



Ordonnancement sous contrainte d'énergie

Nicolas Navet

► To cite this version:

Nicolas Navet. Ordonnancement sous contrainte d'énergie. Action Spécifique N°106 Conception faible consommation, 2004, Paris/France, 38 p. inria-00099915

HAL Id: inria-00099915

<https://inria.hal.science/inria-00099915>

Submitted on 26 Sep 2006

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.

Ordonnancement sous contrainte d'énergie

Nicolas NAVET

INRIA projet TRIO

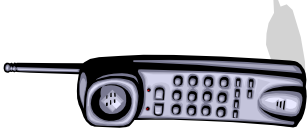
<http://www.loria.fr/~nnavet>

Certains exemples et représentations proviennent de :

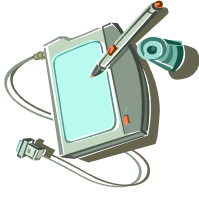
- « Voltage Scheduling Problem for Dynamically Variable Voltage Processors », T. Ishihara, Y. Yasuura
- « Energy-Centric Scheduling for Real-Time Systems », F. Gruian

Le contexte (1/2)

- Pour



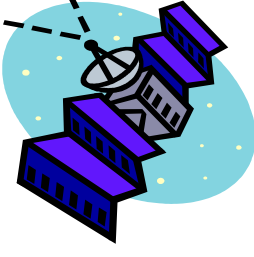
GSM



**Ordinateurs
portables**



**Implants
médicaux**



satellites

- Evolution actuelle:

↑ fréquences $\Rightarrow$ ↑ consommation
(ex: Pentium 4M 1.8Ghz=30W, 386=3W)

or la technologie des batteries progresse peu ...

- On estime que le CPU consomme **entre 30% et 50%** de l'énergie d'un ordinateur portable

Le contexte (2/2)

- Les contraintes de performances se traduisent en contraintes sur les dates d'échéances
- Comment diminuer la consommation CPU ?
par réduction de la tension d'alimentation et donc la fréquence
- Les gains en autonomie sont importants

⇒ Ordonnancer, c'est ici choisir l'ordre ainsi que la fréquence de fonctionnement

Réduction de la consommation

- Technologies matérielles
 - taille transistors
 - codage Grey
 - alimentation des composants individualisée
 - circuits asynchrones
 - ...
- Optimisation logicielles
 - remplacer op. mémoire-à-mémoire par registre-à-registre
 - inlining - loop unrolling
 - ...
- Techniques hybrides
 - mise en veille de composants
 - **modification dynamique de la fréquence CPU**

Technologie des processeurs

- Vitesse fixe et mode(s) veille
 - au moins 2 modes : «normal» et «veille»
 - le pb est de sélectionner le mode de marche courant du système (*Dynamic Power Management* - DPM)
 - standard ACPI
- Vitesses (dynamiquement) variables
 - nombre de fréquences nécessairement fini
 - le pb est de d'adapter la fréquence de fonctionnement en ligne (*Dynamic Voltage Scaling* – DVS)
 - ex: *Crusoe*, *Pentium4M*, *LpARM* etc...

Modèle de consommation (1/4)

- Puissance dissipée en CMOS $\approx P_{dynamic} \sim \alpha \cdot f \cdot C \cdot V^2$
- Relation entre f et V : $f \sim V^{\gamma-1}$ Vitesse CPU: $S_f = \frac{f}{f_{ref}}$
- Puissance dissipée à la fréquence f : $P_f = P_{ref} \cdot S_f^{1+2/(\gamma-1)}$
- Énergie par cycle à la fréquence f :

$$e_f = \int_a^{a+1/f} P_f dt = \frac{1}{f} e_{ref} \cdot f_{ref} \cdot S_f^{1+2/(\gamma-1)} = e_{ref} \cdot S_f^{2/(\gamma-1)}$$

⇒ Pour $\gamma = 2$ (modèle MOSFET), l'énergie est quadratique en la vitesse – autres modèles: toujours convexe

Modèle de consommation (2/4)

- **Remarque** : fonctionnement lent > mode veille (DPM)

Une tâche de x cycles devant s'exécuter dans un temps T :

- x cycles à la fréquence f_{ref} puis veille: $E_{ref} = x \cdot e_{ref}$
- x cycles à la fréquence $f = x / T$:

