Algorithmes distribués synchrones et systèmes répartis asynchrones : concepts, mises en oeuvre et expérimentations

Michel Adam, Philippe Ingels, Michel Raynal

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél (1) 39 63 55 11

Rapports de Recherche

N° 862

ALGORITHMES DISTRIBUES SYNCHRONES ET SYSTEMES REPARTIS ASYNCHRONES : CONCEPTS, MISES EN OEUVRE ET EXPERIMENTATIONS

Michel ADAM
Philippe INGELS
Michel RAYNAL

JUILLET 1988

RR-0862

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042-RENNES CÉDEX
FRANCE
Téléphone: 99 36 20 00
Télex: UNIRISA 950 473 F
Télécopie: 99 38 38 32

Publication Interne n° 411
Juin 1988
28 Pages

ALGORITHMES DISTRIBUES SYNCHRONES ET SYSTEMES REPARTIS ASYNCHRONES : CONCEPTS, MISES EN OEUVRE ET EXPERIMENTATIONS *

Michel ADAM, Philippe INGELS, Michel RAYNAL
E-mail: raynal@irisa.fr

Résumé

La conception de certains algorithmes distribués et parallèles est fondée sur le concept de synchronisme; ce concept suppose que le comportement de chaque processus est exprimé en terme d'une séquence d'étapes, chaque étape se déroulant au même instant pour tous les processus. Après avoir precisé les définitions des modèles synchrones et asynchrones dans les systèmes distribués, nous étudions l'interprétation des algorithmes synchrones sur des systèmes asynchrones. Dans cette étude, deux types de synchronismes sont définis et leurs propriétés respectives sont captées par deux théorèmes; certaines relations entre les concepts de synchronisme et les mécanismes d'horloges logiques sont par ailleurs mises en évidence. Nous examinons ensuite les problèmes de synchronisation que posent la mise en œuvre des synchroniseurs et certaines de leurs solutions. Enfin, des résultats issus des expérimentations des synchroniseurs sur l'hypercube d'*Intel* sont présentés.

DISTRIBUTED SYNCHRONOUS ALGORITHMS AND ASYNCHRONOUS DISTRIBUTED SYSTEMS : CONCEPTS, IMPLEMENTATIONS AND EXPERIMENTATIONS

Abstract

Design of some parallel and distributed algorithms relies on the synchronism concept; roughly speaking this concept supposes that the behaviour of each process is expressed in term of a sequence of steps, each step occuring at the same time for all the processes. After precise definitions of the asynchronous and synchronous models for distributed computations, we study the interpretation of synchronous algorithms. Two kinds of synchronism are defined and their related properties are caught by two theorems. Some relations between synchronism concepts and logical clocks mechanisms are exhibited. Then, we examine synchronization problems due to synchronizers implementation and some of their solutions. Lastly, results of synchronizers experimentations on the *Intel* hypercube are presented.

*Ce travail a été en partie supporté par le Greco C3

Table des matières

1 Introduction

Le temps constitue l'un des objets les plus délicats à définir et à manipuler dans les systèmes et les applications informatiques. Tout algorithme nécessite, en effet, au moins un processeur pour réaliser le calcul ou le contrôle auquel il est dévolu. Ainsi, bien que n'ayant pas besoin logiquement de temps, l'exécution d'un algorithme va en consommer : l'allocation de ce temps va alors se confondre avec l'allocation des ressources que nécessite l'exécution de l'algorithme. Dans le cas de systèmes d'exploitation (ou d'applications parallèles formées de plusieurs entités), la présence de ressources à accès exclusif, dont l'utilisation consomme du temps peut conduire à retarder l'exécution de certains processus sans que cela ne modifie leurs résultats.

A coté de ce temps *support d'exécution*, le temps peut également être l'un des objets manipulés par l'algorithme, au même titre que les entiers ou les booléens par exemple. Le temps doit alors avoir une sémantique bien définie puisque celle-ci participe à la sémantique de l'algorithme. Il existe généralement deux appréhensions du temps à ce niveau : le temps *date* et le temps *durée*. Le premier fait référence à une quantité mesurée à partir d'une origine et le second à un nombre permettant de définir un intervalle entre deux dates, indépendamment de celles-ci. Dans les deux cas, la mesure utilisée est fondée sur le concept d'unité de temps.

A coté de ce temps mesurable, il existe une autre perception du temps, essentiellement logique, où l'on ne s'intéresse ni à la durée, ni à la valeur absolue de la date, mais à la comparaison entre deux dates, c'est à dire aux notions de avant et de après. Les processus du système sont alors considérés comme le siège d'événements qui sont liés par une relation de précédence. Si l'on considère un temps abstrait global, identique pour tous les processus, une relation de précédence temporelle précise leur ordre d'exécution par rapport à ce temps global. Dans un certain nombre de systèmes (notamment les systèmes répartis) un tel ordre est inobservable. De manière plus générale, on s'intéresse à une relation de précédence causale entre événements : si tel événement est la cause de tel autre, alors il doit le précéder quel que soit le référentiel de temps ; le temps est alors perçu comme un ordre partiel sur l'ensemble des événements.

Ces différentes perceptions du temps ont été appréhendées (en tout ou partie) dans un certain nombre de langages [7,4] et des systèmes centralisés ou répartis [8,9]. Elles conduisent à mettre en œuvre un ensemble de mécanismes et de règles permettant d'énoncer et de réaliser le contrôle nécessité par l'allocation des ressources, la coordination des processus, la définition du temps (objet de programmation) et l'exploitation des relations de précédence causale ; c'est à dire, tout ce que l'on regroupe généralement sous le terme de *synchronisation.*

Le développement de certains algorithmes parallèles a introduit un autre concept lié au temps : le *synchronisme.* Ce concept recouvre deux choses : simultanéité et périodicité. Dans une première approche, on peut donc dire qu'un ensemble de processus évolue de manière synchrone si leurs comportements peuvent être exprimés en terme de ce qui se passe à une *étape* de calcul (qu'ils exécutent simultanément) et si l'exécution de l'ensemble de processus consiste en une succession de telles étapes [1].

Le but de ce rapport est l'étude du concept de synchronisme dans le contexte de

systèmes répartis, c'est à dire, lorsque les interactions entre processus/processeurs ne peuvent se faire que par échange de messages [13]. La partie 2 commence par définir les systèmes asynchrones (§2.1), puis synchrones (§2.2), avant de montrer comment simuler un comportement synchrone des processeurs dans un système asynchrone (§2.3). On étudie ensuite (§2.4) l'interprétation des algorithmes synchrones sur des systèmes asynchrones ; après avoir énoncé les problèmes rencontrés, on montre que diverses notions de synchronisme sont envisageables, en fonction des problèmes à résoudre. A partir des propriétés dont doivent rendre compte les interpréteurs de synchronisme (appelés synchroniseurs), la partie 3 propose des mécanismes de synchronisation adéquats et présente plusieurs mises en œuvre de synchroniseurs. Enfin la partie 4 exhibe des résultats issus de l'expérimentation de certains synchroniseurs mis en œuvre sur l'hypercube d'*Intel*.

L'étude du synchronisme s'inscrit dans le cadre plus général de la ***dérivation modulaire*** d'algorithmes distribués. L'utilisation du parallélisme et d'un modèle synchrone permet de concevoir des algorithmes distribués plus faciles à énoncer, à prouver et souvent plus efficaces que leurs analogues asynchrones. La définition des interpréteurs que sont les synchroniseurs permet d'autre part de rendre effectif ces algorithmes. L'ensemble, algorithme synchrone (résolvant un problème particulier) et synchroniseur (offrant le modèle de synchronisme) donne en effet en résultat un algorithme distribué pouvant être exécuté sur tout réseau asynchrone.

2 Systèmes synchrones et synchroniseurs

2.1 Le modèle asynchrone

Un système asynchrone est composé d'un réseau de communication point à point connectant un ensemble fini de sites (ou processeurs) par des canaux bidirectionnels. Un tel système peut être modélisé par un graphe $G = (X, U)$ où X représente les sites et U les canaux ; G non dirigé est supposé connexe. Il n'y a pas de mémoire centrale partagée par les processeurs. En conséquence tous les échanges se font à l'aide de messages véhiculés par les canaux. Un site peut recevoir des messages de ses voisins, exécuter des calculs locaux et envoyer des messages à ses voisins. Les événements correspondants sont respectivement appelés : réception, événement interne et émission. Le délai de transfert des messages est fini mais non prévisible. Les canaux sont supposés fiables (c'est-à-dire qu'ils ne perdent pas les messages) et fifo (c'est-à-dire l'ordre de réception est celui d'émission). Les durées d'émission, de réception et de traitement des messages sont négligeables. On les supposent nulles, c'est-à-dire *absorbées* dans le temps de transfert.

Un algorithme asynchrone est un algorithme distribué conçu pour être exécuté par un système asynchrone. On supposera, sans perte de généralité, que le réseau de processus défini par l'algorithme se superpose au système de la façon suivante : un processus pour un processeur ; il existe un canal entre deux processeurs si et seulement si les processus qui leur sont associés communiquent directement.

