dégager des relations plus précises entre paramètres et mesures. Dans un second temps, la mesure pourrait être raffinée par la prise en compte, par exemple, du nombre d'intervalles de discontinuité.

Références

[But97] G. Buttazzo : *Hard real-time computing systems : Predictable scheduling algorithms and applications.* Kluwer Academic Publishers, 1997.

[CHO96] A.Choquet-geniet, D.Geniet, F.Cottet : *Exhaustive computation of scheduled task execution sequences of real-time application.* Proc FTRTFT'96.(1996)

[CP00] A. Colin, I. Puaut : *Worst case execution time analysis for a processor with branch prediction.* Real-Time Systems, Vol 18(2), pp. 249-274. 2000.

[L04] G. Largeteau : *Quantification du taux d'invalidité d'applications temps réel à contraintes strictes.* thèse, Octobre 2004.

[LCG04] G. Largeteau, B. Chauvière, D. Geniet : *Une approche géométrique pour la validation d'application temps réel à contraintes ctricte.* Real Time Systems. BIRP. (2004).

[SSR98] A.Stankovic, M.Spuri, K.Ramamrithan, G.C.Buttazzo : *Deadline Scheduling for real-time systems.* Kluwer Academic Publishers, ISBN 0-7923-8269-2. (1998).

[Sta88] J. Stankovic : *Misconception about real-time computing.* IEEE computer magazine 10, n 21, pp 1-19. (1988).