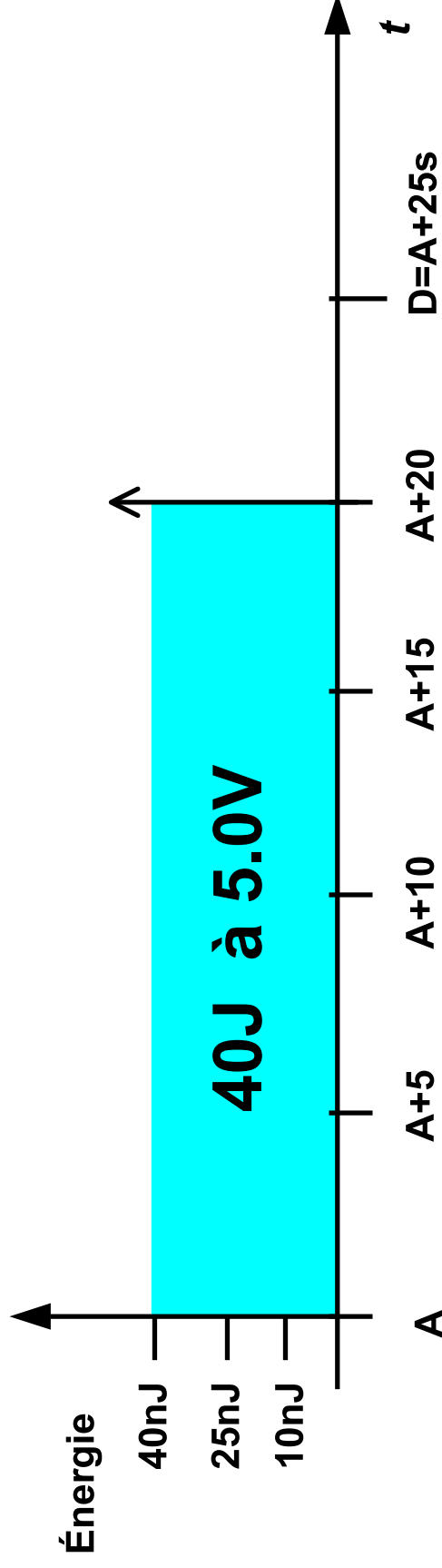
$$E_f = x \cdot e_{ref} \cdot S_f^{\frac{2}{\gamma-1}} \leq E_{ref}$$

- **Plus généralement**, du point de vue de la consommation, une vitesse unique jusqu'à l'échéance est la meilleure solution

Modèle de consommation (3/4)

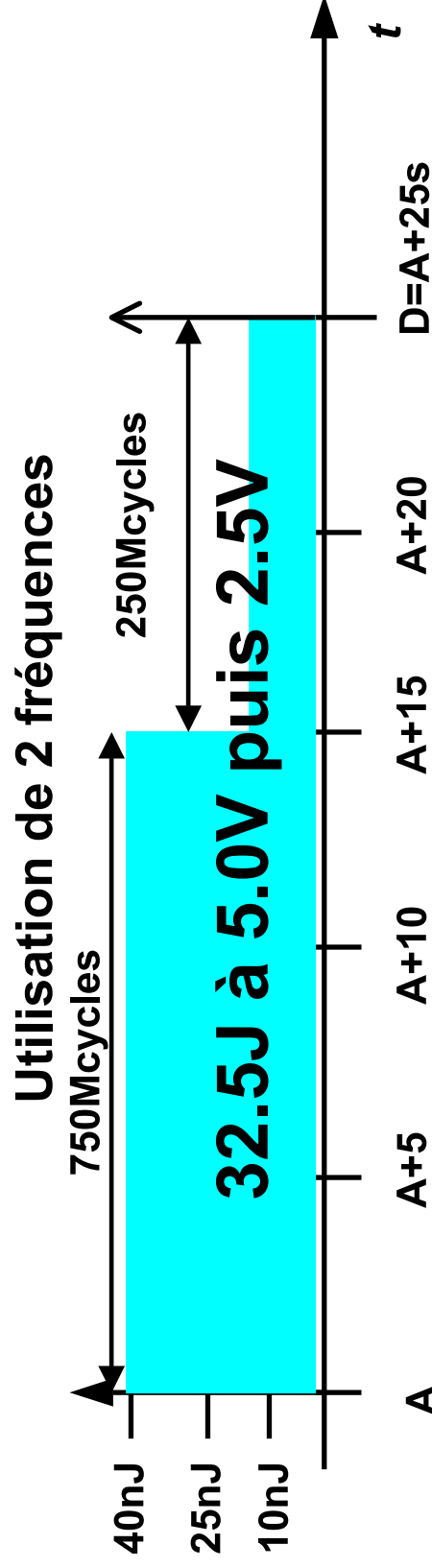
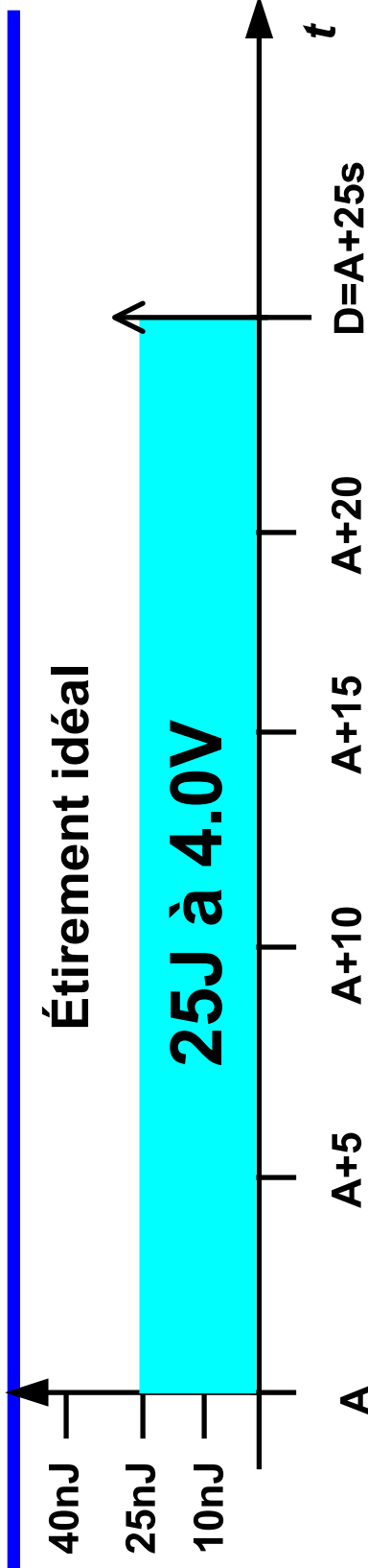
- Exemple:
code: 10^9 cycles
 $D-A = 25s$