Il est important de constater que dans un algorithme/système asynchrone, les processus/processeurs évoluent à des vitesses *a priori* indépendantes les unes des autres, les envois et réceptions de messages étant les seuls moyens de les synchroniser.

2.2 Le modèle synchrone

Au point de vue de la structure, il n'y a pas de différence entre un système synchrone et système asynchrone. C'est dans le comportement des processeurs et des canaux que celle-ci réside.

1. Synchronisme des processeurs :
   L'évolution des processus est cadencée par une horloge globale qui produit des signaux (signal de pulsation).

   - A chaque signal de pulsation :
     - une variable globale (numéro de pulsation) est incrémentée de un ; les numéros de pulsation prennent donc les valeurs 0,1,2,...
     (les processeurs peuvent consulter le numéro de pulsation) ;
     - chaque processeur, pour le compte des processus qu'il exécute, peut émettre un nombre fini de messages vers ses voisins.
   - Entre deux signaux de pulsation :
     - tout processeur peut réaliser des actions locales et des réceptions de messages.

   On appellera *pulsation p*, la période de temps qui commence au moment où le numéro de pulsation prend la valeur p et se termine au moment où il prend la valeur $p+1$.

2. Synchronisme des canaux :
   Le délai de transfert d'un message sur tout canal est inférieur à une pulsation.

On suppose comme précédemment que les durées d'émission, réception et traitement des messages sont nulles, c'est-à-dire sont *absorbées* dans le temps de transit.

Il est facile de montrer que la propriété I suivante est toujours vérifiée dans un système synchrone. Appelons $E_i(p)$ l'ensemble fini des événements dont le site P_i est le siège à la pulsation p de l'horloge globale :

$$\forall P_i, P_j \in X, \forall p \in N \ :$$

$$\{(envoyer\ m\ à\ j) \in E_i(p)\} \Leftrightarrow \{(recevoir\ m\ de\ i) \in E_j(p)\}$$

Comme on le voit un système synchrone définit une périodicité (les pulsations de l'horloge). Durant la pulsation p tous les processeurs exécutent leur $p^{ième}$ étape de calcul. Lorsque P_i démarre sa $p^{ième}$ étape, il acquiert *de facto* les connaissances globales suivantes :

- $CG_1(p) = \forall j : P_j$ démarre sa $p^{ième}$ étape et a donc reçu et traité tous les messages qui lui ont été envoyé à *p-1*,

- $CG_2(p) = \forall k : P_k$ connait $CG_1(p)$,

...

- $CG_x(p) = \forall l : P_l$ connait $CG_{x-1}(p)$.

Comme on le voit, un système synchrone rend possible l'acquisition d'une telle connaissance commune [10] :

$$CG(p) = \bigcup_{x \geq 1} CG_x(p)$$

Un algorithme distribué synchrone est un algorithme distribué conçu pour être exécuté par un système synchrone. Un tel algorithme repose donc sur l'hypothèse d'une horloge globale cadençant son évolution et sur l'hypothèse de canaux synchrones.

Le problème que l'on se pose est la construction d'interpréteurs d'algorithmes synchrones construits sur des systèmes asynchrones. De tels interpréteurs sont appelés synchroniseurs [2]. Le couple formé par un tel synchroniseur σ et un algorithme synchrone S constitue un algorithme asynchrone A dont l'exécution doit produire le même résultat que S exécuté sur un système synchrone. Avant d'étudier les problèmes posés par la mise en œuvre des synchroniseurs, nous montrons comment simuler le synchronisme du seul ensemble de processeurs. La difficulté supplémentaire que doit résoudre un synchroniseur est de simuler le synchronisme des communications (le délai de transfert borné).

2.3 Simuler des processeurs synchrones

Rendre les processeurs synchrones consiste à mettre en œuvre une horloge globale qui cadence leurs évolutions. Pour cela, on peut associer à chaque processeur une horloge locale. A chaque événement, défini par l'algorithme qu'il exécute, le processeur incrémente son horloge. Les processeurs ayant leur vitesse propre, l'ensemble brut de ces horloges locales ne met en œuvre aucune propriété du temps fourni par une horloge globale :

1. ni la notion de date globale (la valeur fournie par l'horloge que tous les processeurs voient, a la même valeur au même instant) : Chaque processeur évolue en effet, à sa propre vitesse ;

2. ni la notion de durée (entre deux valeurs $t1$ et $t2$ de l'horloge globale, le même laps de temps doit s'écouler sur chacun des sites) ;

3. ni la notion de précédence :
   un message m émis alors que l'horloge locale de l'émetteur P_i a la valeur x peut être reçu alors que l'horloge locale du récepteur P_j a la valeur y avec $y < x$.

Sans hypothèses supplémentaires, on ne peut mettre en œuvre les propriétés 1 et 2. On peut cependant construire la propriété 3 sur la précédence, c'est à dire, dans le cas précédent, garantir $y \geq x$. L'égalité $x = y$ revient alors à considérer que dans le temps global simulé le

délai de transfert est nul. Dans le cas $y > x$, la différence $y - x$ donnera la durée de transfert dans le temps simulé. Du point de vue de l'horloge globale simulée, un message ne pourra alors être reçu avant d'avoir été émis : la précédence imposée par la causalité sera toujours garantie.

Simuler le synchronisme des seuls processeurs revient donc à construire un temps global (simulé) permettant de dater les événements de façon cohérente. Pour cela, chaque processeur P_i est doté d'une horloge logique locale h_i qu'il incrémente après chaque événement local.

A tout message émis m est associé la valeur de l'horloge logique de son émetteur P_i. A la réception d'un tel couple (m, h) par le récepteur P_j, deux attitudes sont possibles pour garantir un temps de transit non négatif du point de vue de l'horloge globale que l'on veut réaliser :

1. effectuer, si nécessaire, un recalage de l'horloge locale du récepteur :

$$h_j := max(h_j, h)$$

   et considérer la nouvelle valeur de h_j comme la date de l'événement de réception.

2. de manière duale, conserver le couple (m, h) et ne délivrer m au récepteur que lorsque $h_j > h$.

Il est facile de montrer que la simulation de l'horloge globale générant les pulsations qui cadence l'évolution des processeurs, réalisée par les horloges logiques locales $(h_1, \ldots, h_i, \ldots, h_n)$ et l'un ou l'autre de ces deux protocoles est correcte. C'est à dire qu'elle simule un réseau de processeurs synchrones reliés par un système de communication asynchrone.

Le premier des deux protocoles présentés est le protocole, désormais classique, des horloges logiques de Lamport [8]. Le second a été proposé simultanément par Welsh [14] et Neiger et Toueg [11]. [14] l'étudie sous diverses hypothèses de pannes, alors que [11] l'introduit pour simplifier la conception et la preuve des algorithmes distribués et caractérise formellement la classe des problèmes auxquels cette simulation peut être appliquée.

Il est important de remarquer qu' il ne s'agit là que de simulation. Le synchronisme des processeurs n'est que logique : à un instant donné, les valeurs des horloges locales peuvent être différentes. Toutefois, la différence entre les horloges locales de deux processeurs quoique finie ne peut être bornée *a priori*.

2.4 Interpréter les algorithmes synchrones

2.4.1 Les problèmes posés

Un interpréteur *parfait* d'algorithme synchrone doit assurer :

$P1$: la périodicité, c'est à dire, produire à des intervalles réguliers des signaux de pulsation sur tous les sites ;

$P2$: la simultanéité, c'est à dire, tous les processeurs doivent être à la même pulsation à un instant donné ;

$P3$: la communication synchrone, c'est à dire, tout message émis à la pulsation p doit être reçu à la pulsation p.

Il n'est pas possible de mettre en œuvre l'égalité des intervalles de temps relatifs à la périodicité. En effet, cela suppose que l'on sache borner la durée des étapes de calcul des processus et les délais de communication. Cette borne définissant alors la durée de la pulsation, c'est à dire l'unité de temps du système synchrone construit. Or le réseau sur lequel sont véhiculés les messages de l'algorithme synchrone, est asynchrone et donc leurs délais de transfert ne peuvent être bornés *a priori*. La durée de la pième pulsation interpretée sera donc déterminée, entre autres facteurs, par le plus long délai de transit durant cette pulsation. Appelons δ_i cette durée. L'interpréteur ne pourra donc faire mieux que de générer une séquence de pulsations 1,2,3,... de durées respectives $\delta_1, \delta_2, \delta_3, \ldots$ avec des δ_i indépendants.

On peut concevoir deux types de synchroniseurs, suivant qu'ils réalisent les propriétes $P2$ et $P3$ (synchronisme fort §2.4.2) ou, seulement la propriété $P3$ (synchronisme faible §2.4.3). Quel que soit le type de synchronisme réalisé, on dote chaque processeur P_i d'une horloge logique locale pul_i qui lui donne le numéro de pulsation en cours :

$$pul_i \ : \ R^+ \ \mapsto \ N$$

$pul_i(t) = p$ signifie, pour un hypothétique observateur global omniscient, que la valeur à l'instant global t de la variable pul_i est p.

Le rôle de tout synchroniseur est dès lors :

1. de générer sur chaque processeur des séquences de pulsations satisfaisant $P2$ et $P3$ ou seulement $P3$;
2. d'interpréter le texte des processus de l'algorithme synchrone.

2.4.2 Le synchronisme fort

Afin de générer la séquence de pulsations et d'assurer la simultanéité voulue, un synchroniseur va utiliser un certain nombre de messages de contrôle et se multiplexer entre l'interprétation proprement dite de l'algorithme synchrone et la réalisation du contrôle. Soient $dp_i(p)$ et $fp_i(p)$ les instants, mesurés dans le référentiel de temps global de l'observateur omniscient, auxquels la pulsation p sur le processeur P_i, respectivement est démarrée et est terminée. En d'autres termes, si $[dt_x, ft_x]$ est l'intervalle de temps durant lequel est exécuté l'événement x, on a :

$$\forall i, p, \{x \ \in \ E_i(p)\} \ \Leftrightarrow \ \{[dt_x, ft_x] \ \subseteq \ [dp_i(p), fp_i(p)[\}$$

Théorème 2.1 *Tout synchroniseur simulant le synchronisme fort doit maintenir l'invariant suivant :*

$$\forall p \in N \ : \ \left\{ \min_{i \in X} \{dp_i(p)\} \ \geq \ \max_{j \in X} \{fp_i(p-1)\} \right.$$

Preuve :
Soit P_j le dernier processeur à terminer la pulsation $p-1$ et P_i le premier à commencer la pulsation p. Supposons $dp_i(p) < fp_j(p-1)$. Si $i=j$, on ne peut avoir $dp_i(p) < fp_j(p-1)$. On n'examine donc que le cas où $i \neq j$. La valeur de la pulsation sur P_i à l'instant $dp_i(p)$ est, par définition p :

$$pul_i(dp_i(p)) = p$$

La valeur de la pulsation sur P_j à l'instant $dp_i(p)$ est inférieure ou égale à $p-1$, car $dp_i(p) < fp_j(p-1)$:

$$pul_j(dp_i(p)) \leq p-1$$

Ce qui signifie qu'à un instant donné, $dp_i(p)$, deux variables pulsations ont des valeurs différentes :

$$\exists t = dp_i(p),\ \exists P_i, P_j \in X,\ \ pul_i(t) \neq pul_j(t)$$

Cette proposition contredit la propriété $P2$ du synchronisme fort. □

Corollaire 2.1 *Tout synchroniseur simulant le synchronisme fort maintient :*

$$\forall P_i, P_j \in X,\ \forall p_1, p_2 \in N\ :$$

$$\{p_1 \neq p_2\} \Rightarrow \{[dp_i(p_1), fp_i(p_1)[\cap [dp_j(p_2), fp_j(p_2)[= \emptyset\}$$

Preuve :
Supposons faux le corollaire précédent :

$$\exists p_1, p_2 \in N,\ p_1 > p_2,\ \exists P_i, P_j \in X \textit{ tels que}$$

$$[dp_i(p_1), fp_i(p_1)[\cap [dp_j(p_2), fp_j(p_2)[\neq \emptyset$$

Prenons une date t appartenant à l'intersection considérée. A cet instant, la valeur de la pulsation sur le site P_i est p_1 et celle sur le site P_j a pour valeur p_2. Comme p_1 et p_2 sont différents, on a :

$$\exists t \in R^+,\ \exists P_i, P_j \in X,\ \ pul_i(t) \neq pul_j(t)$$

Ce qui contredit la propriété $P2$ du synchronisme fort. □

En conséquence un synchroniseur satisfaisant le théorème 2.1 assure que lorsqu'un processeur interprète le texte relatif à la pulsation p d'un processus :

- aucun processeur P_j n'interprète un calcul relatif à une pulsation $q \neq p$;
- tous les messages envoyés à la pulsation $p-1$ par les processus de l'algorithme synchrone ont été reçus et traités.

Nous dirons qu'un tel synchroniseur réalise un synchronisme fort.

2.4.3 Le synchronisme faible

Le synchronisme décrit précédemment est *fort*, au sens du corollaire : deux processus de l'algorithme synchrone ne peuvent être, à un instant donné, à des pulsations différentes. Dans un certain nombre d'applications, cette contrainte peut être affaiblie : la propriété $P3$ doit toujours être vérifiée, mais deux processus peuvent être à des pulsations différentes. Nous appellerons un tel affaiblissement synchronisme faible.

Lemme 2.1 *Tout interpréteur de synchronisme faible doit maintenir l'invariant suivant :*

$$\forall P_i, P_j \in X \ : \ P_i \text{ et } P_j \text{ voisins directs}, \ \forall t \ \in \ R^+ \ :$$

$$| \ pul_i(t) \ - \ pul_j(t) \ | \leq \ 1$$

Preuve :
A l'instant t, un site, P_i est à la pulsation $p-1$. Au même instant, un autre site, P_j, voisin de P_i, est à la pulsation p. P_j ne peut passer à la pulsation $p+1$ avant d'avoir reçu tous les messages qui lui ont été adressés à la pulsation p. Mais, il ne peut pas, *a priori*, savoir si P_i lui en enverra à la pulsation p. Cette connaissance ne pourra être acquise que quand P_i aura terminé les traitements relatifs à la pulsation $p-1$ et sera passé à la pulsation p. Il s'en suit que deux sites voisins ne peuvent, à un instant donné, avoir une différence de valeur de pulsations supérieure à 1. □

Théorème 2.2 *Tout interpréteur de synchronisme faible maintient l'invariant suivant (d est le diamètre du réseau de processus de l'algorithme synchrone) :*

$$\forall P_i, P_j \in X, \ \forall t \ \in \ R^+ \ : | \ pul_i(t) \ - \ pul_j(t) \ | \leq \ d$$

Preuve :
La preuve peut se faire par récurrence sur la distance entre les sites. La distance entre un site P_i et un site P_j, notée $d(i,j)$, est le nombre minimum d'arcs qu'il faut parcourir dans le graphe G pour passer de P_i à P_j.

- $d(i,j) = 1$:
  d'après le lemme 2.1, le premier pas de récurrence est immédiat :

$$\forall t \ \in \ R^+ \ : | \ pul_i(t) \ - \ pul_j(t) \ | \leq \ 1$$

- $d(i,j) = l$:
  La propriété est supposée vraie au rang l :

$$\forall t \ \in \ R^+ \ : | \ pul_i(t) \ - \ pul_j(t) \ | \leq \ l$$

- $d(i,j) = l+1$:
  La distance entre P_i et P_j étant $l+1$, il existe un site P_k tel que $d(i,k) = l$ et $d(k,j) = 1$. En utilisant respectivement l'hypothèse de récurrence et le lemme 2.1, on obtient les deux relations suivantes :

$$\forall t \in R^+ : | pul_i(t) - pul_k(t) | \leq l$$

$$\forall t \in R^+ : | pul_k(t) - pul_j(t) | \leq 1$$

  En additionnant les deux inégalités :

$$\forall t \in R^+ : | pul_i(t) - pul_j(t) | \leq l+1 = d(i,j)$$

Entre deux sites quelconques du réseau, la différence de valeur de pulsations à un instant donné, ne peut être supérieure à la distance entre ces deux sites. Dans un graphe, la distance maximum entre deux sites est le diamètre (d). Ce qui démontre le théorème. □

Remarque : la connaisance commune $CG(p)$ ne peut donc être une connaissance commune avec un synchroniseur faible.

2.5 Schéma récapitulatif

il est possible de représenter les simulations et interprétations vues précédemment par un schéma unique. En ordonnées sont placés les canaux qui peuvent être asynchrones A ou synchrones S. En abscisses sont placés les processus qui peuvent, à un instant t :

- être tous à la même pulsation : SF,
- avoir une différence bornée entre leurs numéros de pulsation : Sf,
- avoir des horloges qui ne mettent pas en défaut la causalité : Df,
- ne pas avoir d'horloges : A.

Le passage du point (A,A) au point (A,Df) est réalisé par les protocoles de Lamport et de Neiger-Toueg (cf §2.3). Les synchroniseurs faibles assurent le passage du point (A,A) au point (S,Sf), alors que les synchroniseurs forts font passer de (A,A) à (S,SF). Tous les synchroniseurs réalisent la périodicité (séquence des pulsations), mais ne peuvent réaliser l'égalité des durées des périodes. Ils se distinguent par la façon dont ils rendent compte (physique ou logique) de la simultanéité.