Tension	Fréquence	Energie/Cycle
2.5 V	25Mhz	10nJ
4.0 V	40Mhz	25nJ
5.0 V	50Mhz	40nJ



Fct à vitesse maximale

Modèle de consommation (4/4)



⇒ si la vitesse idéale manque, l'utilisation des 2 vitesses adjacentes est la meilleure solution

Notions d'ordonnement temps réel

- Modèles d'activité :
 - job : activité non récurrente ou
 - tâche : activité récurrente (généralement périodique) – les activations successives sont les instances de la tâche
- Caractéristiques des activités :
 - A_i : date de mise à disposition d'un job (ou d'une instance)
 - C_i : pire temps d'exécution d'un job (ou Worst-Case Execution Time - **WCET**) en général bien plus grand que le temps effectif (Actual E.T. ou **AET**)
 - T_i : période d'une tâche
 - D_i : échéance d'un job

➤ Le problème principal du temps réel est la faisabilité du système (*i.e.* respect de toutes les échéances) ...

Les $\neq$ stratégies d'ordonnancement

A. Politiques hors-ligne (ou statique): choix des fréquences effectué à la conception

B. Politiques en-ligne (ou dynamique): choix des fréquences effectué en-ligne

- Dans les 2 cas, l'ordre sur les activités est déterminé par un algorithme dynamique (le plus souvent, ex: EDF ou FPP) ou s'exécute selon un cycle pré-calculé
- Les politiques en-ligne utilisent la connaissance des temps d'exécution réels (très $\neq$ du pire temps d'exécution)

Politiques hors-ligne

Classification selon le nombre de vitesses allouées:

- I. Vitesse unique
- II. Vitesse unique chaque tâche (*i.e.* pour toutes les instances d'une tâche récurrente)
- III. Vitesse unique pour chaque instance d'une tâche
- IV. Vitesse choisie pour un cycle d'horloge

Politiques en-ligne

- I. Redistribution des temps d'exécution non-utilisés ('gain reclaiming')
 - II. Ordonnancement avec instrumentation du code ('compiler assisted scheduling' ou 'intra-task voltage scheduling')
- Les vitesses initiales sont calculées avec une stratégie hors-ligne

Politiques hors-ligne à vitesse unique

- Principes et limites
- Ordonnancement EDF ('Earliest Deadline First')
- Ordonnancement FPP ('Fixed-Priority Policy')
- Ordonnancement Round-Robin

Hors-ligne à vitesse unique (1/3)

- **Objectif:** déterminer la fréquence minimale admissible valable pendant toute la durée de vie du système
- **Points + :**
 - OS / CPU «standards» - pas d'overhead en-ligne
 - des résultats existent pour EDF, FPP, RR, ... (possible si $\exists$ analyse d'ordonnabilité pour la politique)
- **Points - :** performances ! L'activité la plus contrainte impose sa vitesse ..
- **Nb:** une telle politique peut néanmoins être optimale dans des cas particuliers (ex: EDF, $D_i=T_i$, tâches synchrones)

Hors-ligne à vitesse unique (2/4)

- EDF [Gru02]: utilisation du test de Lui et Layland pour des tâches périodiques avec $D_k = T_k$
- Principe: les tâches sont faisables sous EDF ssi

$$\left(U_T = \sum_{\tau_i} C_i / T_i \right) \leq 1$$

La vitesse minimale acceptable est donc

$$S_{edf} = U_T$$

- **Nb:** pour des échéances quelconques, il faut utiliser le test (seulement suffisant) :

$$\left(U_T = \sum_{\tau_i} C_i / \min(D_i, T_i) \right) \leq 1$$

Hors-ligne à vitesse unique (3/4)

- FPP [Gru02]: utilisation du test de Lui et Layland pour des tâches périodiques avec $D_k = T_k$
- Principe: les tâches sont faisables sous FPP si

$$U_T = \sum_{\tau_i} C_i / T_i \leq m \cdot (2^{1/m} - 1)$$

La vitesse minimale acceptable est donc

$$S_{fpp} = \frac{U_T}{m \cdot (2^{1/m} - 1)}$$

- Pb: Le test n'est que suffisant et non nécessaire .. Il existe un test suffisant et nécessaire (dit test de Lehoczki)

Hors-ligne à vitesse unique (4/4)

- **Round-Robin [BrNa04]:** pas de test de faisabilité, il faut faire un calcul de temps de réponse pour chaque job ou instance ...
- **Observations :**
 - la faisabilité n'est pas monotone en la vitesse
 - la vitesse nécessaire sous RR est toujours supérieure à la vitesse requise sous EDF (*Sedf*)
 - un job qui finit plus tôt que prévu peut entraîner la non-faisabilité du système (mécanismes en-ligne)
- **Algorithme:** en partant de *Sedf*, choisir la première vitesse disponible qui soit admissible

Politiques hors-ligne

vitesse choisie par instance

- Principes et limites
- Ordonnancement EDF :
 - algorithme classique de Yao et al.
 - une solution algorithmiquement moins complexe

Hors-ligne - vitesse choisie par instance (1/x)

- **Objectif:** déterminer la fréquence de chacune des instances des tâches (ou des jobs) de telle façon à minimiser la consommation
- **Points + :**
 - efficacité très supérieure à une solution à vitesse unique
 - des résultats optimaux existent pour EDF et FPP
- **Points - :**
 - changements de fréquences effectués au niveau de l'ordonnanceur : adaptation de l'OS ..
 - nécessite de calculer et de stocker les fréquences pour (au minimum) un PPCM des périodes .. d'où l'intérêt des politiques à changements de vitesse par tâche

Hors-ligne - vitesse choisie par instance (2/x)

- La politique FPP: prouvé NP-difficile et approximation $(1+\varepsilon)$ [YuKi2003]
- Le cas EDF: algo. classique de Yao [YaDeSh95]

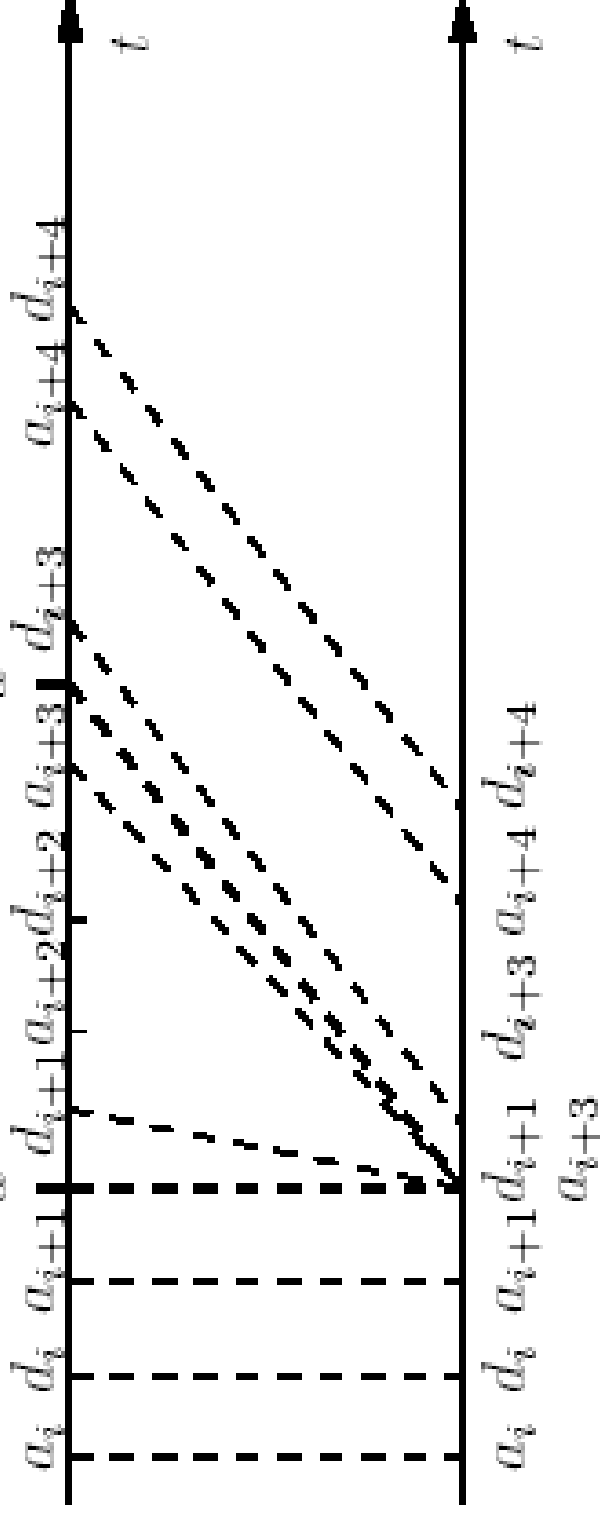
$$\text{facteur de charge : } U(t_1, t_2) = \frac{\sum_k C_k \cdot 1_{[t_1 \leq A_k < t_2]} \cdot 1_{[D_k \leq t_2]}}{t_2 - t_1}$$

- faisabilité ssi $\forall t_1, t_2 \quad U(t_1, t_2) \leq 1$
- algorithme:

- 1) $[a, b[= \operatorname{argmax}_{t_1, t_2} U(t_1, t_2)$; fixer $S_{EDF}[a, b[= U(a, b)$
- 2) retirer $[a, b[$ et revenir en 1)

Hors-ligne - vitesse choisie par instance (3/x)

- Étape 2 de l'algorithme de Yao : suppression de l'intervalle $[a, b]$ (dit intervalle critique)