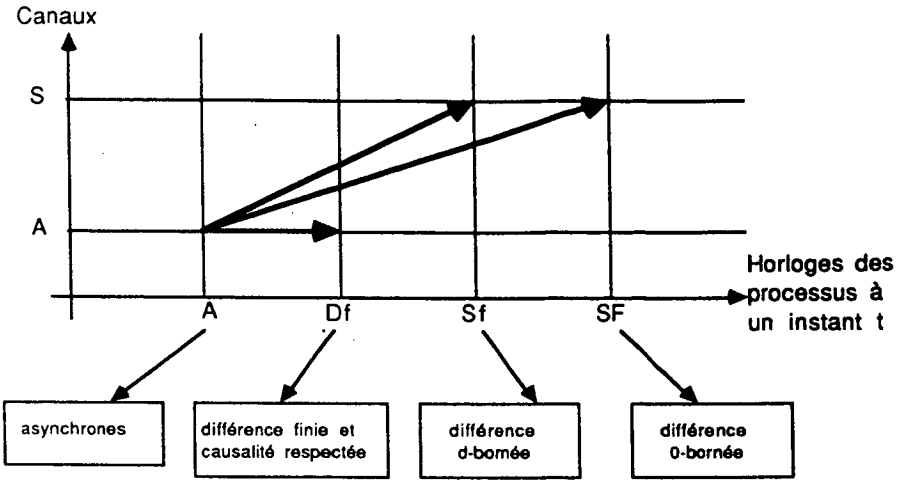

Figure 1 : Schéma récapitulatif

3 Mise en œuvre d'un système synchrone

3.1 Problèmes posés et principes de mise en œuvre

3.1.1 notion de sureté

On a vu qu'un interpréteur d'algorithme synchrone (synchroniseur) doit, dans tous les cas, assurer une communication synchrone (propriété $P3$) : tout message émis à la pulsation p doit être reçu à la pulsation p.

Un site P_i ne peut donc passer de la pulsation p à la pulsation $p+1$ que s'il est certain d'avoir reçu tous les messages qui lui ont été adressés à la pulsation p. P_i ne peut évidemment prendre cette décision sans information complémentaire de ses voisins, car il ne sait pas *a priori* combien de messages il doit recevoir lors d'une pulsation.

On introduit, pour résoudre ce problème, une propriété locale à un site et liée à son activité d'émetteur de messages, *la sureté* [2] : un site est dit *sûr par rapport à une pulsation* p dès lors qu'il ne pourra plus provoquer d'actions de l'algorithme synchrone dans le réseau, relativement à la pulsation p.

Les divers interpréteurs se distinguent les uns des autres par la façon dont, pour chaque site et à chaque pulsation, ils communiquent l'information *"je suis sûr par rapport à la pulsation courante"*.

Sûreté dans le synchronisme faible

Pour assurer le synchronisme faible (communication synchrone), il faut respecter la propriété suivante. P_i peut passer à la pulsation $p+1$ si, et seulement si :

- $Pf1$: P_i a terminé le traitement de la pulsation p ;
- $Pf2$: P_i sait que tous ses voisins sont sûrs par rapport à la pulsation p.

La propriété $Pf1$ assure simplement que P_i ne passe pas à la pulsation $p+1$ avant d'avoir terminé la pulsation p. La propriété $Pf2$ assure la communication synchrone.

On voit que le synchronisme faible ne nécessite la coordination d'un site qu'avec *ses voisins dans le réseau* (coordination "locale").

Sûreté dans le synchronisme fort

Pour assurer le synchronisme fort (communication synchrone, plus simultanéité), il faut respecter la propriété suivante. P_i peut passer à la pulsation $p+1$ si, et seulement si :

- $PF1$: P_i a terminé le traitement de la pulsation p ;
- $PF2$: P_i sait que tous les sites du réseau sont sûrs par rapport à la pulsation p.

Les propriétés $Pf1$ et $PF1$ sont identiques. L'apport de $PF2$ par rapport à $Pf2$ est la simultanéité : il n'y a pas deux sites à des pulsations distinctes au même moment.

Le synchronisme fort nécessite la coordination de chaque site avec ***tous les autres sites du réseau*** (coordination "globale").

3.1.2 Classes de solutions

La mise en œuvre du synchronisme fort ou faible nécessite la coordination de chaque site avec d'autres sites. On a deux grandes classes de solutions :

- celle dans lesquelles les sites échangent des messages de sûreté avec tous leurs voisins ;
- celles dans lesquelles les sites n'échangent des messages de contrôle relatifs à la sûreté, qu'avec un sous-ensemble de leurs voisins, dans le but de limiter le nombre de messages de contrôle envoyés.

La première classe de solutions, pour laquelle des messages de contrôle circulent sur chaque arc, nécessite *a priori* un plus grand nombre de messages que la deuxième où seul un sous-ensemble des arcs est utilisé. Par contre, le temps de basculement d'une pulsation à une autre est supérieur avec la deuxième solution car les messages de contrôle doivent alors circuler sur plusieurs arcs pour être communiqués entre voisins (le lecteur intéressé trouvera dans [2] une étude comparative en nombre de messages et en temps de ces deux types de solutions).

3.2 Mise en œuvre du synchronisme faible

3.2.1 Première classe de solutions

Dans le cas où les sites échangent les messages de contrôle relatifs à la sûreté avec tous leurs voisins, la solution est simple. Tout site P_i à la pulsation p :

- envoie un message de contrôle *sûr* vers tous ses voisins dès qu'il a émis tous les messages pour le compte de l'algorithme synchrone qu'il interpréte, relatifs à la pulsation p ;

- attend d'avoir reçu un message *sûr* de chacun de ses voisins pour passer à la pulsation $p+1$ (les canaux étant fifo, il a alors reçu les messages envoyés par ses voisins, relatifs à la pulsation p).

Remarque : Dans [2], Awerbuch propose un synchroniseur appelé α qui utilise le principe qui vient d'être énoncé. La condition de sûreté qu'il utilise est plus forte : un site n'y est sûr que lorsque que tous les messages qu'il a émis à la pulsation p ont été reçus (ce qui nécessite un mécanisme d'acquittement supplémentaire).

3.2.2 Deuxième classe de solutions

Dans ce cas, la solution est plus délicate car un site ne communique pas directement d'information de contrôle avec tous les sites avec qui il doit se coordonner (ses voisins). A la pulsation p, le site P_i n'émet l'information *je suis sûr* que vers certains de ses voisins et il faut donc rajouter un *mécanisme de diffusion* permettant de communiquer cette information à tous les voisins de P_i.

De plus, l'existence de canaux sur lesquelles ne circulent pas de messages de contrôle pose le problème suivant : Considérons le schéma suivant (figure 2), dans lequel P_i et P_j sont deux sites voisins connectés par le canal c sur lequel ne circulent pas de messages relatifs à la sûreté. Ces derniers sont échangés entre P_i et P_j par le chemin *ph*. Il est possible dans cette

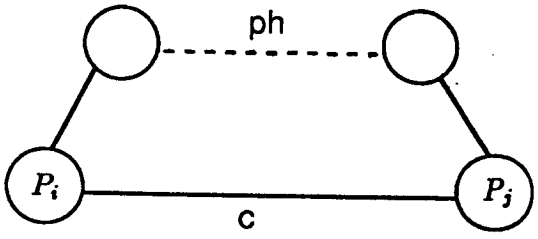

Figure 2

configuration que, P_i ayant envoyé un message m sur le canal c, puis un message de contrôle via le chemin *ph*, P_j reçoive d'abord le message de contrôle puis le message m. Tout se passe alors comme si logiquement, le message de contrôle avait doublé le message m. Et, lorsque P_j reçoit ce dernier, il peut donc se trouver à la pulsation $p+1$. Une solution simple acceptable consiste à n'autoriser un site P_i à émettre le message *je suis sûr par rapport à la pulsation p* que lorsque tous les messages qu'il a émis ont été effectivement reçus. Un site peut facilement savoir que les messages émis ont été reçus à l'aide d'un mécanisme d'acquittement. L'ordre de réception entre le message m et le message de contrôle est alors le même que l'ordre d'émission.

On sait donc garantir qu'un site P_i ne peut passer à la pulsation $p+1$ avant d'avoir reçu tous les messages relatifs à la pulsation p. Il faut de plus garantir que tout site P_i ne consomme, à la pulsation p, que les messages relatifs à cette pulsation ; l'exemple suivant, illustré comme auparavant par la figure 2, montre que, sans précaution supplémentaire, cela peut ne pas être le cas. Comme précédemment, ph représente le chemin utilisé par les messages de contrôle allant de P_i à P_j. Supposons que P_i sache que tous ses voisins, en particulier P_j, sont sûrs par rapport à la pulsation p. Il peut donc passer à la pulsation $p+1$, puis émettre un message de calcul m vers P_j. Mais ce dernier peut, à la réception, ne pas encore savoir que P_i est sûr. Il s'agit ici du problème dual du précédent : tout se passe comme si le message de calcul m avait "doublé" le message de contrôle entre P_i et P_j. Dans ce cas, P_j ne sait pas si m à été émis à la pulsation p (il doit alors le consommer) ou à la pulsation $p+1$ (il doit alors attendre pour le consommer). Une manière de lever l'ambiguïté consiste à numéroter les messages avec le numéro de pulsation. La propriété du synchronisme faible énoncée au §2.4.3 permet de faire cette numérotation modulo 2.

On voit donc que pour assurer le synchronisme faible, trois mécanismes sont nécessaires lorsqu' un site n'échange pas de messages de contrôle avec tous ses voisins :

- un mécanisme de diffusion d'information de contrôle (pour prévenir les voisins) ;
- un mécanisme garantissant que les messages de l'application synchrone n'arrivent pas trop tard, c'est-à-dire ne sont pas doublés par des messages de contrôle (mécanisme d'acquittement et de sûreté);
- un mécanisme garantissant que les messages de l'application synchrone ne sont pas consommés trop tôt, c'est-à-dire n'ont pas doublé des messages de contrôle (mécanisme d'identification de messages).

Le synchroniseur δ de Peleg et Ullman [12] est un exemple d'algorithme de ce type dans lequel les messages de contrôle circulent sur un sous-graphe t-couvrant du réseau, la diffusion de l'information se faisant grâce à un algorithme à phases [3].

3.3 Mise en œuvre du synchronisme fort

Mettre en œuvre le synchronisme fort revient à adjoindre la simultanéité au synchronisme faible. Tout ce qui a été dit sur la manière d'assurer le synchronisme faible reste donc vrai.

La coordination nécessaire à un site pour passer à la pulsation suivante doit se faire avec *tous* les sites du réseau. Cette coordination globale peut être réalisé soit de manière *non centralisée* (chaque site décide de passer à la pulsation suivante à partir d'informations provenant des autres sites), soit de manière *centralisée* (un ou plusieurs sites étant chargés de décider du passage à la pulsation suivante).

3.3.1 Solution non centralisée

Dans ce cas un site *sûr* le fait savoir à tous les autres sites du réseau grâce à un mécanisme de diffusion globale [6]. Quand un site a terminé le traitement de la pulsation p et qu'il a reçu

un message *je suis sûr par rapport à la pulsation p* de chacun des autres sites du réseau, il décide de passer à la pulsation $p+1$.

3.3.2 Solution centralisée

Il existe un ou plusieurs sites privilégiés appelés *coordinateurs*. Dans la suite, on se limitera au cas d'un seul coordinateur. Ce site particulier est chargé de prendre la décision de faire passer l'ensemble des sites du réseau à la pulsation suivante. Le passage de la pulsation p à la pulsation $p+1$ peut être décrit de la manière suivante :

1. quand un site est *sûr* par rapport à la pulsation p, il le fait savoir au coordinateur ;
2. quand le coordinateur sait que tous le sites sont *sûrs* par rapport à la pulsation p, il décide de passer à la pulsation $p+1$, et il le fait savoir à tous les sites ;
3. quand un site apprend que le coordinateur a décidé le changement de pulsation, il commmence le traitement de la pulsation $p+1$.

Le principe peut être mis en œuvre de diverses manières suivant les méthodes utilisées pour remonter l'information relative à la sûreté vers le coordinateur et pour diffuser les informations du coordinateur vers tous les sites.

le synchroniseur β proposé par Awerbuch [2] , représente une telle solution, dans laquelle la remontée de l'information relative à la sûreté vers le coordinateur et la diffusion de l'information à partir du coordinateur se font grâce à une arborescence couvrante du réseau initial. D'autres solutions sont envisageables [6] en utilisant d'autres structures de parcours telles que l'anneau par exemple. Il faut remarquer que dans ce cas, on assure la communication synchrone sans échanger directement de l'information de contrôle avec tous ses voisins. Cela nécessite donc l'utilisation des outils décrits §3.2.2.

4 Quelques résultats sur la mise en œuvre des synchroniseurs

4.1 Pourquoi faire des expérimentations sur une machine réelle?

Nous avons présenté les propriétés attendues des synchroniseurs, ainsi que les mécanismes nécessaires à leur mise en œuvre. La complexité des différents synchroniseurs en temps et en nombre de messages est connue [12,2], mais ces valeurs ne décrivent pas entièrement le comportement que peuvent avoir ces algorithmes dans la réalité. D'une part, les paramètres *temps d'exécution* et *nombre de messages échangés* ne sont pas indépendants dans les systèmes réels (le temps de transfert d'un message de taille donné peut augmenter avec le nombre de messages circulant). D'autre part, ces complexités sont des valeurs asymptotiques prenant toute leur signification lorsque le nombre de processus devient grand, alors que ce nombre reste assez petit sur la plupart des machines. Etudier ces algorithmes, de ce point de vue, par simulation demanderait une simulation très fine du réseau d'échanges des messages qui est très difficile à réaliser et très coûteux. C'est pourquoi, il nous semble intéressant de faire ces études par expérimentation directe sur une machine réelle.

4.2 Le support des expérimentations

4.2.1 Le matériel : l'*iPSC* d'*Intel*

Les expériences ont été realisées sur l'*iPSC* installé à l'*IRISA* dans le cadre du *PRC C*^{3}. C'est une machine constituée de processeurs puissants et indépendants ne communiquant entre eux que par échanges de messages, reliés suivant une topologie de type hypercube.

Un cube de dimension d est un graphe $H_d = (V_d, E_d)$ où l'ensemble V_d des sommets est défini par $V_d = \{0,1\}^d$ et l'ensemble E_d des arcs est tel que :

$$E_d = \{(x,y) | x \in V_d,\ y \in V_d, x \textit{ et } y \textit{ diffèrent exactement d'un bit}\}$$

L'hypercube a 2^d nœuds, $d*2^{d-1}$ arcs et un diamètre de d. De plus, chaque nœud a exactement d voisins.

Dans l'*iPSC*, les liaisons entre nœuds se font grâce à une liaison *Ethernet* à 20Mbit/sec. Les messages sont découpés en fragments de 1024 octets. Le système permet d'envoyer un message à un nœud qui n'est pas physiquement voisin de l'émetteur, en assurant le routage. Le temps de transfert d'un fragment d'un nœud à l'autre est de l'ordre de 5 ms, mais peut varier en fonction de la charge du réseau. Chaque processeur est doté d'une horloge. Celles-ci ne sont pas synchronisées (on a pu noter des dérives de six secondes par jour). D'autre part, le chargement des programmes sur les nœuds se fait à partir de l'hôte et on a des écarts importants sur l'instant de début d'exécution des programmes sur chaque site.

On est bien dans le cadre asynchrone :

- pas d'horloge globale;

- temps de transfert d'un message arbitraire mais fini (si l'on choisissait une borne à ce temps, il faudrait la prendre tellement grande que cela rendrait totalement inefficaces les algorithmes basés sur cette information).

4.2.2 L'environnement logiciel : *ECHIDNA*

Le logiciel de programmation standard de l'*iPSC* est le langage *C* et une bibliothèque de sous-programmes d'émission et de réception de messages. Nous avons préféré utiliser un environnement logiciel en cours de développement à l'*IRISA* : *ECHIDNA*. D'une part, il permet de faire exécuter sur l'*iPSC* des algorithmes écrits en *ESTELLE* [5] et, d'autre part, il fournit des outils de suivis de l'exécution. Nous y avons trouvé plusieurs avantages :

- le langage *ESTELLE* est plus adapté à l'écriture de ce type d'algorithme;
- on dispose à l'*IRISA* de simulateurs, acceptant en entrée un programme écrit dans un sous-ensemble d'*ESTELLE*. Ce qui permet de vérifier plus facilement la correction des programmes. On peut donc utiliser exactement le même programme pour faire la validation de la mise en œuvre par simulation et l'expérimentation sur la machine réelle;
- les programmes sont indépendants des primitives de base de l'*iPSC* et sont donc plus facilement transportables sur les versions ultérieures de la machine (*iPSC2*).

4.3 Les mesures effectuées

4.3.1 Les synchroniseurs étudiés

Nous avons étudié un synchroniseur assurant le synchronisme fort : le synchroniseur β de [2], et deux synchroniseurs assurant un synchronisme faible : les synchroniseurs α de [2] et δ de [12]. Les expériences ont été faites en utilisant un réseau synchrone ayant une topologie hypercubique identique à celle du réseau asynchrone servant de support. Il y a donc toujours un processus du réseau synchrone par processeur réel et tous les arcs du réseau synchrone correspondent à une liaison physique sur l'*iPSC*, cela afin de simplifier l'exploitation des résultats en évitant les intèractions avec le système de routage de l'*iPSC*.