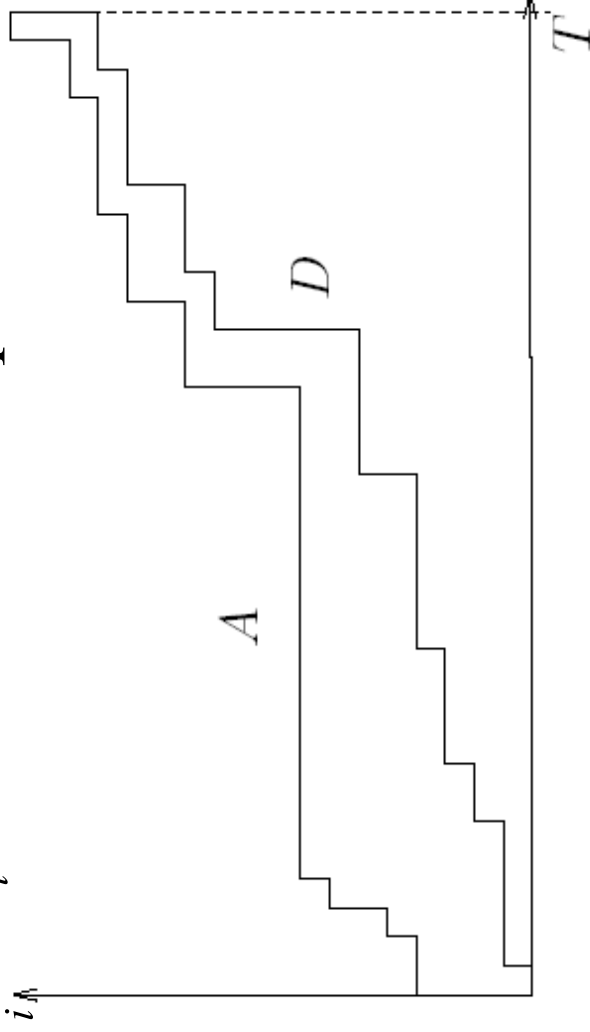
Complexité : $O(n^3)$ où n est le nombre d'instances

Nb: le premier intervalle critique détermine la faisabilité des tâches sous EDF : test de Spuri en $O(n^2)$

Hors-ligne - vitesse choisie par instance (4/x)

■ Une solution moins complexe que l'algo. de Yao

- $A(t)$: fonction d'arrivée de travail
- $D(t)$: quantité de travail à faire avant t
- $u(t)$: vitesse du processeur en l'instant t
- $U(t) = \int_0^t u(s) ds$
- $T = \max_i D_i$ est l'horizon d'optimisation



Hors-ligne - vitesse choisie par instance (5/x)

■ Problème :

trouver $u(t)$ qui minimise $\int_0^T g(u(t)) dt$ sous les contraintes:

$$\bullet \int_0^t u(s) ds \leq A(t) \quad \forall t \in [0, T]$$

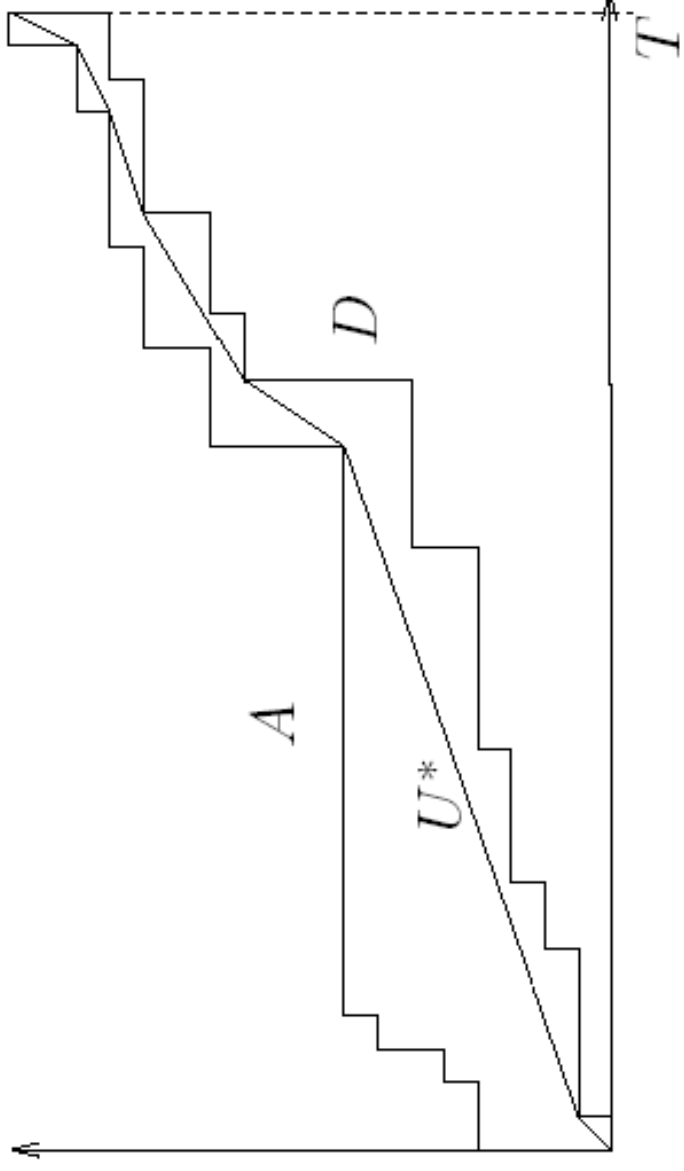
$$\bullet \int_0^t u(s) ds \geq D(t) \quad \forall t \in [0, T]$$

$$\bullet u(t) \geq 0$$

➤ Pour des tâches à contraintes FIFO ($A_i \leq A_j \rightarrow D_i \leq D_j$) sous EDF, le respect de ces contraintes garantit la faisabilité du système

Hors-ligne - vitesse choisie par instance (6/x)

- Solution pour des tâches FIFO (ie. des tâches t.q. $A_i \leq A_j \rightarrow D_i \leq D_j$) [GaNaWa03]:



U^* est le plus court chemin de 0 à T

Hors-ligne - vitesse choisie par instance (7/x)

■ Étapes:

- la solution optimale est unique et valable pour toutes les fonctions convexes croissantes

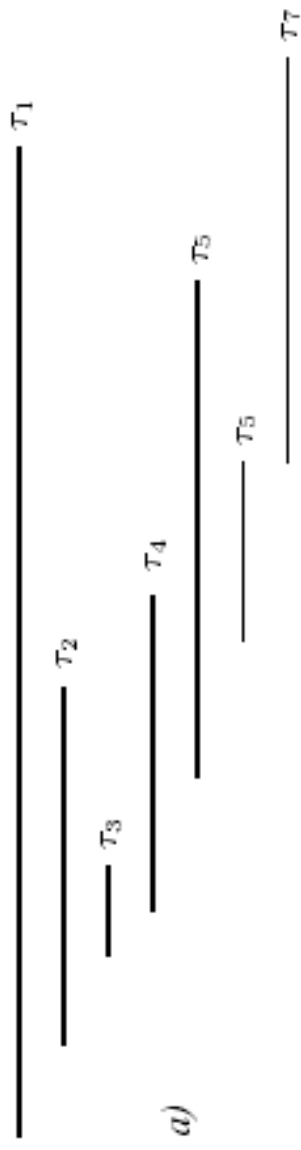
- $g(x) = \sqrt{1 + x_f^2}$ est convexe croissante - la longueur de $u(t)$ est $L = \int_0^f \sqrt{1 + u^2(s)} ds$ - la solution optimale qui minimise L est donc le plus court chemin

■ Avantages:

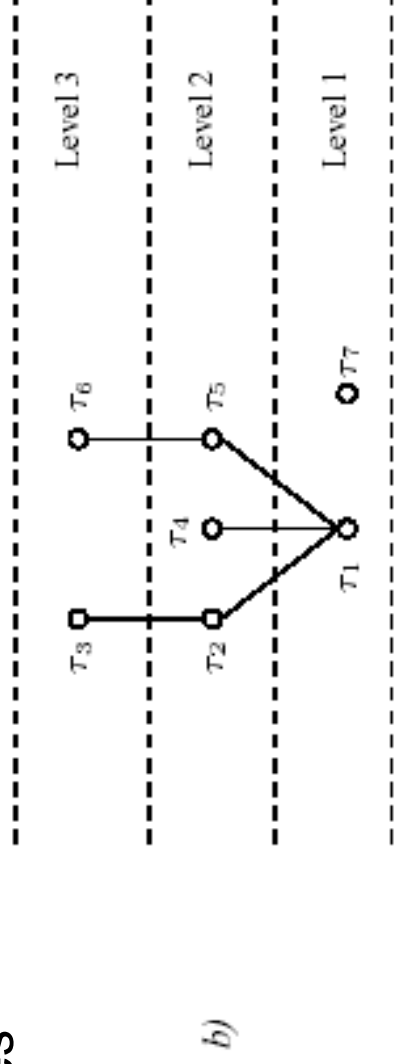
- complexité : $O(n \log(n))$
- extensions possibles : nombre de vitesses finies, nombre de changements de contexte minimum, tâches fluides ...

Hors-ligne - vitesse choisie par instance (8/x)

- Pour des tâches quelconques [GaNa04]



Relation d'ordre partiel
« inclus dans » sur les
intervalles $[A_i, D_i]$



Hors-ligne - vitesse choisie par instance (8/x)

- **Notations:**

- K est le nombre de niveaux dans le diagramme de Hasse
- $u_i^*(t)$ est la vitesse qui conduit au plus court chemin pour le pb restreint aux tâches de niveau $i \dots K$ (cf. tâches FIFO)

- **Algorithme:**

1. pour $i=1 \dots K$ construire $u_i^*(t)$; $r_i = \sup_{0 \leq t \leq T} u_i^*(t)$
l'intervalle critique $[a,b]$ est t.q. sa charge est $W_{[a,b]} = \sup_i r_i$
2. la vitesse sur $[a,b]$ est $W_{[a,b]}$; on retire $[a,b]$ et retour en 1.

- **Complexité** : $O(n^2 K)$ dans le pire cas - $O(n^2 \log(n))$ en moyenne (vs n^3 dans les 2 cas pour Yao)

Politiques hors-ligne vitesse choisie par cycle

- Ordonnancement stochastique

Ordonnancement stochastique (1/2)

- Objectif: Minimiser l'espérance de l'énergie consommée

avec:
$$\overline{E} = \sum_{x=1}^{WCE} (1 - F(x)) \cdot e_x$$

- WCE : le nbre de cycles correspondant au WCET
- e_x : énergie consommée au cycle x
- $F(x)$: probabilité de terminer avant le cycle x

- Résultat [Gru02] : temps de cycle optimal k_y

$$k_y = (D - A) \cdot \frac{\sqrt[3]{1 - F(y)}}{\sum_{x=1}^{WCE} \sqrt[3]{1 - F(x)}}$$

Ordonnancement stochastique (2/2)

- **Remarque** : la vitesse augmente graduellement – les vitesses élevées ne seront généralement pas utilisées
- **Problèmes**:
 - nécessité d'une distribution des temps d'exécution
 - WCE est grand d'où la nécessité de travailler sur des groupes d'instructions : comment découper ?
 - transcription fréquences calculées vers les fréquences disponibles ...
 - une tâche seule se verra allouer un temps (D-A), quid du cas multi-tâches ??