Nous rappelons ci-dessous les complexités de ces synchroniseurs en nombre de messages (C) et en temps (T) (évaluée en considérant que la transmission d'un message prend un unité de temps), sur un cube de dimension d ayant $n = 2^d$ nœuds.

synchroniseurs	C	T
α	$O(nlog_2(n))$	$O(1)$
β	$O(n)$	$O(log_2(n))$
δ	$O(n)$	$O(1)$

4.3.2 Les résultats

Les mesures de la durée d'exécution d'une certaine suite d'actions sur un processeur ont été faites en accèdant à l'horloge locale de ce processeur, via une procédure système qui fournit une valeur avec une précision de 5 ms.

Le coût de la synchronisation

Nous avons essayé d'évaluer le coût de l'algorithme de synchronisation pour les différents synchroniseurs et pour différentes tailles de l'hypercube. Pour cela, nous avons mesuré la durée moyenne d'une pulsation sur un site pour un algorithme synchrone n'échangeant aucun message avec ses voisins et n'effectuant aucun calcul. Ce qui veut dire que seul le synchroniseur s'exécute. La moyenne est calculée sur 500 pulsations. La figure 3 montre que le synchroniseur

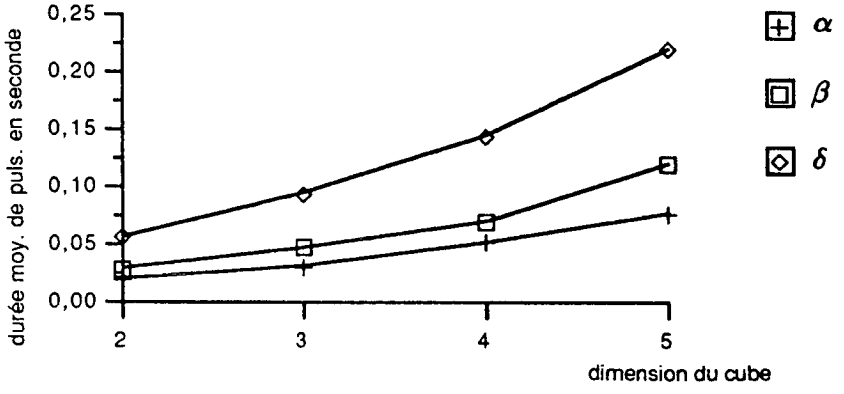

Figure 3 : Durée moyenne des pulsations n'échangeant pas de message

δ, bien qu'ayant une complexité en temps $O(1)$ comme α est en fait plus coûteux en temps que celui-ci. Cela s'explique par le fait que le nombre de processeurs (≤ 32) est relativement petit et que, si le temps de transfert d'un message est t, et que l'on considère que l'on a le maximum de parallèlisme, la durée de pulsation pour α est t, alors qu'elle est de $3t$ pour δ.

On constate également pour α et δ, que bien qu'ayant une complexité constante, la durée de la synchronisation d'une pulsation augmente avec la taille de l'hypercube. Cela est dû au fait que tous les transferts ne se font pas en parallèle, bien que cela soit théoriquement possible.

Nous avons également étudié la variation de la durée de pulsation sur un nœud. Les courbes suivantes (figures 4, 5 et 6) fournissent, pour chaque synchroniseur, la plage de durée correspondant à l'intervalle $[moyenne - \sigma, moyenne + \sigma]$ (σ représente l'écart type). β, assurant le synchronisme fort, fournit des pulsations dont la durée est bien concentrée. Alors que α et δ, assurant le synchronisme faible, fournissent des pulsations dont la durée peut varier de façon importante, cette variation augmentant avec la taille de l'hypercube. Cette variation est directement liée à l'écart maximum pouvant exister à un instant donné entre le numéro de pulsation sur deux sites quelconques. Pour un hypercube de dimension d, cette différence est de 1 pour β, de d pour α et β.