Politiques en-ligne

- Redistribution des gains ('Gain Reclaiming')
- Instrumentation du code ('Compiler Assisted Scheduling')

Redistribution des gains

- **Idée** : lorsqu'une tâche termine, on calcule la différence entre son WCET et son ACET. On redistribue le temps 'gagné' :
 - à la tâche qui débute son exécution [AyMe04]
 - à toutes les tâches en attente (selon le temps d'exécution [PiSh01], selon l'ordre ...)
- **Extension** : ('speculative speed reduction')
 - on anticipe que les tâches en attente n'utiliseront pas leur WCET
 - ... mais on fixe les vitesses de façon à pouvoir respecter les échéances si c'était le cas

Ordonnancement avec instrumentation du code

- **Idée:** évaluer en-ligne le pire temps d'exécution restant en insérant des points de mesure dans le code (ex: [ShLeKi01])
- **Principe:**
 - re-calcul du WCET restant après chaque section ('basic block')
 - répartition du gain sur les sections suivantes
- **Problèmes:**
 - nécessité d'instrumenter le code
 - choix de la bonne granularité d'instrumentation ??

Profils de consommation selon la stratégie

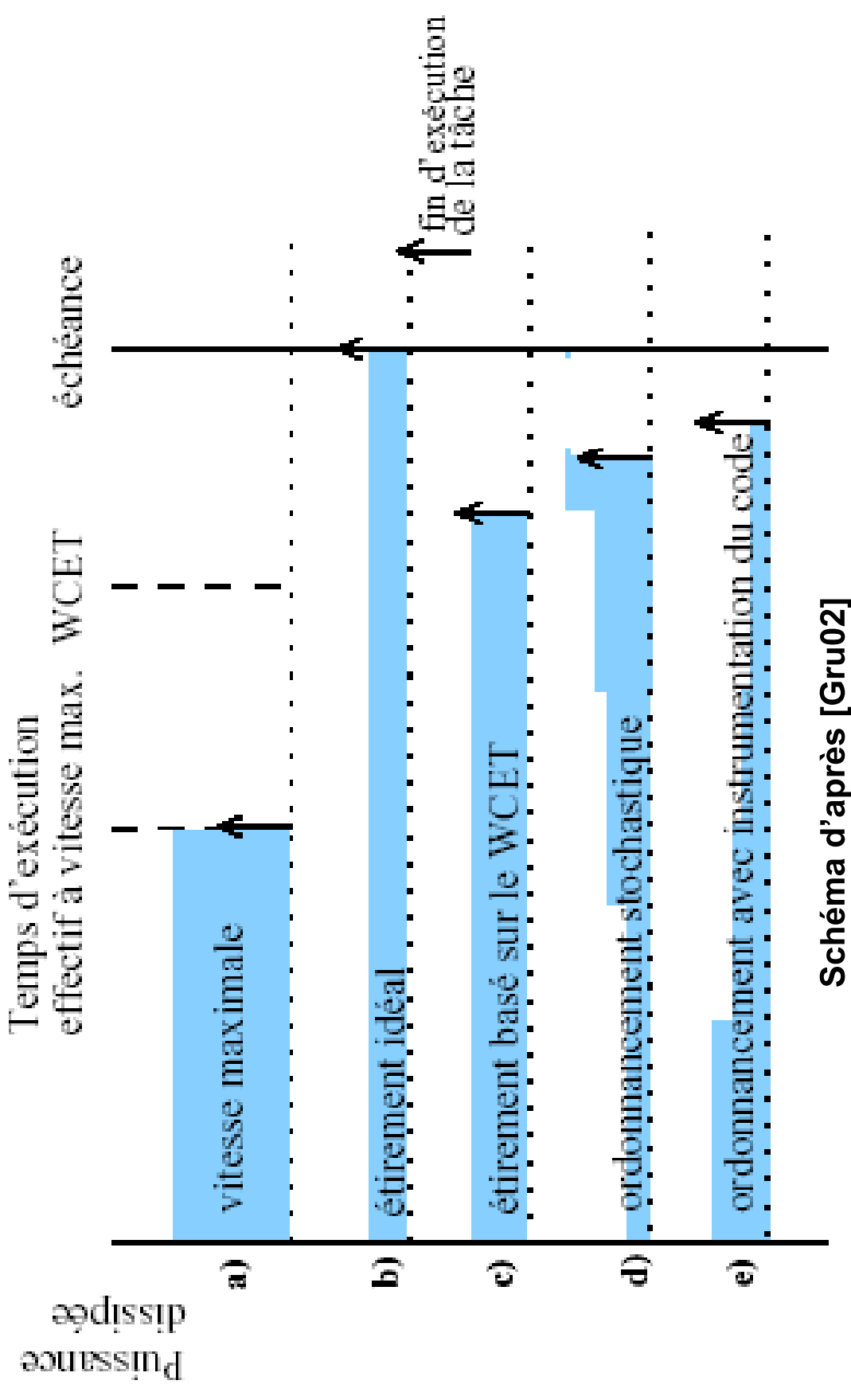
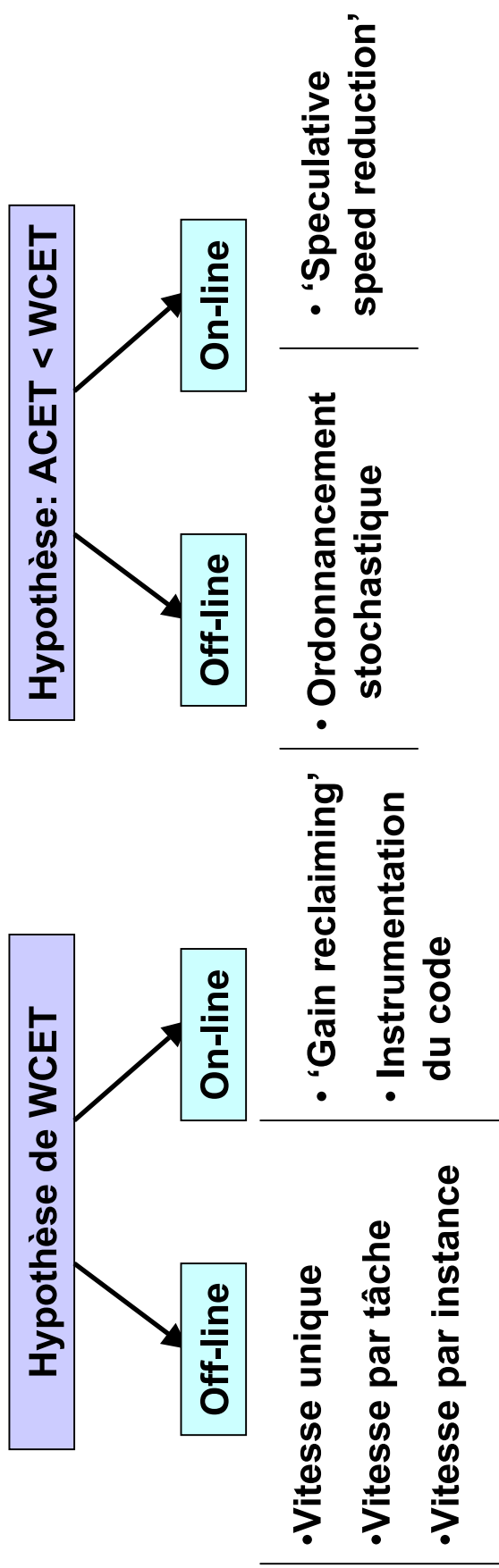


Schéma d'après [Gru02]

Conclusions (1/2)

- Le gain est fonction de la charge du système ...
- La stratégie d'ordonnancement sera choisie en fonction :
 - des contraintes de l'OS, du CPU et de l'application
 - de l'écart entre le pire temps d'exécution (WCET) et le temps d'exécution effectif (AET)



Conclusions (2/2)

- Des solutions d'ordonnancement efficaces existent dans la majorité des contextes ..
- Des modèles plus réalistes sont possibles:
 - prise en compte du temps de changement de fréquence et du temps minimum entre deux changements
 - toutes les instructions ne consomment pas la même énergie ..
 - modèles de batteries plus réalistes

⇒ Les stratégies d'ordonnancement s'insèrent dans une approche de conception « système » qui intègre compilation, OS, architecture matérielle etc...

Références

- [YuKi03] H.-S. Yun and J. Kim. *On energy-optimal voltage scheduling for fixed-priority hard real-time systems*. ACM TECS, 2003.
- [YaDeSh95] F. Yao, A. Demers, and S. Shenker. *A scheduling model for reduced CPU energy*. Proc. IEEE FOCS, 1995.
- [GaNaWa03] B. Gaujal, N. Navet, C. Walsh. *Shortest path algorithms for real-time scheduling with minimal energy use*. submitted to ACM TECS, 2003. Available as INRIA Research Report RR-4886.
- [GaNa04] B. Gaujal, N. Navet. *A new EDF feasibility test*. To appear as INRIA Research Report, 2004.
- [BrNa04] R. Brito, N. Navet. *Low Power Round-Robin Scheduling*. To appear in Proc. RTS'04.
- [GaNa03] B. Gaujal, N. Navet. *Ordonnancement sous Contraintes de Temps et d'Énergie*, Actes École Temps Réel, 2003.
- [Gru02] F. Gruian. *Energy-Centric Scheduling for Real-Time Systems*. PhD thesis, Lund Institute of Technology, 2002.
- [AyMe04] H. Aydin, R. Melhem, D. Mosse and P.M. Alvarez. *Power-aware Scheduling for Periodic Real-time Tasks*. To appear in IEEE Transactions on Computers.
- [PiSh01] P. Pillai and K. G. Shin. *Real-time Dynamic Voltage Scaling for Low Power Embedded Operating Systems*. In Proceedings of the 18th Symposium on Operating System Principles, October 2001.
- [ShLeKi01] D. Shin, S. Lee and J. Kim. *Intra-task Voltage Scheduling for Low-Energy Hard Real-time Applications*. In IEEE Design and Test of Computers, March 2001.