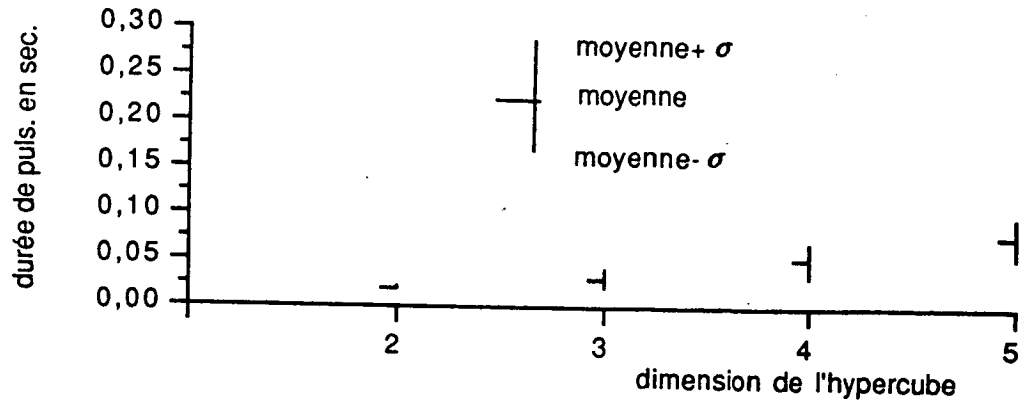

Figure 4 : Synchroniseur α

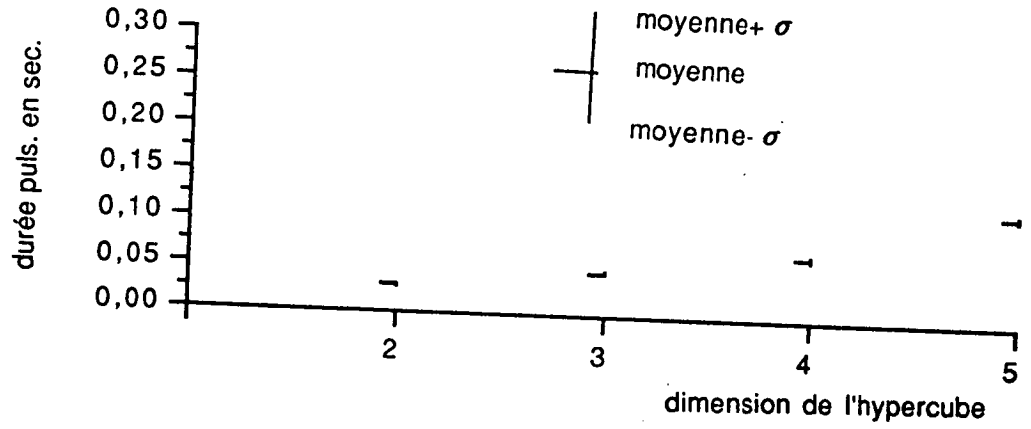

Figure 5 : Synchroniseur β

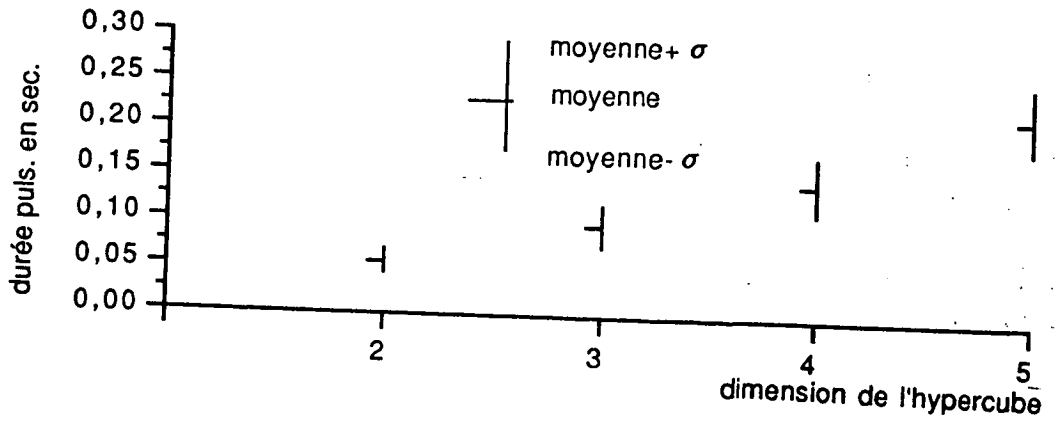

Figure 6 : Synchroniseur δ

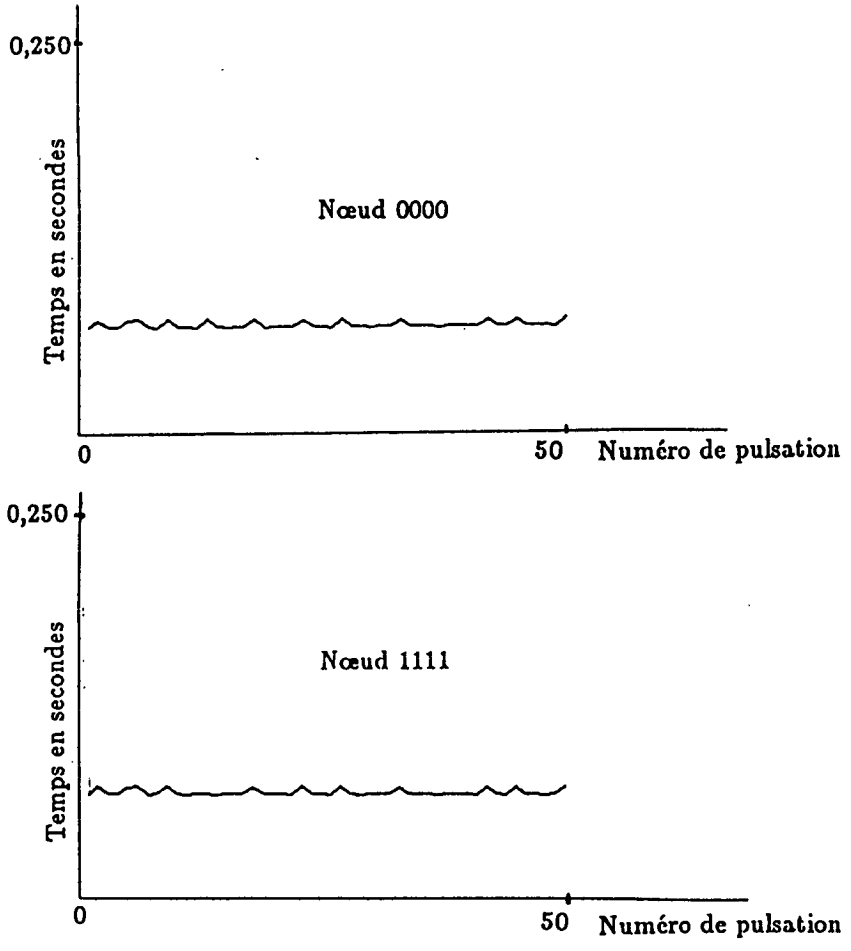

Figure 7 : Durée de pulsation avec le synchroniseur β

Si nous comparons la durée de la $i^{ème}$ pulsation sur deux sites différents, nous observons les résultats suivant pour β (figure 7) sur un hypercube de dimension 4. En abscisse figure le numéro de pulsation et en ordonnée, la durée de pulsation en secondes. Sur le schéma figurent les courbes pour les nœuds 0000 et 1111 qui sont physiquement sur l'hypercube à une distance 4 l'un de l'autre. On observe une forte similitude entre les deux comportements.

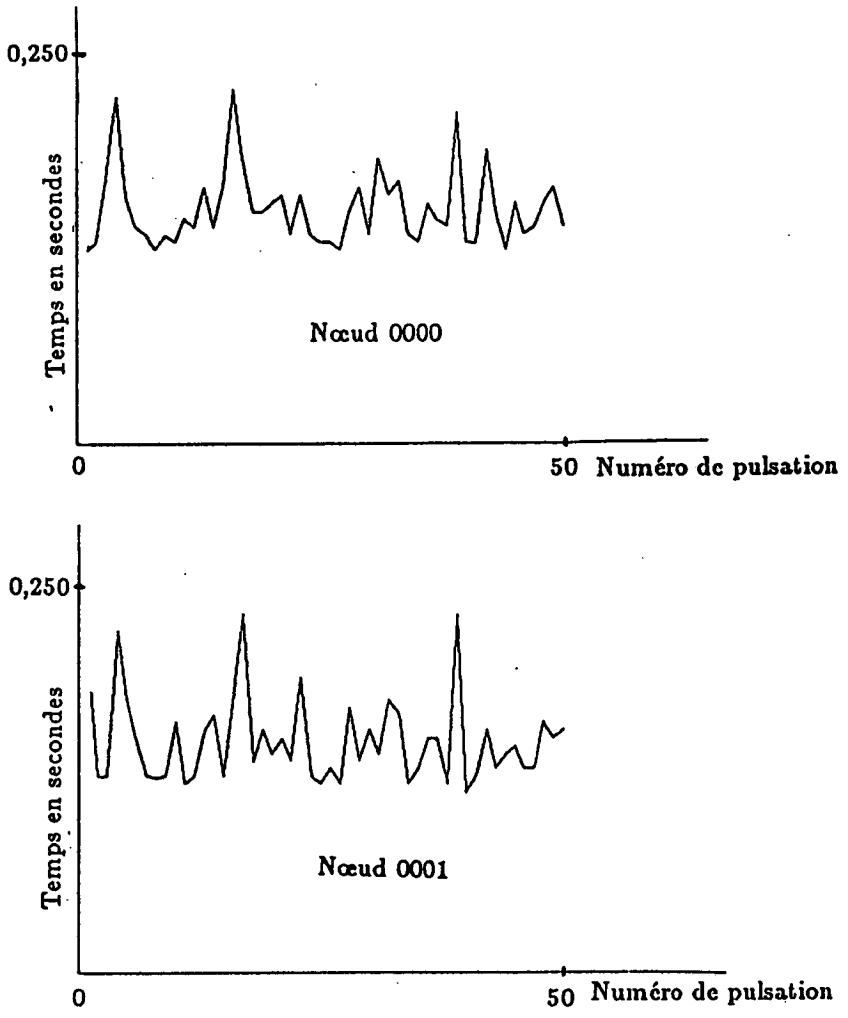

Figure 8 : Durée de pulsation avec le synchroniseur δ

La figure 8 fournit la même type d'information pour δ. On observe d'une part une grande variation sur un site et d'autre part des comportements moins liés entre deux sites qui ici sont voisins.

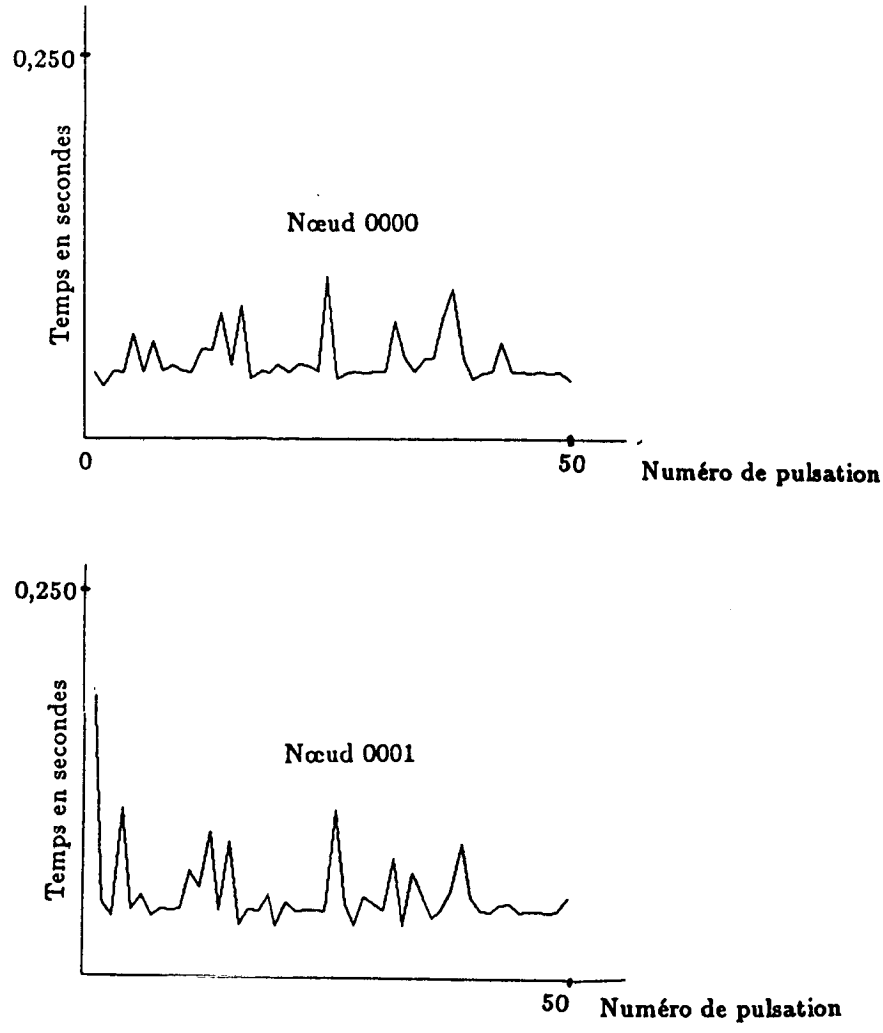

Figure 9 : Durée de pulsation avec le synchroniseur α

Avec α (figure 9), on observe des comportements encore moins liés entre eux.

Influence de l'algorithme synchrone

Nous voulions déterminer si les actions effectuées par l'algorithme synchrone pouvaient avoir un influence sur l'efficacité des synchroniseurs. En particulier, le traffic de messages augmentant, un synchroniseur coûteux en message, comme α, peut-il se trouver défavorisé par rapport à un synchroniseur échangeant moins de messages comme β.

Pour évaluer ce phénomène nous avons mesuré la durée moyenne de pulsation pour un algorithme échangeant un message avec tous ses voisins à chaque pulsation. Il faut noter qu'un message émis à la pulsation p contient des informations calculées à la pulsation $p-1$.

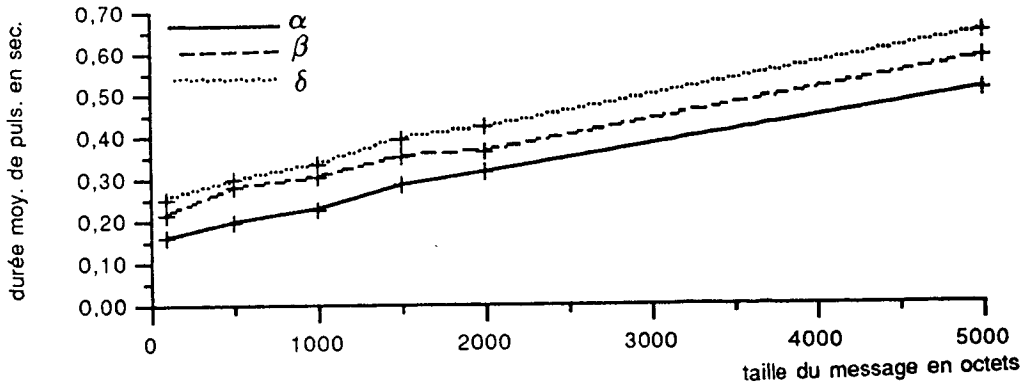

Figure 10 : Durée de pulsation avec un message

On peut donc considérer, sans perte de généralité, qu'un site émet au plus un message vers un site voisin lors de la pulsation. Les moyennes ont été calculées sur 500 pulsations.

La figure 10 fournit, pour chaque synchroniseur, la durée moyenne de pulsation en fonction de la taille des messages échangés (lors d'une exécution, tous les messages échangée ont la même taille). Les expériences ont été faites sur un hypercube de dimension 4. Il faut se rappeler que les messages sont émis physiquement par morceaux de taille inférieure à 1024 octets. ainsi, l'émission d'un message synchrone de 1000 octets correspond à un envoi physique, alors l'émission d'un message synchrone de 1500 octets correspond à deux messages physiques. Ceci explique l'aspect non régulier de la courbe.

On constate que la hiérarchie (en terme de temps d'exécution) des synchroniseurs n'est pas modifiée par l'augmentation de la taille des messages synchrones émis. Le temps d'exécution total d'un algorithme synchrone sur un réseau asynchrone peut être décomposé en deux parties, d'une part le temps passé effectivement à exécuter des actions pour le compte de l'algorithme synchrone et d'autre part, le temps d'exécution du synchroniseur. Pour un synchroniseur et un réseau donnés, ces temps peuvent être considérés comme indépendants.

La figure 11 montre la dispersion des durées de pulsations pour les différents synchroniseurs dans le cas d'une émission d'un message synchrone par pulsation à chaque

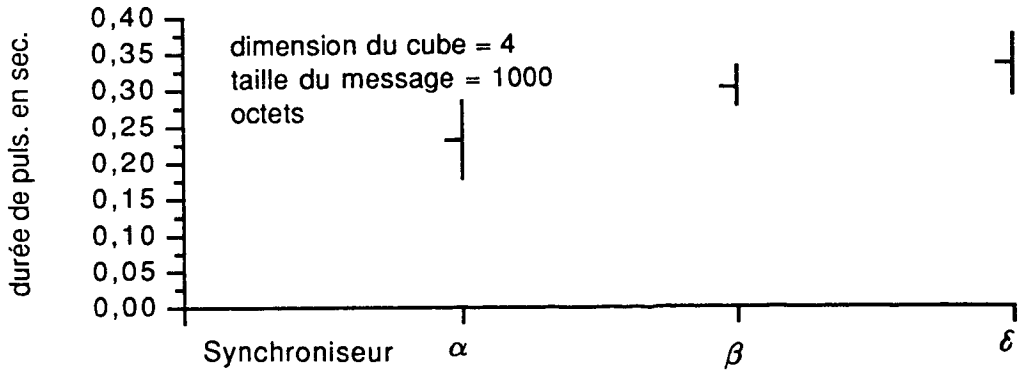

Figure 11 : Répartition de la durée de pulsation pour les différents synchroniseurs

voisin. L'émission des messages synchrones tend à augmenter la dispersion des durée de pulsation, en particulier pour β.

La complexité en communication (nombre de messages) ne semble pas être un critère pertinent pour comparer l'efficacité des synchroniseurs sur l'*iPSC*.

5 Conclusion

Pour prendre en compte la notion de temps dans les systèmes distribués, de nombreuses théories ont été développées. Les premières inspirées des travaux de Lamport, ont visé à construire un ordre partiel entre les événements de l'algorithme distribué. Plus récemment, une autre approche a visé à construire un temps global cohérent connus de tous les processeurs : L'interpréteur permettant de construire ce temps global est un synchroniseur. Les algorithmes distribués utilisant les synchroniseurs sont cadencés par l'horloge reconstruite et sont dits synchrones.

Nous avons mis en évidence deux formes de synchronismes : fort et faible. Le synchronisme fort respecte la simultanéité au contraire du synchronisme faible. Les propriétés de chacune de ces deux formes de synchronisme ont été établies.

Ensuite, nous avons examiné les problèmes résultant de la mise en œuvre des différents synchroniseurs. Les principaux mécanismes fondamentaux nécessaires ont été mis en évidence. Le choix des algorithmes de routage des informations de contrôle est déterminant pour l'efficacité du synchroniseur résultant.

Différentes mises en œuvre ont été testées sur une machine distribuée : l'*iPSC*. Les résultats obtenus rappellent que, si les évaluations de complexité théorique sont nécessaires, elles ne suffisent pas pour faire un choix de mise en œuvre dans un cadre réel et qu'elles doivent être complétées par des expérimentations. En particulier, le compromis entre le nombre de messages et le temps de calcul se révèle difficile à évaluer : la topologie du réseau, le nombre de processeurs, le nombre de messages de l'algorithme synchrone sont déterminants, mais leur influence respective est difficile à mesurer.

L'étude des algorithmes synchrones pouvant s'exécuter à l'aide de synchroniseurs n'est encore qu'à son début. Les algorithmes distribués synchrones connus sont encore peu nombreux. Une étude appropriée de leurs propriétés est encore à réaliser et semble devoir offrir des perspectives prometteuses.

Remerciement

Ces travaux ont été effectués en partie dans le cadre du *PRC* C^3. De plus, nous tenons à remercier d'une part Cl. Jard et JM. Jézéquel qui développent le logiciel *ECHIDNA* que nous avons utilisé et d'autre part, JM. Hélary pour ses remarques constructives sur les synchroniseurs.

Bibliographie

[1] M. Adam, PH. Ingels, and M. Raynal. The meaning of synchronous distributed algorithms run on asynchronous distributed systems. In *The Third International Symposium on Computer and Information Sciences, Izmir*, November 1988.

[2] B. Awerbuch. Complexity of network synchronization. *Journal of ACM*, 32(4):801–823, October 1985.

[3] J.C. Bermond, J.C. Konig, and M. Raynal. General and efficient decentralized consensus protocols. In *Proc. of* 2nd *Int. Workshop on Distributed Algorithms, Amsterdam*, July 1987. to appear : Springer Verlag LNCS 312 (1988).

[4] G. Berry, P. Couronne, and G. Gonthier. Programmation synchrone des systèmes réactifs : le langage Esterel. *Technique et Science Informatiques*, 6(4):305–316, 1987.

[5] J.P. Courtiat, P. Dembinski, R. Groz, and C. Jard. Estelle : un langage ISO pour les algorithmes distribués et les protocoles. *Technique et Science Informatique*, 6(2), 1987.

[6] J.-M. Helary and M. Raynal. *Synchronisation et contrôle des systèmes et des programmes répartis.* Eyrolles, Septembre 1988. 193 p.

[7] J. Ichbiah. *Reference Manual for the ADA Programming Language.* ANSI/MIL-STD 1815a, January 1983.

[8] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications. of the ACM*, 21(7):558–565, July 1978.

[9] L. Lamport. Using time instead of time-out for fault-tolerant distributed systems. *ACM Toplas*, 6(2):254–280, April 1984.

[10] Y. Moses, D. Dolev, and Halpern Y. Cheating husbands and other stories: a case of study of knowledge, action and communication. *Distributed Computing*, 1:167–176, 1986.

[11] G. Neiger and S. Toueg. Substituting for real time and common knowledge in distributed sytems. In *6th Annual ACM Symposium on Principles of Distributed Computing*, pages 281–293, August 1987.

[12] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. In *6th Annual ACM Symposium on Principles of Distributed Computing*, pages 77–85, August 1987.

[13] M. Raynal. *Systèmes répartis et réseaux : concepts, outils et algorithmes.* Eyrolles, Février 1987. (également the MIT Press, 1988).

[14] J. L. Welsh. Simulating synchronous processors. *Information and Computation*, 74:159–171, 1987.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique