LAPP-T-95/02

Université de Savoie
Laboratoire d'Annecy-Le-Vieux de Physique des Particules
BP 110, 74941 Annecy-Le-Vieux Cedex, France

THÈSE

présentée pour obtenir le titre de

Docteur en Sciences
Spécialité: Physique Experimentale & Instrumentation

par

Alain MASSEROT

Mise en œuvre et intégration dans l'expérience L3 d'un déclenchement de deuxième niveau avec assemblage de l'événement, développé autour d'un réseau de routeurs dynamiques C104 et de Transputers T9000.

Soutenue le 15 septembre 1995 devant la Commission d'Examen

L. Massonnet	(Président)
A. Degré	(Directeur de thèse)
R. Dobinson	(Rapporteur)
P. Le Dû	(Rapporteur)
J. J. Blaising	
D. Grabas	

Les hommes, au fond, ça n'a pas été fait pour s'engraisser à l'auge, mais ça été fait pour maigrir dans les chemins, traverser des arbres et des arbres, sans jamais rencontrer les mêmes; s'en aller dans sa curiosité, connaître. C'est ça connaître.

Jean Giono.

Remerciements

Je remercie vivement Monsieur L.Massonnet d'avoir accepté la présidence de ce jury et je suis reconnaissant à Messieurs R.Dobinson et P.Le Dû de l'intérêt qu'ils ont bien voulu porter à mon travail.

Je tiens à exprimer ma reconnaissance à Monsieur D.Grabas d'avoir accepté de participer au jury.

Monsieur A.Degré a dirigé cette thèse; je lui exprime ici ma reconnaissance. Je remercie Monsieur D.Linglin pour son accueil au sein du laboratoire.

Mes remerciements s'adressent aussi à tous les menbres de l'équipe engagée sur le projet. Ainsi je remercie Mesdames F.Chollet–Le Flour, G.Moynot–Daguin et Messieurs J.J.Blaising, X.Cai, J.C.Cruz, G.Perrot avec lesquels il est à la fois agréable et instructif de travailler.

Je souhaite remercier Monsieur B.Martin, R.Heeley et M.Zhu du *CERN* pour leurs collaborations.

Je remercie enfin toutes les personnes qui, au L.A.P.P, m'ont apporté leurs concours.

Résumé

La thèse décrit le nouveau système de déclenchement de niveau-2 développé pour assurer la prise de données de l'expérience L3 pendant la phase 2 du LEP. Ce système assure l'acquisition des données de type déclenchement à chaque croisement de faisceaux, puis l'assemblage des événements sélectionnés par les processeurs cablés de niveau-1, enfin l'identification et le rejet en ligne du bruit de fond identifié par des algorithmes codés en *FORTRAN*.

Elaboré autour d'un réseau de *Transputers T9000*, interconnectés par des routeurs dynamiques *C104*, il met en œuvre des composants prototypes conçus par INMOS/SGS THOMSON pour des systèmes informatiques à architecture "parallèle".

L'accent est mis sur une nouvelle technique d'assemblage de l'événement, sur sa mise en œuvre et son intégration dans L3 et sur les performances obtenues.

Mots clés

• *L3* • *Déclenchement* • *Assemblage d'événement* • *Transputer* • *Routeur Dynamique* • *T9000* • *C104*

Abstract

The thesis describes the new level-2 trigger system. It has been developped to fit the L3 requirements induced by the LEP phase 2 conditions. At each beam crossing, the system memorizes the trigger data, builds-up the events selected by the level-1 hardwired processors and finally rejects on–line the background identified by algorithms coded in *Fortran.*

Based on *T9000 Transputers* and on *C104* data driven crossbar switches, the system uses prototypes designed by INMOS/SGS THOMSON for parallel processing applications.

Emphasis is set on a new event building technic, on its integration in L3 and on performances.

Keywords

• *L3* • *Trigger* • *Event Building* • *Transputer* • *Crossbar Switch* • *T9000* • *C104*

Table des matières

Table des figures

Liste des tableaux

Introduction

L'expérience L3 est l'une des quatre grandes expériences qui étudient les intéractions électrons-positrons (e^+e^-) auprès du collisionneur LEP au CERN. Depuis juillet 1989, dans une première phase d'exploitation, le LEP a produit plusieurs millions d'événements Z^0 dont l'observation a permis un test très fin du modèle électrofaible de Glashow, Salam et Weinberg [refGSW]. A partir de 1995, dans une seconde phase d'exploitation, l'energie des faisceaux du LEP sera augmentée pour atteindre le seuil de production des paires de W et accumuler la statistique jusqu'à la fin de ce siècle.

La collaboration L3, qui assure le fonctionnement du détecteur et l'exploitation des données, regroupe plus de 500 physiciens provenant de près de 50 Universités Américaines (MIT, Princeton, Caltech ...), Europeennes (Aix la Chapelle, NIKHEF, Genève, Rome, ETH Zurich, ITEP Moscou ..) ou Asiatiques (Beijing, Bombay, Taiwan..).

Depuis plus de 12 ans, deux groupes Français, l'I.P.N de Lyon et le LAPP d'Annecy–le–vieux, sont fortement impliqués dans la construction, la prise des données et leur interprétation physique.

En particulier le LAPP est responsable du système de déclenchement de niveau 2, dont la fonction est de rejeter en ligne le maximum de bruit de fond ayant déclenché le premier niveau, mais sans introduire aucune inefficacité sur le signal physique. La décision est prise dans les quelques millisecondes suivant l'intéraction, après traitement des données par des algorithmes spécifiques à chaque source de bruit.

Le système actuel, developpé dans le début des années 80 autour du microprocesseur en tranche XOP, à structure parallèle, optimisé pour cette application, fournit d'excellentes performances, mais devenu obsolete il ne peut assurer le programme LEP phase 2.

Mon travail de thèse se situe dans le cadre du développement du nouveau système de déclenchement de niveau–2 pour l'expérience L3. Son objectif est double:

1. Tout d'abord remplacer le système actuel par un système développé autour de composants industriels et piloté par des langages de haut niveau Fortran ou C, afin d'assurer la prise de données de l'expérience L3 pour la durée du programme LEP phase 2.

2. Prendre en main les nouvelles technologies issues du domaine des télécommunications et du traitement parallèle pour les évaluer dans les techniques d'assemblage des événements et de déclenchement.

Pour répondre à la demande des expériences futures dans ce domaine (LHC ...), le CERN a engagé plusieurs programmes de *Recherche & Développement* dont l'échéance est à plusieurs années. Notre projet, plus modeste dans son cahier des charges, veut montrer que ces technologies sont dès à présent intégrables dans un déclenchement de niveau-2, et veut vérifier en situation reelle quelques une des performances espérées. L'intégration de ces technologies dans une expérience ne manque pas d'interesser les physiciens et ingénieurs qui simulent leur fonctionnement dans ces R et D.

Dans notre développement nous avons sélectionné la nouvelle technologie transputer développée par INMOS du groupe SGS Thomson, essentiellement le transputer T9000 et le routeur dynamique C104. Cette technologie optimisée pour assurer l'échange de données par communication point à point dans le cadre de traitement massivement parallèle, est également un bon candidat pour assurer l'assemblage de l'événement. Sa supériorité n'est pas tant dans les performances de ses composants que dans leur association qui s'effectue sans interface, permettant à l'utilisateur de personnaliser son architecture à la façon d'un jeu de construction *Lego*. L'association du routeur dynamique capable de sélectionner la destination du paquet en moins de $1\mu s$ en décodant l'en–tête du paquet, et du *Transputer T9000* avec ses 4 liens série, nous a permis de développer un système d'acquisition de données sur 43 canaux en parallèle, avec assemblage de l'événement et traitement des données, à la fois simple, souple et performant.

Ce système est clairement un prototype. Il a été réalisé à partir de composants prototypes souffrant de restrictions importantes. Toutefois les contraintes en temps imposées par le calendrier de l'expérience n'ont pas permis d'attendre l'arrivée de composants validés pour mettre en route le système. Les performances mesurées sont donc affectées par ces restrictions. Les retards successifs de livraison des composants ont énormément perturbé notre programme de travail. Le développement des logiciels a beaucoup souffert de ces retards, ainsi que de l'environnement de développement encore rudimentaire et instable fourni par le constructeur.

Installé dans L3 début Juillet 95, ce système assure la prise de données et le rejet *en ligne* avec une très bonne fiabilité depuis cette date. Ce système est le premier construit autour de la technologie T9000 et C104 installé sur le site du CERN dans une expérience en phase de production à assurer *en ligne* l'assemblage des données et le rejet des événements.

La présentation de ce travail commence par un survol du LEP et du détecteur L3, suivi d'une présentation succincte du système d'acquisition des données. L'implémentation matérielle puis logicielle du déclenchement de niveau-2 est ensuite décrite. les performances essentielles du système sont présentées. L'intégration du système sous le contrôle de l'acquisition de L3 est ensuite présentée. Pour alléger le texte, certaines parties très techniques ont été décrites dans les annexes dites *techniques* auxquelles le lecteur motivé ne manquera pas de se reporter.

Sans la collaboration efficace de Brian Martin (CERN division ECP-RA) et des étudiants travaillant dans le cadre *GPMIMD*, plus particulièrement Minghua Zhu et Roger Heeley, de INMOS et de la société Archipel, et sans le support financier de l'IN2P3, ce travail n'aurait pu aboutir.

Chapitre I

Présentation du contexte expérimental

I.1 Le *LEP*

Le *LEP* est un collisionneur de particules circulaire de 27 km de circonférence, implanté dans un tunnel de 3 mètres de diamètre situé 100 mètres sous terre à la frontière Franco-Suisse entre le Jura et l'aéroport de Genève. Un faisceau d'electrons et un faisceau de positrons circulent en sens opposé dans un même tube à vide sur des trajectoires voisines qui ne se rencontrent qu'en 4 points, chacun d'eux étant observé par l'une des 4 expériences appelées respectivement L3, ALEPH, OPAL et DELPHI (figure I.1).

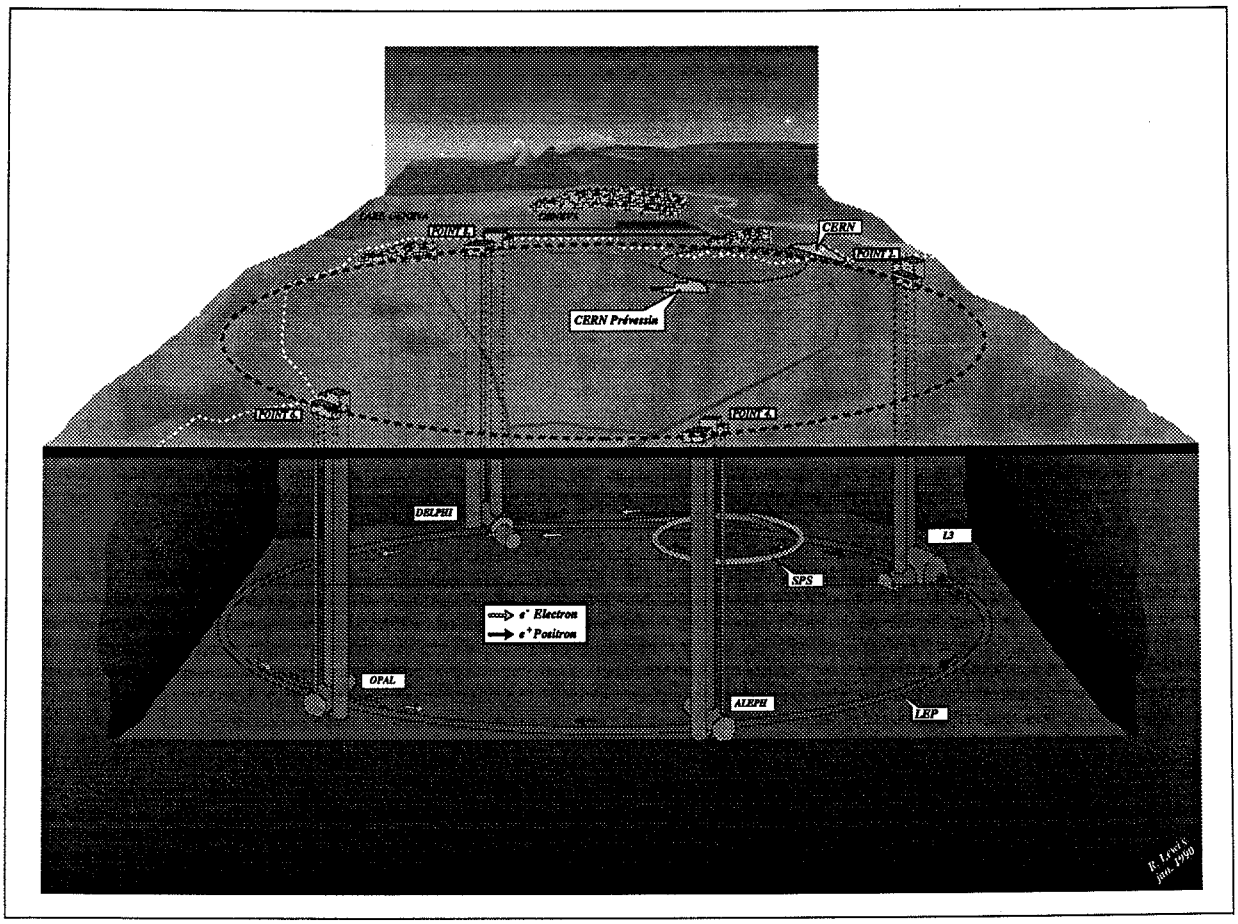


FIG. I.1 - *Le* LEP

Chaque faisceau est en fait constitué de 4 ou 8 paquets de particules également répartis sur sa trajectoire. Une fois les particules accélérées à l'energie voulue, les faisceaux circulent pendant une dizaine d'heures. Au point de collision les paquets ont une dimension de 200 μm horizontalement, 10 μm verticalement et $1cm$ longitudinalement. Ainsi 45000 croisements (90000 avec 8 paquets) sont observés chaque seconde par chacune des expériences, soit un croisement chaque 22 (ou 11) μs .

Lors du remplissage du *LEP*, chaque paquet véhicule un courant de l'ordre de quelques centaines de μA. L'intensité des faisceaux diminue lentement avec le temps, principalement à cause du rayonnement synchrotron. Le vide très poussé entretenu dans le tube (10^{-9} à $10^{-10} torr$) limite les collisions du faisceau avec le gaz résiduel à un niveau très bas. La trajectoire des faisceaux est controlée par une optique composée de plus de 3000 aimants dipolaires (déflection), et plus

de 2000 aimants quadrupolaires (focalisation).

En raison de la forme circulaire de la trajectoire, les électrons et les positrons perdent une partie de leur energie en émettant du rayonnement synchrotron. Deux sections de la trajectoire sont instrumentées de cavités accélératrices par radiofréquence (RF) qui compensent l'énergie perdue par rayonnement synchrotron et ainsi maintiennent constante l'énergie des faisceaux. La table I.1 donne quelques paramètres essentiels du LEP [31].

Circonférence	$26658.883m$
Rayon de courbure des dipôles	$3096.175m$
Section horizontale du faisceau	$200\mu m$
Section verticale du faisceau	$10\mu m$
Fréquence de révolution	$11245Hz$
Nombre de paquets par faisceau	$4/8$
Nombre de points de collision équipés	4
Nombre de cavités RF	128
Fréquence RF	$352.20904Hz$
Temps de révolution	$88.92446\mu s$
Puissance nominale de klystron	$16MW$
Gradient RF	$1.474MV/m$
Energie à l'injection	$20Gev$
Energie maximum	$\approx 60(100)Gev$
Luminosité ($3mA$ par faisceau)	$1.6 * 10^{31}cm^{-2}s^{-1}$

TAB. I.1 - *Principaux paramètres de* LEP

Le système d'injection des particules dans le LEP transite par plusieurs accélérateurs, il est schématisé sur la figure I.2. Les positrons, créés en bombardant une cible de Tungstène par des

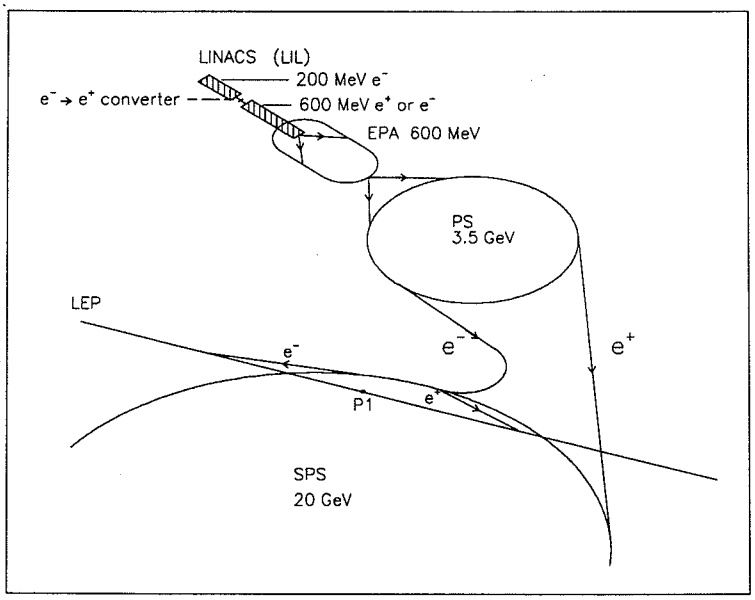


FIG. I.2 - *Le système d'injection du* LEP

electrons, sont focalisés et accélérés jusqu'à 600 MeV dans un accélérateur linéaire $LINAC$, puis stockés dans l'anneau d'accumulation EPA. Lorsque l'intensité est suffisante, les positrons sont transférés au Synchrotron à Protons PS puis accélérés jusqu'à 3,5 Gev. Ils sont ensuite injectés

dans le Super Synchrotron à Protons) *SPS* et accélérés jusqu'à 20 GeV, pour être enfin injectés dans le *LEP*. Les électrons suivent un itinéraire assez semblable hormis la phase de production.

Après injection dans le *LEP*, les faisceaux sont structurés en paquets et accélérés jusqu'à 45 Gev. Ensuite les trajectoires des faisceaux sont ajustées et les collimateurs de protection des détecteurs sont mis en place. Enfin les faisceaux sont mis en mode "collision". La prise de données peut commencer. L'ensemble de cette opération appelée *remplissage* nécessite quelques heures, généralement mises à profit par l'expérience pour vérifier la calibration des différents détecteurs.

Depuis sa mise en service en Juillet 89, le *LEP* n'a cessé d'améliorer ses performances, d'une part en améliorant la luminosité instantanée, d'autre part en doublant le nombre de paquets dans la machine depuis fin 92. Ainsi depuis le demarrage du *LEP*, la luminosité intégrée a pratiquement doublé chaque année.

Pour atteindre le seuil de production des paires de W autour de 200 GeV d'énergie dans le centre de masse, le *LEP* doit augmenter considérablement son energie, en ajoutant des cavités accélératrices supraconductrices. Afin d'accroître la luminosité, le *LEP* doit également modifier son mode de fonctionnement. A partir de 1995, il fonctionne avec 4 paquets équidistants, où chaque paquet est constitué par un train de 2, 3 ou 4 paquets individuels appelé *Bunch train*.

I.2 Le détecteur L3

I.2.1 Introduction

Les détecteurs implantés auprès des collisionneurs sont caractérisés par une structure en couches de detecteurs concentriques essayant de couvrir au mieux les 4π de l'angle solide. Les détecteurs de traces, les plus proches du point d'interaction, étant entourés par les calorimètres électromagnétique et hadronique. La mesure d'impulsion des traces chargées est déduite de la déflection induite par un champ magnétique. Le dessin final de chaque détecteur est un compromis entre les priorités de physique, le choix des techniques de détection mises en oeuvre et le coût. L3 a donné la priorité à la résolution des electrons, des muons et des photons, au détriment de l'identification des hadrons.

En allant du point d'intéraction vers l'extérieur du détecteur on rencontre:

- un détecteur de traces composé d'une chambre à expansion temporelle (*TEC, Time Expansion Chamber*) et d'un détecteur Silicium à microstrip (*SMD, Silicon Microvertex Detector*).
- un calorimètre electromagnétique composé de cristaux de Germanate de Bismuth (*BGO*).
- un cylindre de scintillateurs donnant une mesure en temps très précise.
- un calorimètre hadronique à absorption totale composé de sandwichs Uranium 238 appauvri et chambres proportionnelles.
- un ensemble de chambres à muons.
- les moniteurs de luminosité.
- enfin un aimant toroïdal qui baigne l'ensemble du detecteur dans un champ magnétique uniforme de 0,5 Tesla.

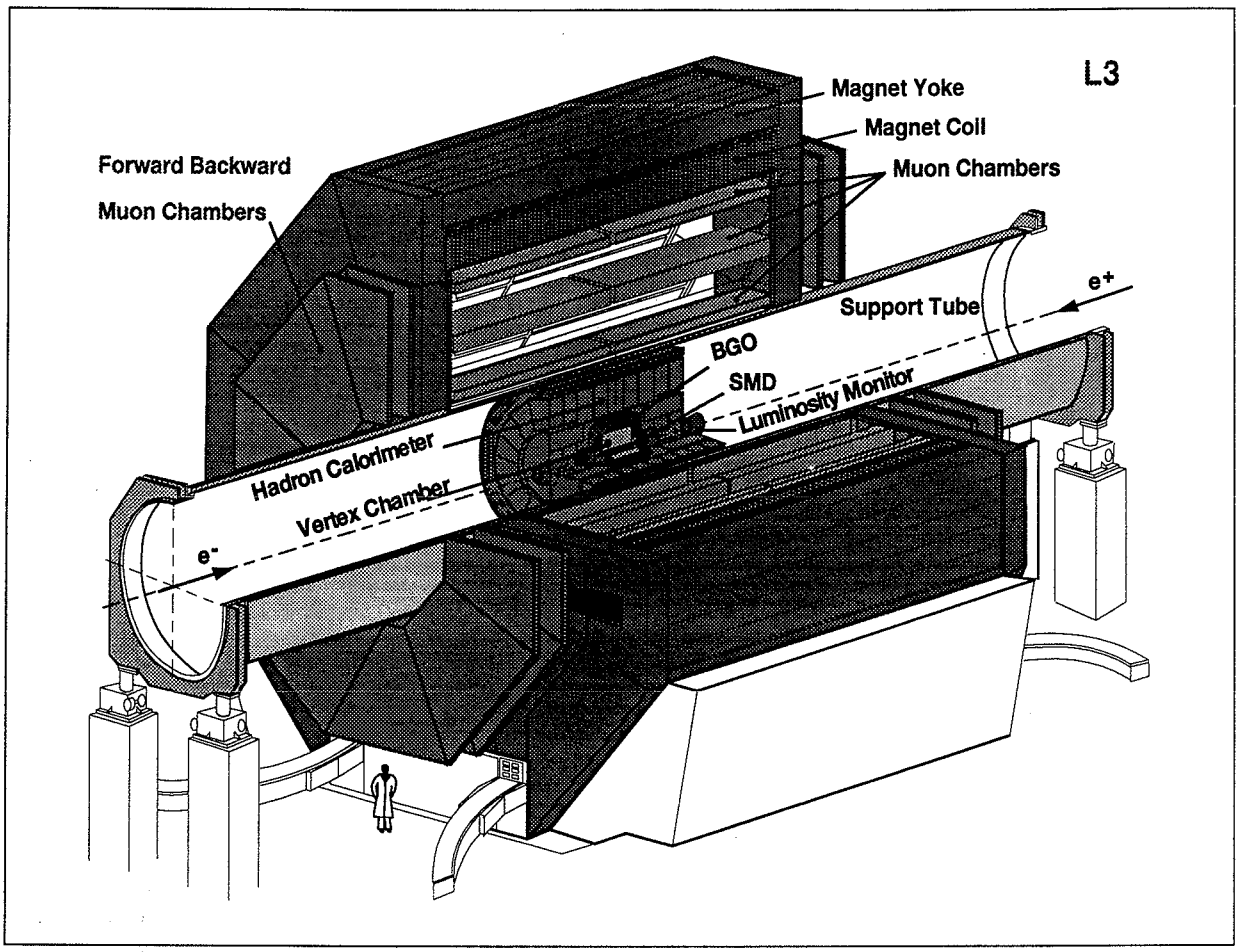


FIG. I.3 - *Schéma du détecteur L3*

L3 est la seule expérience *LEP* dont l'ensemble du détecteur baigne dans le champ magnétique. L'aimant de 12 mètres de diamètre sur 12 de longueur est de loin le plus gros des expériences *LEP* et donne un aspect gigantesque au détecteur (poids de fer équivalent à la tour Eiffel). Ce choix assure une excellente précision dans la mesure de l'impulsion des muons.

Un autre choix original de L3 est le calorimètre électromagnétique constitué par 12000 cristaux de Germanate de Bismuth BGO, qui permet de mesurer l'énergie des electrons et des photons avec une excellente résolution.

La figure I.3 est une vue schématique du détecteur montrant l'arrangement des differents sous–détecteurs. Nous allons maintenant présenter chacun des sous detecteurs en se résumant aux généralités. Pour une description plus détaillée consulter les références [29, 23].

I.2.2 La chambre *TEC*

C'est une chambre [37] à dérive très lente, $6\mu m\ ns^{-1}$, de type "expansion temporelle" qui mesure la trajectoire des traces dans le plan R–phi avec une résolution de 55 μm par fil (confère figure I.4). Cylindrique de 50 cm de rayon, longue de 126 cm, elle est constituée de deux régions concentriques appelées respectivement chambre intérieure *Innner TEC* et chambre extérieure *Outer TEC*. La chambre extérieure est subdivisée en 24 secteurs, chacun étant équipé de 54 fils d'anode parallèles à l'axe du faisceau. La chambre intérieure est subdivisée en 12 secteurs équipés de 8 fils chacun. Le point d'impact est déduit du temps de dérive mesuré sur chaque

fil. Ces temps de dérive sont numérisés par un convertisseur analogique–numérique *FADC, Fast Analog to Digital Converter*, piloté par une horloge à 100 MHz. Le mélange gazeux est composé de 80% de CO2 et de 20 % de butane (C4H10) sous une pression de 1.2 bar.

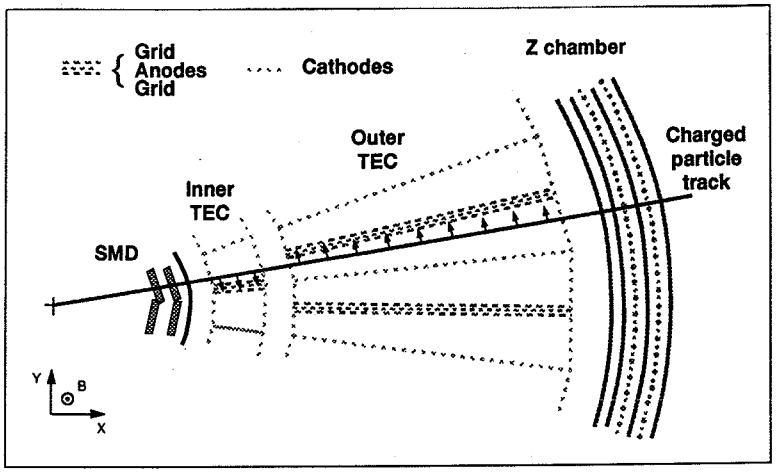


FIG. I.4 - *Le détecteur de trace*

Chaque trace, définie par une cinquantaine de points de la *TEC* et contrainte par la mesure du *SMD* à 6 μm, est reconstruite avec une résolution de 30 μm.

La chambre *TEC* externe est elle même entourée par la chambre appelée *TEC* RZ, qui est une chambre proportionnelle équipée d'une électronique de lecture avec division de charges fournissant les impacts dans le plan RZ avec une résolution de 110 μm avec la contrainte du *SMD*.

I.2.3 Le calorimètre électromagnétique

Entourant la chambre *TEC*, le calorimètre électromagnétique [30] est constitué par 11000 cristaux de BGO ($Bi_4Ge_3O_{12}$) répartis entre le tonneau (partie centrale) et les bouchons (extrémités près du faisceau) précédés de chambres à dérive *FTC, Forward Tracking Chambers* , figure I.5. Caractérisé par une densité élevée, une longueur de radiation de 1.1 cm, ce détecteur est optimisé pour mesurer l'energie d'electrons et de photons entre 100 MeV et 50 GeV. La structure porteuse en fibre de carbone a été dessinée pour optimiser la détection de particules. Le tonneau a un rayon intérieur de 52 cm, mesure 1m de long, et couvre la région angulaire supérieure à 42 degrés par rapport au faisceau. La résolution est de 1.4% à 45 GeV et inférieure à 2% à 3 GeV. Les bouchons couvrent la région angulaire entre 11 et 36 degrés.

I.2.4 Les scintillateurs

30 scintillateurs [26] disposés selon une structure en tonneau entre les calorimètres électromagnétique et hadronique, figure I.6, donnent une mesure de temps de vol très précise, utilisée essentiellement pour rejeter les muons d'origine cosmique.

I.2.5 Le calorimètre hadronique

L'énergie hadronique est mesurée par absorption totale des particules. Ce calorimètre [27, 25], figure I.6, est constitué de sandwiches de plaques d'Uranium 238 appauvri et de chambres à fil proportionnelles, structurés en tours. 9 couronnes de 16 tours chacune constituent le tonneau (*HB* qui couvre la région angulaire entre 35 et 145 degrés. Les bouchons (*HC*)descendent la couverture angulaire jusqu'à 6 degrés. Sa résolution en energie est donnée par $\Delta E/E = (55/\sqrt{E} + 5)\%$.

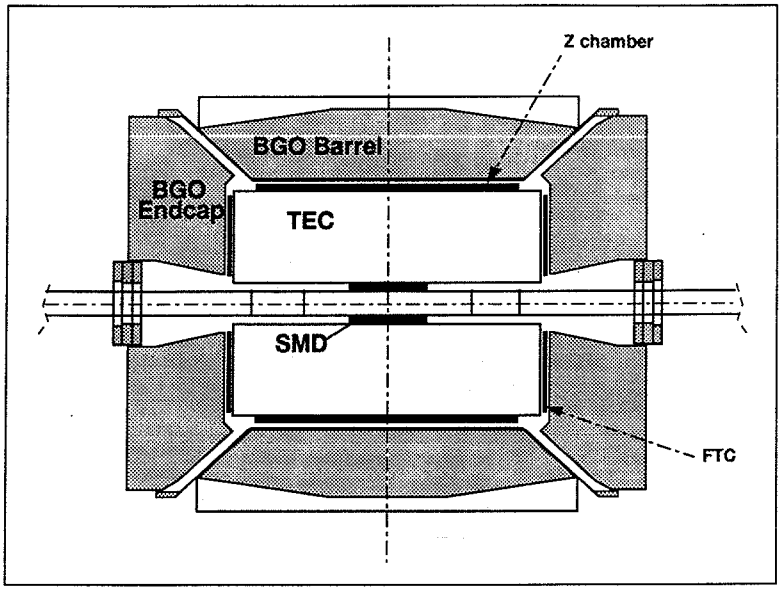


FIG. I.5 - *Le calorimètre électromagmétique*

Un filtre à muon rajoute encore une longueur d'absorption avant les chambres à muons. Il est nécessaire pour filtrer le bruit généré par l'Uranium.

I.2.6 Le détecteur de muons

Composé de 3 plans de chambres à dérive implantés [28] selon une structure octogonale il entoure les autres détecteurs. La figure I.7 schématise une coupe perpendiculaire au faisceau d'un octant. Les chambres les plus externes sont situées à 5,4 m du faisceau. Chaque plan est composé de chambres P qui mesurent les coordonnées dans le plan RPhi (donc l'impulsion du muon) avec une résolution de 250 μm par fil, et de chambres Z qui mesurent la position selon l'axe Z avec une résolution de 500 μm . Un muon émis dans la région centrale traverse 56 plans de dérive répartis dans les 3 plans de fil. La résolution obtenue sur la mesure de l'impulsion est de 2,5% à 45 GeV.

Afin d'augmenter la couverture angulaire des chambres à muons, les portes de l'aimant viennent d'être équipées de bobines de déflection, d'un système de déclenchement construit autour de *RPC (Resistives Plates Chambers)*, et d'un système de mesure des trajectoires à l'aide de 3 plans de chambres (1 à l'interieur de l'aimant et 2 à l'exterieur). Ce système a été installé dans L3 en 1994 et 95.

I.2.7 Le détecteur de luminosité

Composé de deux calorimètres électromagnétiques en BGO, il couvre la région comprise entre 1,43 et 2,65 degrés. Il mesure la luminosité vue par le détecteur avec une précision meilleure que 1%.

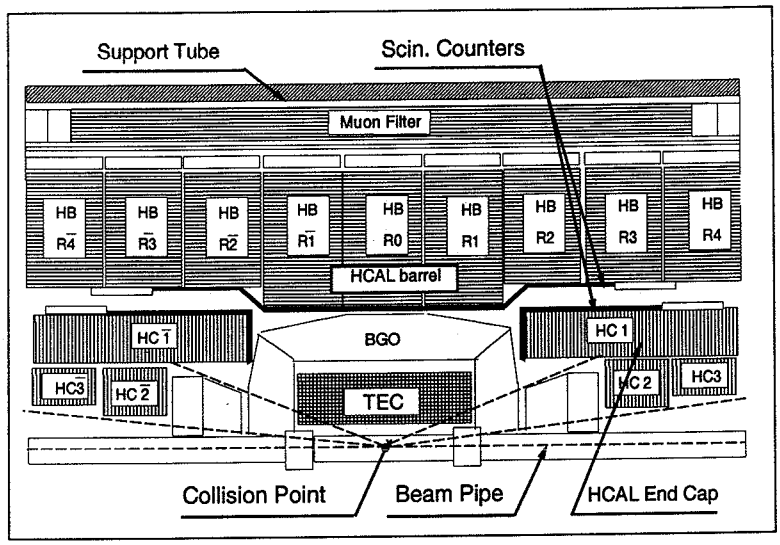


FIG. I.6 - *Coupe de la partie interne du détecteur L3*

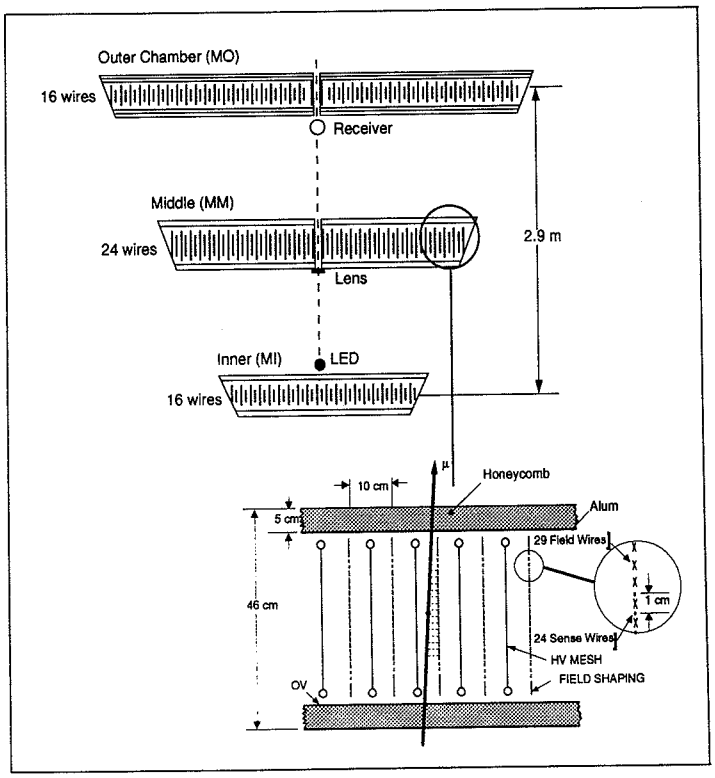


FIG. I.7 - *Schéma d'un octant du détecteur à muons*

I.3 Le système d'acquisition et de déclenchement

I.3.1 Le système d'acquisition

Chaque 11 ou 22 μs (respectivement en mode 8 ou 4 paquets) un croisement e^+e^- est produit par le *LEP*. A la luminosité nominale du *LEP* (10^{31} $cm^{-2}s^{-1}$) une information interessante pour la physique est produite toutes les 3 secondes. Par ailleurs la lecture des quelques dizaines de milliers de canaux d'électronique contenant l'information pour l'analyse, nécessite entre 1 et 3 ms, pendant ce temps, appelé temps mort, le système d'acquisition est inactif.

Si on lisait le détecteur systématiquement le système analyserait moins de 1 % des événements de physique intéressants produits par le *LEP*.

Si par contre on est capable de *signer* une collision interessante et d'activer la lecture des canaux d'électronique uniquement pour cet événement, on aura un système capable d'analyser plus de 99.5% des événements de physique intéressants produits par le *LEP*. Cette *signature* est donnée par le système de déclenchement.

En réalité, le système de déclenchement n'est généralement pas capable de séparer parfaitement le signal du bruit de fond, et l'utilisateur doit chercher le meilleur compromis entre accepter du bruit de fond et rejeter un peu de signal.

Dans les expériences *LEP*, en raison du faible taux de production de Z^0, la priorité est donnée à l'efficacité de détection du signal au détriment du pouvoir de réjection. Ce choix a pour conséquence que le taux de déclenchement effectif, quelques dizaines par seconde, est encore trop élevé pour le système d'écriture sur mémoire de masse. D'où la nécessité d'introduire des systèmes de sélection plus raffinés, capables de rejeter la majorité du bruit de fond résiduel pour limiter le taux d'écriture sur mémoire de masse à quelques événements par seconde. C'est le rôle des systèmes de déclenchement de niveau 2 et de niveau 3. Afin de pouvoir faire face à une situation expérimentale inattendue, il est indispensable que ces systèmes soient programmables. Il est aussi très important qu'ils n'introduisent aucun temps mort additionnel dans le système d'acquisition.

Le schéma de principe du système d'acquisition et de déclenchement est représenté sur la figure I.8. A chaque croisement le détecteur envoie l'information analogique d'une part vers une voie basse résolution mais rapide qui est utilisée par les déclenchements de niveau 1 et 2, d'autre part vers une voie haute résolution. La voie rapide est numérisée et lue en quelques microsecondes puis transmise aux processeurs de déclenchement de niveau–1 et de niveau–2. Le niveau–1 prend sa décision avant le prochain croisement.

Si la décision est négative la voie haute résolution n'est pas lue, et le détecteur se prépare à mémoriser les informations du prochain croisement. Si la décision est positive, la voie haute résolution est numérisée puis mémorisée. La lecture, non interruptible, de l'ensemble du détecteur engendre alors un temps mort de 1 à 5 ms.

Dans le même temps, le niveau-2, qui a déjà l'information en mémoire d'entrée, commence l'assemblage de l'information suivi du traitement et prend sa décision en quelques millisecondes. Cette décision est d'abord transmise aux processeurs de réduction de données de la *TEC* (DRP), leur permettant de limiter le traitement (10ms/event) aux événements sélectionnés par le niveau-2 et ainsi d'augmenter leur bande passante. La décision est ensuite transmise vers le système d'assemblage central. Celui–ci assemble uniquement les événements acceptés par le niveau–2 et remet à zéro les mémoires lorsque l'événement est rejeté.

Après assemblage de l'événement complet ($100 Koctets$), le niveau–3 traite l'information et prend une décision en quelques centaines de millisecondes. Il est constitué d'une ferme de 4 stations de travail VAX 4090. Les événements acceptés sont finalememt mémorisés sur cassette

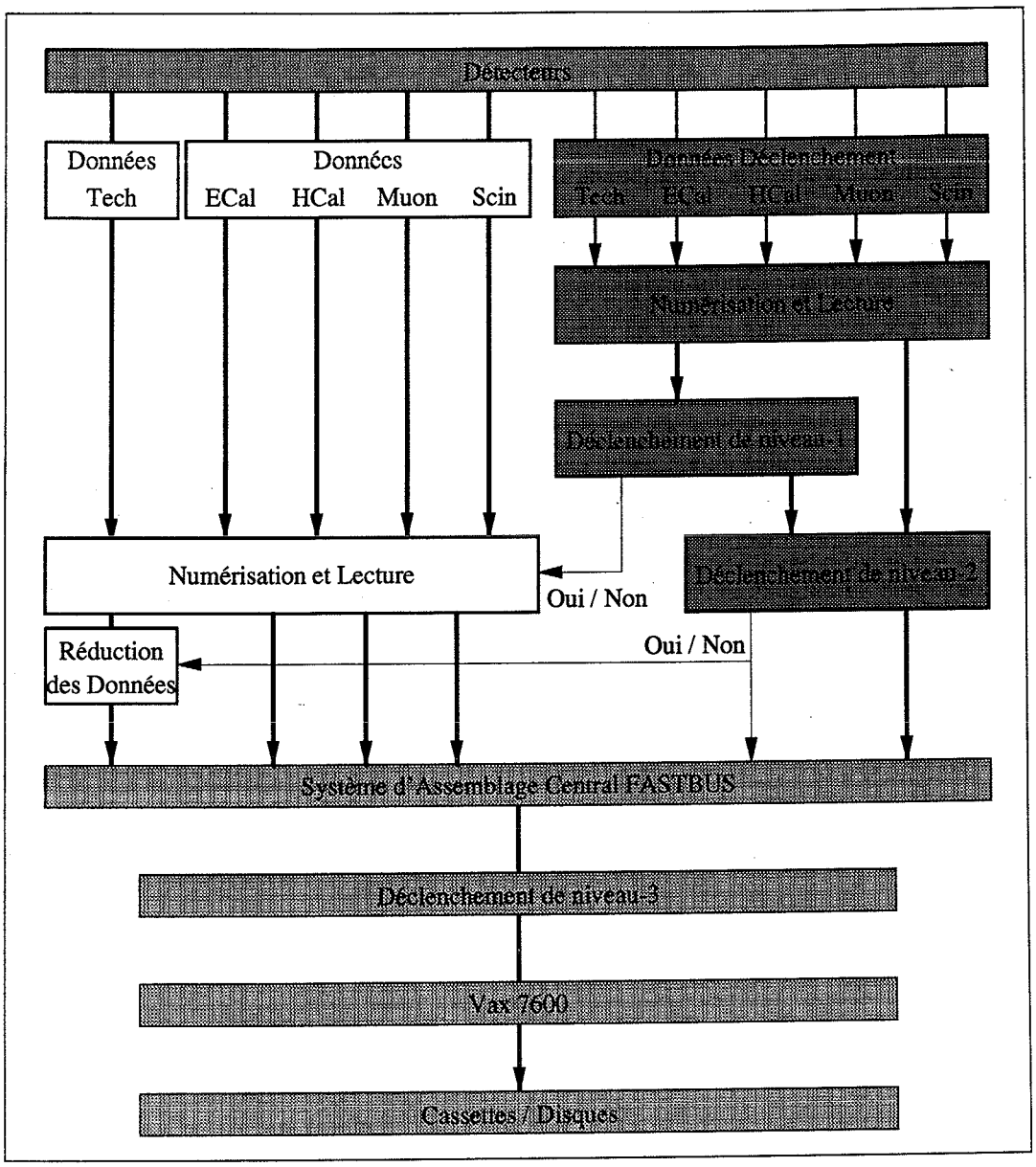


FIG. I.8 - *Structure du système d'acquisition de l'expérience L3*

par le VAX 7600 chargé de l'acquisition des données.

Le tableau I.2 donne les caractéristiques des trois systèmes de déclenchement.
Le fonctionnement du système d'acquisition et de déclenchement est décrit plus précisément dans [24, 5, 7].

Avant de décrire plus en détail le déclenchement de niveau-2, nous allons brièvement résumer les déclenchements de niveau-1.

Niveau de déclenchement	Type de processeur	Taux de déclenchement			Temps de décision
		entrée	analyse	accepté	
1	cablé	$45/90KHz$	$45/90KHz$	$\leq 100Hz$	$11/22\mu s$
2	programmable	$45/90KHz$	$\leq 100Hz$	$\leq 30Hz$	$\approx 1/10ms$
3	programmable	$\leq 30Hz$	$\leq 30Hz$	$\leq 10Hz$	qqs $100ms$
Les systèmes de déclenchement de niveau–2 et de niveau–3 ont été conçus pour ne pas introduire de temps mort dans la chaîne d'acquisition					

TAB. I.2 - *Les taux de déclenchement de l'expérience L3*

I.3.2 Le déclenchement de niveau–1

A chaque croisement, la décision de niveau 1 est effectuée par un "OU" logique de 5 systèmes cablés et indépendants, respectivement:

1. **le déclenchement en Energie** utilise l'information des calorimètres electromagnétique (512 canaux) et hadronique (384 canaux) pour signer les sous-catégories suivantes:

   ENERGIE TOTALE :
   - Energie totale >20 GeV
   - Energie totale dans les tonneaux (BGO et Hcal >15 GeV)
   - Energie totale électromagnétique (BGO >20 GeV)
   - Energie totale électromagnétique dans le tonneau BGO >1o GeV

   CLUSTER si l'énergie BGO+HCal dans une cellule en theta,phi est > 7 GeV

   HIT si au moins 2 cellules ont une énergie > 5 GeV.

   SINGLE PHOTON si l'energie d'un "cluster" BGO est > 80% du total BGO.

2. **le déclenchement multiplicité** si au moins 5 parmi les 30 scintillateurs sont touchés dans une fenêtre en temps de 30 ns.

3. **le déclenchement Muon** signe les événements dont au moins 1 particule traverse les chambres à muon avec une impulsion transverse supérieure à 1 GeV. Il est subdivisé en 3 catégories:

   Muon simple : 1 seul octant reconstruit 1 trace dans 2 des 3 chambres P et en coincidence avec 3 des 4 chambres Z

   Dimuon : au moins 2 octants touchés, la seconde trace étant située dans l'un des 5 octants opposés au premier.

   Muon à petit angle : demande une trace à l'avant et une trace à l'arriere dans la partie angulaire la plus large possible.

   Son taux est voisin de 10 Hz. En demandant la coincidence en temps avec au moins 1 scintillateur, il descend à 1 Hz ce qui est acceptable.

4. **le déclenchement** *TEC* identifie les traces chargées en procédant comme suit: d'abord il digitise le temps de dérive mesuré par chacun des fils touchés . A partir de la configuration des fils touchés il recherche les traces issues du point de collision. Après analyse des traces il prend sa décision. Il est utilisé comme déclenchement redondant pour les muons et l'energie, il signe également les événements 2 photons.

5. **le déclenchement luminosité** signe les electrons ou positrons déposant plus de 20 GeV dans les compteurs luminosité BGO et recemment Silicium implantés près du faisceau. Ils fournissent une mesure précise (0.2%) de la luminosité vue par L3.

Pour les raisons de temps mort déjà mentionnées le taux de comptage du niveau-1 doit rester voisin de quelques dizaines par seconde. Pendant la prise de données chaque système doit rester inférieur à un taux maximal qui lui est alloué.

I.3.3 Le déclenchement de niveau–2

Les fonctions

Il a 4 fonctions essentielles:

1. le stockage, dans une mémoire multi–port et multi–événement partagée avec le niveau-1, de l'information déclenchement émise à chaque croisement par les différents sous détecteurs. Seuls les croisements validés par un déclenchement de Niveau-1 sont gardés pour traitement par un processeur de niveau-2.
2. l'assemblage de l'information en un bloc *déclenchement* au format prédéfini.
3. le traitement des données à l'aide d'algorithmes spécialisés capables d'identifier le bruit de fond. Puis le rejet en ligne du bruit de fond.
4. L'ensemble de l'information est enfin transmis dans une mémoire multi–événement partagée avec le système d'assemblage central.

Le matériel

Le système actuel, assure la prise de données depuis le début du *LEP* en Juillet 1989. En raison des incertitudes sur le bruit de fond du *LEP*, (taux de déclenchement induit par le *LEP* et surtout seuil très bas du déclenchement en Energie) un cahier des charges plus sévère lui était imposé. En particulier, le taux de déclenchements de niveau-1 était estimé à 100 Hz avec un maximum à 500 Hz, et le temps de traitement des algorithmes arbitrairement estimé à 10 ms. Ce cahier des charges très sévère a été satisfait par un développement spécialisé, en technologie ECL autour du microprocesseur en tranches XOP [1], intégré dans le standard Fastbus. Ce système très performant a été décrit dans [4, 8].

La mémoire d'entrée profonde de 8 événements, double port ECL/Fastbus à accès simultané vrai, assure la mémorisation des données en parallèle sur 48 ports indépendants, à vitesse variable jusqu'à 60 ns/motde 16 bits.
L'assemblage de l'événement est assuré à travers une interface XOP/Fastbus qui transfere les données en mode pipeline à 125 ns avec un temps d'initialisation logiciel de $1\mu s$. Les 76 blocs de l'événement sont assemblés en moins de $400\mu s$ (2500 mots de 16 bits), il est environ 30 fois plus rapide qu'un CHI *Cern Host Interface* [34].
Le processeur à structure parallèle XOP assure le traitement d'un événement en 2 ms il est environ 40 fois plus rapide qu'un 68000.
La mémoire de sortie à double accès Fastbus est profonde de 64 événements.

La sélection des événements

La séparation signal/bruit de fond est obtenue en déroulant des algorithmes spécialisés sélectionnés en fonction de la signature du déclenchement de niveau-1. Les principales sources de bruit capables de déclencher le niveau-1 sont:

- des canaux "chauds" qui déclenchent le photon unique (*single photon*).
- du bruit corrélé capté par le calorimètre électromagnétique.

1. eXanpdable On–line Processor

- des dépots d'énergie induits par l'activité des plaques d'Uranium du calorimètre hadronique.
- du bruit rayonné capté par la *TEC*.
- des intéractions faisceau–gaz résiduel, faisceau–tube.
- ou encore du rayonnement d'origine cosmique.

Le niveau-2 réduit le bruit de fond résiduel en introduisant des critères de sélection plus raffinés que le niveau-1 [2]:

1. **Dans le déclenchement en ENERGIE:**
   - Le bruit de fond corrélé est réduit en applicant un seuil en energie variable d'un événement au suivant. Le seuil est fixé à une valeur telle que seuls 2% du total des canaux sont autorisés à le dépasser.Cette contrainte réduit drastiquement la contribution du bruit corrélé.
   - Le bruit de fond induit par l'Uranium est réduit en ne considérant dans le calorimètre hadronique que les dépots d'energie avec corrélation géométrique entre HcalA et HcalB ou entre (HcalA ou B) et BGO. Le bruit de fond de l'Uranium, aléatoire dans le temps et dans l'espace, est donc exclus de l'estimation, sauf coincidence fortuite.
2. **Dans le déclenchement MUON:**
   - Le déclenchement niveau-1 est essentiellement activé par les cosmiques. Son taux est limité à 1 Hz en demandant une coincidence avec au moins 1 scintillateur touché. Le niveau-2 réduit encore les cosmiques en demandant en plus une coincidence spatiale. Au moins un scintillateur touché doit se trouver dans l'un des octants muons "touchés". Toutefois pour prévenir le risque d'inefficacité qui serait induit par une dérive de la coincidence en temps, cette condition requiert également que aucune trace chargée ne soit reconstruite par la *TEC*. Ainsi un mauvais fonctionnement de la *TEC* ou des scintillateurs n'introduit pas d'inefficacité sur les muons . Par contre le rejet n'est pas maximisé.
3. **Dans le déclenchement trace chargée:**
   - Le bruit émis sous forme de rayonnement et capté par la *TEC* est caractérisé par l'arrivée simultanée de signaux sur un grand nombre de canaux. Un histogramme du temps d'arrivée des signaux signe très bien ce bruit. Le seuil de rejet est fixé à 65 canaux touchés dans un même interval de 200 ns.
   - Les déclenchements "faisceau-gaz" sont caractérisés par des traces de faible impulsion piégées par le champ magnétique torroidal, décrivant des spirales dans le détecteur. Ces événements présentent de nombreux secteurs touchés, mais on ne peut reconstruire de trace dans aucun. C'est ce critère qui permet de les signer.
   - La plus petite distance d'approche entre une trace reconstruite et le point de croisement des faisceaux permet de signer les interactions qui ne proviennent pas des interactions faisceau–faisceau (par exemple faisceau-gaz ou cosmique).

I.3.4 Le système d'assemblage central et le déclenchement de niveau–3

Le système d'assemblage central a pour fonction d'assembler l'information transmises par les différents sous–détecteurs après avoir vérifié leur appartenance à un même croisement. Le système d'assemblage est construit dans le standard *FASTBUS*. L'assemblage est effectué uniquement pour les événements acceptés par le système de déclenchement de niveau–2. L'événement

ainsi constitué est transmis au système de déclenchement de niveau–3 pour la sélection finale.

Le système de déclenchement de niveau–3 est constitué par une ferme de 4 stations de travail *VS4090* interfacées au système d'assemblage central par une carte *FT800*, qui est une interface *FASTBUS/SCSI*[2] contrôlé par un transputer T800.

2. Small Computer System Interface

Chapitre II

Description matérielle du système de déclenchement de niveau–2

II.1 Contraintes expérimentales

Les contraintes expérimentales imposées au système peuvent être classées en 2 catégories:

1. les contraintes concernant les performances requises par le cahier des charges de l'expérience pour assurer le programme *LEP phase 2*,
2. les contraintes imposées par l'environnement dans lequel il doit s'intégrer.

II.1.1 Les contraintes concernant les performances

- La mémoire d'entrée doit acquérir l'information de déclenchement à chaque croisement suivant la fréquence d'intéraction imposée par le fonctionnement du *LEP* 45 ou 90 KHz.
- La mémorisation des croisements sélectionnés par le déclenchement de niveau–1 ne doit introduire aucun temps mort additionnel dans la chaîne d'acquisition.
- Le système de traitement doit supporter un codage des algorithmes en *FORTRAN* afin d'assurer l'adaptabilité nécessaire à des conditions expérimentales nouvelles ou imprévues.
- Le système de traitement doit être extensible. La mise en œuvre de nouveaux algorithmes peut exiger une puissance de calcul supérieure. Une structure en *ferme* permet cet accroissement en augmentant le nombre d'éléments de la ferme.
- Afin de ne pas introduire de temps latence dans le traitement de l'information *TEC* par les processeurs de réduction de données, le temps moyen de traitement des événements avec les algorithmes utilisés dans *LEP phase 1* ne doit pas excéder 10 ms.
- Pour des considérations de temps mort acceptable par l'expérience, la fréquence maximale de déclenchement de niveau–1 ne peut pas dépasser $100Hz$.

II.1.2 Les contraintes imposées par l'environnement de L3

- La taille moyenne de l'information de déclenchement délivrée à chaque croissement est inférieure à $5Koctets$. Le temps disponible pour stocker l'information varie entre $2\mu s$ et $10\mu s$ selon le sous–détecteur. L'information totale associée à un événement est ainsi répartie en 43 ports indépendants. Pour les 43 ports de la mémoire d'entrée, la bande passante nominale du système de collection des données est de 1.4 $Goctets\ s^{-1}$.
- Les données émises par les différents déclenchements de niveau-1, codées sur 16 bits, sont transmises par des cables 25 paires de 60 mètres de longueur selon le protocole ECL differentiel (*Data–Strobe*), à la vitesse maximale de $60ns$ par mot.
- Le temps mort minimum associé à chaque déclenchement de niveau–1 est de $500\mu s$. C'est le temps nécessaire pour numériser l'information *haute résolution* sur chaque sous–détecteur. Nous avons retenu cette valeur comme limite acceptable pour permettre au déclenchement de niveau–2 de mémoriser l'information stockée dans les mémoires d'entrées.
- Les algorithmes de sélection utilisent les données relatives à un événement selon une structure prédéfinie gérée par des pointeurs. Cette structure est constituée de 76 blocs répartis sur les 43 ports d'entrées.
- Le système de déclenchement de niveau–2 doit fournir au système d'assemblage central les événements dans l'ordre de déclenchement de niveau–1.

- Le système d'acquisition de données de L3 est mis en œuvre dans le standard FASTBUS, ce qui impose le double accès *Transputer–FASTBUS* à la mémoire de sortie. La bande passante de l'interface doit au moins être égale à 500 $Koctets\ s^{-1}$.

- L'acquisition des données est synchronisée avec le *LEP*, par contre le traitement et la transmission vers le système d'assemblage central sont asynchrones.

- Il est nécessaire de concevoir un environnement de test permettant de simuler les conditions expérimentales de prise de données.

D'après les contraintes expérimentales, sur les quatres fonctions du système de déclenchement, trois sont plus facilement satisfaites:

- Le stockage des informations *déclenchement* à chaque croissement est assuré par une mémoire *First In First Out* associée à chaque port d'entrée. La mémorisation des événements validés par le système de déclenchement de niveau–1 est assurée par un processeur capable de tranférer l'information des ports d'entrée dans une mémoire multi–événements en moins de $500\mu s$.

- Les contraintes, pour le traitement, ne sont pas particulièrement sévères pour un processeur de la dernière génération. La ferme de traitement est constituée de processeurs *RISC*[1].

- La mémoire de sortie doit être accessible par les processeurs de traitement et *FASTBUS*. La bande passante de 500 $Koctets\ s^{-1}$ minimum ne pose pas de difficultés majeures.

La fonction la plus délicate à satisfaire concerne l'assemblage des événements. Cette fonction se complique lorsqu'elle est assortie de la gestion d'une ferme de processeurs de traitement qu'il convient d'alimenter.
La suite de ce travail porte essentiellement sur cette fonction et sur sa mise en œuvre à partir de la technologie *Transputer*.

II.2 L'assemblage d'événement

L'assemblage des événements consiste à prendre des données réparties dans des mémoires *sources* distribuées et à les assembler via un réseau d'interconnexion dans une mémoire *destination* pour constituer un seul bloc de données (figure II.1).

II.2.1 Les réseaux d'interconnexion

Il exite différents réseaux d'interconnexion entre les *sources* et les *destinations* [1]:

Les bus à temps partagé (figure II.2(a)): Le même bus permet l'interconnexion entre un ensemble de mémoires *sources* et les processeurs *destinations*. Chaque processeur *destination* possède une mémoire dans laquelle est assemblé l'événement. L'assemblage est entièrement contrôlé par les processeurs *destinations*. Un processeur *destination* demande l'accès au bus, via un système d'arbitration, puis lit *séquentiellement* les mémoires *sources* avant de relacher le bus pour le prochain processeur prêt.

La bande passante limite est donnée par la bande passante du bus, les temps d'arbitrage et d'initialisation des processeurs. L'initialisation est généralement la plus pénalisante.

1. Reduce Instruction Set Computer: processeur à jeu d'instructions réduit

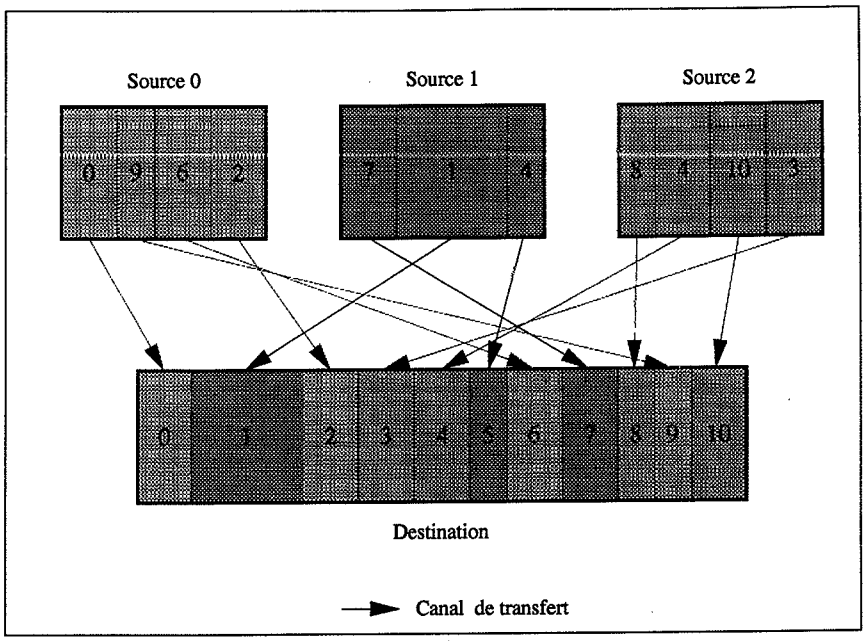


FIG. II.1 - *Principe d'assemblage*

Les mémoires multi–port (figure II.2(b)): Dans le cas particulier des mémoires *dual–port*, on utilise une mémoire double accès comme moyen d'interconnexion entre une *source* et un processeur *destination*. Les fragments d'un événement sont transmis en *parallèle* par les détecteurs dans les mémoires de la ligne sélectionnée. Ces fragments sont lus *séquentiellement* par le processeur associé à la ligne pendant que le prochain événement est transmis dans les mémoires d'une autre ligne.

La limitation est la taille de la matrice d'interconnexion qui croît comme $Nombre_{source} * Nombre_{destination}$. La bande passante totale est limitée par la bande passante du bus des processeurs et le temps d'initialisation de l'assemblage.

Les matrices d'interconnexion (figure II.2(c)): Chaque *source* et *destination* possède un point d'accès à la matrice d'interconnexion. Une matrice d'interconnexion est mise en œuvre par des circuits de routage. L'intéret de ces circuits de routage réside dans la facilité à interconnecter un grand nombre de *sources* et de *destinations* ainsi que dans la facilité à en étendre le nombre.

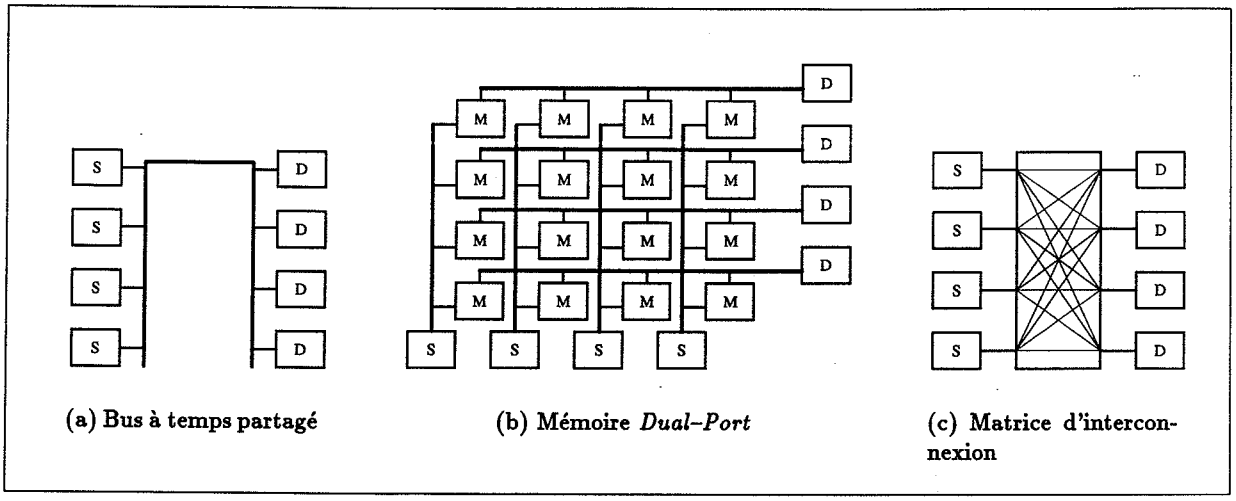
Pour notre application, nous avons choisi comme réseau d'interconnexion les circuits de routages.

II.2.2 Les circuits de routage

Les circuits de routage sont développés pour les systèmes parallèles multi–processeurs et pour les télécommunications. Ils sont optimisés pour l'échange de données dans le cas d'un trafic *aléatoire bi–directionnel*. Tandis que l'assemblage de données correspond à un trafic *déterministe uni–directionnel*.

Un circuit de routage est caractérisé par:

- le mode transmission de l'information: un message peut être transmis
  - en une seule fois,
  - divisé en *paquets* ou *cellules* de taille $\in \{0 \ldots max\}$ octets.
- le mode de routage: il détermine le moyen de progression du message (ou paquet) à travers le réseau.



(a) Bus à temps partagé (b) Mémoire *Dual–Port* (c) Matrice d'interconnexion

FIG. II.2 - *Différents réseaux d'interconnexion*

- l'algorithme de routage: c'est le moyen de sélection du canal de sortie.
- la présence ou la non–présence d'un contrôle de flux:
  - la présence d'un contrôle de flux assure qu'il n'y a pas de pertes d'information lors de la transmission d'un message (ou paquet) entre deux *nœuds*[2]. Cependant le contrôle de flux détériore la bande passante si la distance physique entre deux nœuds est supérieure à une valeur limite.
  - la non–présence d'un contrôle de flux autorise la perte d'information et ne limite pas la distance physique entre deux nœuds. La gestion de la perte d'information doit alors s'effectuer entre la source et la destination, la probabilité de perte d'information étant liée à la nature du trafic.

Modes de routage

Les performances des communications dépendent du mode de routage et sont notamment estimées par:

- le temps de latence défini comme le temps nécessaire au circuit de routage pour déterminer le lien de sortie (fonction de l'algorithme de routage).
- la bande passante définie comme le nombre d'octets transmis durant le temps pris par la communication par rapport à la bande passante disponible sur l'ensemble des liens utilisés lors de cette communication.

Il exite différents modes de routage [12] et [3]:

- **Mode** *Store–and–Forward*: Dans ce mode de routage, un message progresse dans un réseau en étant entiérement stocké sur l'ensemble des nœuds qu'il traverse. La réémission par un nœud intermédiaire ne peut se faire que si le message a été entièrement reçu. L'inconvénient de ce type de routage est un temps de latence important, temps stockage et décodage sur chaque nœud intermédiaire, avec une faible bande passante puisqu'un seul lien est utilisé à la fois.

2. On appelle nœud: une source, un circuit de routage ou une destination

- **Mode** *Cut–Through*: Dans ce mode de routage, lorsqu'un nœud intermédiaire reçoit un message, il peut commencer à le réémettre sans attendre la réception complète des données. On peut réaliser un tel routage de différentes façons. Il existe un modèle dit par circuit et un modèle *wormhole* :

  Modèle par circuit : La source envoie d'abord un message contenant l'adresse de destination. Il permet de réaliser les connexions sur chaque nœud de routage intermédiaire, entre le port d'entrée et le port de sortie. Une fois les connexions établies, la destination renvoie un accusé de réception. Dès que la source reçoit cet accusé, elle envoie le message sur le circuit. Le temps de latence reste important, temps d'établissement du circuit, mais la bande passante est pleinement utilisée puisque l'on utilise l'ensemble des liens.

  Modèle *wormhole* : La destination du message est placée en tête du messsage. Lorsque le message arrive sur un nœud intermédiaire, la sortie est calculée, et le message est routé. Le corps du message suit l'en–tête et ne peut se séparer de celui–ci. Avec ce type de routage, le temps de latence est minimal et la bande passante optimale.

La figure II.3 schématise le temps de communication d'un message entre deux processeurs à distance 3 selon le mode de routage. Le mode de routage *wormhole* offre les meilleures performances,

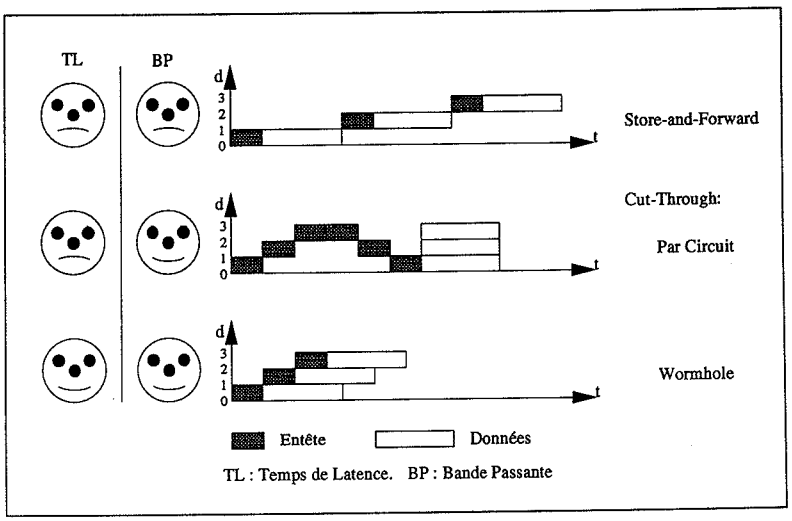


FIG. II.3 - *Différents modes de routage*

nous utilisons le mode de routage *wormhole* associé au mode de transmission par paquets.

La perte d'information n'est pas tolérable lors de l'assemblage d'événements. L'utilisation d'un circuit de routage sans contrôle de flux nécessiterait la mise en place de technique de *distribution de trafic* au niveau des *sources* pour diminuer la probabilité de congestion et la ranemer à une valeur proche de celle pour les trafics *aléatoires* [35, 36]. Pour cette raison, nous utilisons un circuit de routage intégrant le contrôle de flux. Cependant le mode de routage *wormhole* associé à un contrôle de flux peut introduire de l'*interblocage* dans le réseau [11].

Après avoir choisi les caractéristiques du circuit de routage, il reste à choisir le circuit de routage et le processeur adapté. Ce processeur doit être capable d'assurer les fonctions de traitement de données relatives au algorithmes de sélection. Nous avons donc besoin d'un processeur qui allie à la fois le traitement et la gestion des communications, c'est essentiellement pour cette raisons que nous avons choisi la technologie *Transputer T9000–C104* .

II.2.3 Intérêt de la technologie *Transputer T9000–C104*

Au moment de l'élaboration du projet, *INMOS* annonçait la sortie du *Transputer T9000* et du circuit de routage *C104*.

Le *Transputer T9000* est un processeur *RISC* superscalaire suffisamment performant pour assurer le déroulement des algorithmes de sélection du système de déclenchement de niveau–2. Il possède un gestionnaire d'ordonnancement *scheduler* microcodé très performant qui assure le changement de contexte entre deux tâches en moins de $1\mu s$. Par ailleurs, quatre liens séries gérés par un *Direct Memory Access* permettent l'échange de données sans consommation de temps *CPU*.
Ces caractéristiques lui confèrent une excellente flexibilité pour la construction de réseau.

Le circuit de routage *C104* est une matrice d'interconnexion $32 * 32$ liens simultannés ayant un temps de latence par paquet de $1\mu s$ utilisant le mode de routage *wormhole*.

Ces deux circuits intégrés peuvent être associés sans aucune interface matérielle ou logicielle. Ils permettent la construction de réseaux adaptés aux besoins spécifiques de l'application.

La gestion des communications est directement assurée par les deux circuits notamment pour:

- la gestion des paquets.
- la synchronisation des échanges.
- le multiplexage des canaux virtuels.

Ces fonctions sont d'autant plus performantes qu'elles sont mise en œuvre sur le silicium et non au niveau logiciel.

Ces deux composants fournissent toutes les fonctionnalités requises pour l'assemblage d'événements. L'échange de données à travers un canal virtuel est entièrement géré par les composants, il ne nécessite que la connaissance de l'adresse *source*, l'adresse *destination* et de la taille du bloc échangé.

Le *Transputer T9000* et le circuit de routage sont décrit plus précisément dans l'annexe A.

II.3 Structure de système de déclenchement de niveau–2 en technologie *Transputer*

Ce système doit s'intégrer dans la structure générale d'acquisition de l'expérience L3, sans induire aucune modification en amont ou en aval du niveau–2. Il est constitué de trois étages.

L'étage d'entrée dont les fonctions sont:

1. d'acquérir les informations déclenchement provenant des différents détecteurs à chaque croisement des faisceaux dans des mémoires *First In First Out (FIFO)*,
2. de mémoriser les croisements validés par le système déclenchement de niveau-1 en transférant les données des mémoires *FIFO* dans la mémoire multi–événement,
3. de transférer l'information nécessaire à l'assemblage de l'événement sur demande des unités de traitement,

Cet étage est composé de 12 cartes mémoires appelées *Tmb, Transputer Multi–event Buffer*. Il est synchronisé par des signaux de contrôle électroniques du LEP et de l'expérience.

L'étage central constitué d'une *ferme* d'*unités de traitement* interconnectée aux mémoires d'entrée par deux circuits de routage. Ses fonctions sont:

1. d'assembler un événement en rangeant les données selon un format prédéfini,
2. de traiter l'événement en lui appliquant l'algorithme correspondant au déclenchement de niveau–1,
3. de prendre la décision du déclenchement du niveau–2 (accepter ou rejeter), à partir du résultat du traitement et des conditions de rejet demandées lors de la prise de données (rejet effectif, échantillonnage des rejetés..),
4. d'envoyer la décision aux processeurs de réduction de données (*DRP*) du détecteur *TEC* via le site *Tmb Driver*,
5. enfin de transférer l'événement et le résultat associé vers l'étage de sortie.

Cet étage, controlé par logiciel, fonctionne en mode asynchrone par rapport à l'étage précédent.

L'étage de sortie qui a pour fonction de stocker l'événement et son résultat en attendant son transfert vers le système d'assemblage central. Il est constitué par le module *FT9000* qui permet l'interface entre le monde *Transputer* et le monde *FASTBUS*.

La figure II.4 présente le schéma du système de déclenchement de niveau–2.

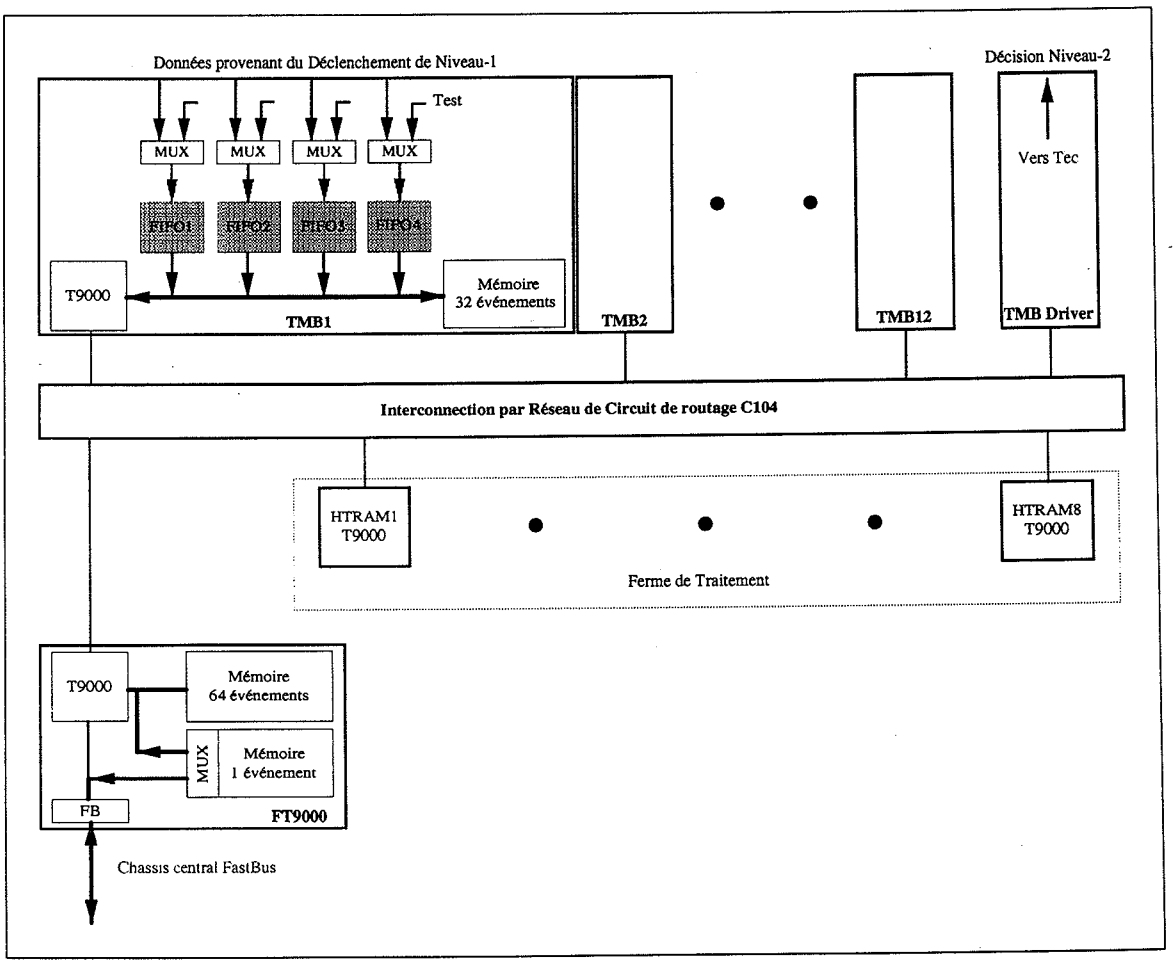


FIG. II.4 - *Schéma du système de déclenchement de niveau–2.*

Sur les mémoires d'entrée *Tmb*, le transfert des données depuis les mémoires *FIFO* dans la mémoire multi–événements est contrôlée par un *Transputer T9000* et effectué par le bus de données. Les données sont transmises aux *unités de traitement* par les liens de communication du T9000. L'interconnexion entre tous les *Transputers* est assurée par un réseau de circuits de routage dynamiques *C104*. La ferme de traitement est constituée de plusieurs *Transputers T9000* implantés soit sur des cartes HTRAM[3] soit sur des cartes *Tmb*.

Ce nouveau système comprend une procédure de test *in situ* qui permet de vérifier le bon fonctionnement du système à tout instant. Il reproduit les *conditions réelles* de prise de données, *indépendamment* de la situation de la chaîne d'acquisition en amont et aval. Le système de test consiste essentiellement en un injecteur de données controlé par un *Transputer* localisé sur la carte mémoire d'entrée *Tmb*.

II.4 Fonctionnement

A chaque croisement des faisceaux, chaque mémoire *FIFO* est initialisée pour recevoir les données associées à ce croisement. L'ensemble des données est collecté en parallèle sur 43 ports d'entrée en moins de $10\mu s$.

Si le croisement, n'est pas validé par le déclenchement de niveau–1, c'est à dire que le signal "Level-1" n'est pas envoyé, les *FIFO*s sont remis à zéro, prets à recevoir le prochain croisement. Si le croisement est validé, le signal "Level-1" active la procédure de transfert des données vers la mémoire tampon profonde de 32 événements. Ce transfert est effectué par le bus de données du *Transputer* et doit être exécuté en moin de $500\mu s$.

Sur l'étage d'entrée, chaque événement est ainsi "éclaté" en 43 paquets de données à raison de 4 paquets au maximun par carte *Tmb* (sur quelques modules *Tmb*, les quatres ports ne sont pas tous utilisés). Certains paquets sont constitués de données non rangées appelée *partitions*, dont l'ordre est défini par des considérations de disponibilité en fonction du temps. Leur utilisation par un algorithme nécessite un réarrangement. Ce re-ordonancement est très couteux en temps s'il est effectué par logiciel. Une optimisation est obtenue en combinant le réarrangement avec l'assemblage de l'événement.

Dès qu'elle est disponible chaque unité de traitement de la ferme émet une demande pour recevoir un événement. Sa demande est enregistrée par un contrôleur servant les unités en respectant l'ordre des demandes. Lorsque son tour d'être servie arrive, les *Tmb*s lui transmettent d'abord le nombre de mots de chaque *partition*. L'unité prépare la réception des *partition*s en leur affectant l'espace mémoire, en accord avec le format prédéfini. Cette pré–affectation permet de ranger les *partition*s reçues directement à leur place, quel que soit leur ordre d'arrivée. Ainsi le transfert des *partition*s, qui ne nécessite aucune synchronisation entre les mémoires d'entrée, peut être initialisé de façon autonome par chaque *Transputer Tmb*, dès qu'il est prêt à emettre, et géré par les processeurs de canaux virtuels via le protocole de communication (cf Annexe A).

Lorsque toutes les *partition*s ont été réceptionnées, l'unité commence le traitement en déroulant l'algorithme associé à la signature du déclenchement de niveau–1. A la fin du traitement, en fonction du résultat obtenu, des rejets activés et du facteur d'échantillonnage des rejetés, la décision finale est prise: soit *accepter*, soit *rejeter* l'événement.

La décision est immédiatement transmise aux processeurs *DRP* chargés de la réduction des données dans la *TEC*, dont la capacité de traitement est limitée à quelques dizaines de Hertz. En ne traitant que les croisements validés par le déclenchement de niveau–2, la *TEC* augmente

3. High performance TrAnsputer Module

artificiellement sa capacité de traitement de près de 50 %, au prix d'un temps de latence de quelques millisecondes.

L'événement assemblé et les résultats du traitement sont ensuite envoyés vers la mémoire de sortie, profonde de 64 événements. Cette mémoire à accès partagé *Transputer/FASTBUS* permet le transfert des données vers le système central d'assemblage des données.

II.5 Les Mémoires d'entrées *Tmb*

Chaque module est constitué de deux parties correspondant aux fonctionnalités du cahier des charges appelées respectivement *Acquisition* et *Injection*. Les données expérimentales et les données injectées sont multiplexées à l'entrée de la partie *Acquisition*.

Pour des raisons d'encombrement physique sur une carte au format *FASTBUS*, chaque module *Tmb* comporte 4 ports. La figure II.5 donne le bloc diagramme fonctionnel d'un module *Tmb*.

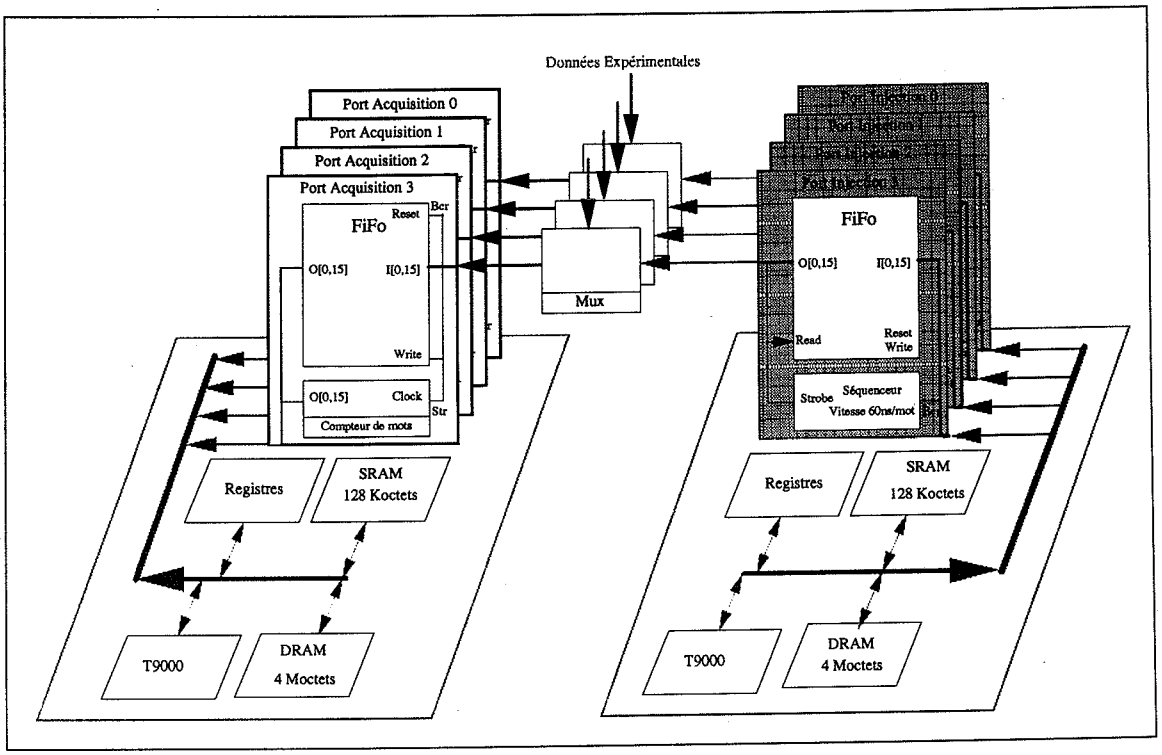


FIG. II.5 - *Diagramme fonctionnel des modules* Tmb

II.5.1 L'acquisition des données

Chaque port est constitué:

- d'une mémoire *First In First Out (FIFO)* profonde de 4096 mots de 2 octets,
- d'un compteur de mots permettant de connaître le nombre de mots présents dans la mémoire *FIFO*.

Sur chaque module, la réception du signal *Bcr* [4] remet à zéro les mémoires *FIFO* et les compteurs de mots des différents ports. Sur chaque port, la réception de l'impulsion *Strobe* [5] incrémente le compteur de mots et permet l'écriture d'une donnée dans la mémoire *FIFO*.

4. signal de synchronisation avec le *LEP*: croisement de faiseaux
5. signal de validation d'une donnée

II.5.2 La mémorisation des données

Lorsque le système de déclenchement de niveau–1 valide l'événement, il génère l'impulsion *Lv1*. La réception de cette impulsion active:

- le signal de contrôle *Overflow* indiquant que les modules ne sont pas disponibles pour recevoir le prochain événement. Ce signal est obtenu en effectuant un *Ou logique* avec des signaux *Overflow local* relatif à chaque module.
- le transfert des données des mémoires *FIFO* vers une mémoire tampon profonde de 32 événements.

La synchronisation entre le signal expérimental *Lv1* et la tâche de transfert est assuré par un canal de synchronisation matérielle *Event* du *Transputer* d'acquisition. Lorsque le transfert des données est effectué sur l'ensemble des modules le signal *Overflow* devient inactif indiquant que ces modules sont de nouveau prêts à recevoir le prochain événement.

Lorsque le systéme de déclenchement de niveau–1 ne valide pas l'événement:

- le transfert de l'événement dans la mémoire tampon n'est pas effectué,
- les mémoires *FIFO* sont remises à zéro lors de la réception du prochain signal *Bcr*.

II.5.3 L'injection de données

Chaque port de la partie injection est constitué:

- d'un séquenceur qui permet de reproduire la séquence *Data–Strobe* à 60 ns par mots (cf figure B.1),
- d'une mémoire *FIFO* profonde de 4096 mots de 2 octets où sont stockées les données à rejouer.

Apres avoir chargé les données à rejouer dans les mémoires *FIFO*, le *Transputer* génère le signal de contrôle *Bcr* qui démarre l'émission des données. Lorsque l'ensemble des données sont émises, le séquenceur s'arrête, le signal de contrôle *Lv1* est ensuiteémis selon un tirage aléatoire. Des canaux de synchronisation matérielle permettent la gestion de la séquence d'injection à l'intérieur d'un module, entre la partie *Acquisition* et la partie *Injection*, et entre les modules.

Ce module a été conçu et réalisé au *LAPP*[6] L'annexe B donne une description technique plus détaillée de ce module.

II.6 La Ferme de Traitement

La ferme de traitement est constituée de 1 à 8 cartes *HTRAM*. Chaque carte *HTRAM* est composée d'un *Transputer T9000* et de 8 Moctects de mémoire dynamique. Les 8 cartes *HTRAM* sont réparties sur 2 cartes de traitement comprenant chacune:

- 4 cartes *HTRAM*,
- 2 circuits de routage *C104*.

L'interconnexion entre les différents sites est assurée par un réseau de contrôle et un réseau de données.

Le réseau de contrôle permet:

- à l'initialisation de définir et de programmer les différents *noeuds* (*Transputer* ou *C104*) constituant le réseau,

6. Laboratoire d'Annecy–Le–Vieux de Physique des Particules

- la surveillance de ce réseau en cours de fonctionnement.

Le réseau de contrôle est représenté en *traits pointillés* sur la figure II.6. Il est réalisé par chaînage des différents sites pour des raisons de faciliter de vérification du réseau.

Le réseau de données est constitué comme suit. Chaque carte de traitement a deux de ses liens connectés à deux liens consécutifs sur chaque circuit de routage *C104*. Ceci permet:

- l'utilisation du *routage adaptatif groupé* (cf Annexe A) mis en œuvre sur le *C104*. Ce routage, géré par le *C104*, permet de répartir les paquets à destination de cette unité de traitement sur les deux liens,
- de répartir la charge des données provenant des mémoires *Tmb*s sur le 2 *C104*.

Dans une première approche, nous avons simplement réparti équitablement les liens provenant des mémoires *Tmb*s entre les deux *C104*. Le réseau de données est représenté en *traits pleins* sur la figure II.6.

La topologie simple du réseau, structure en arbre, nous assure que le réseau est sans inter-blocage.

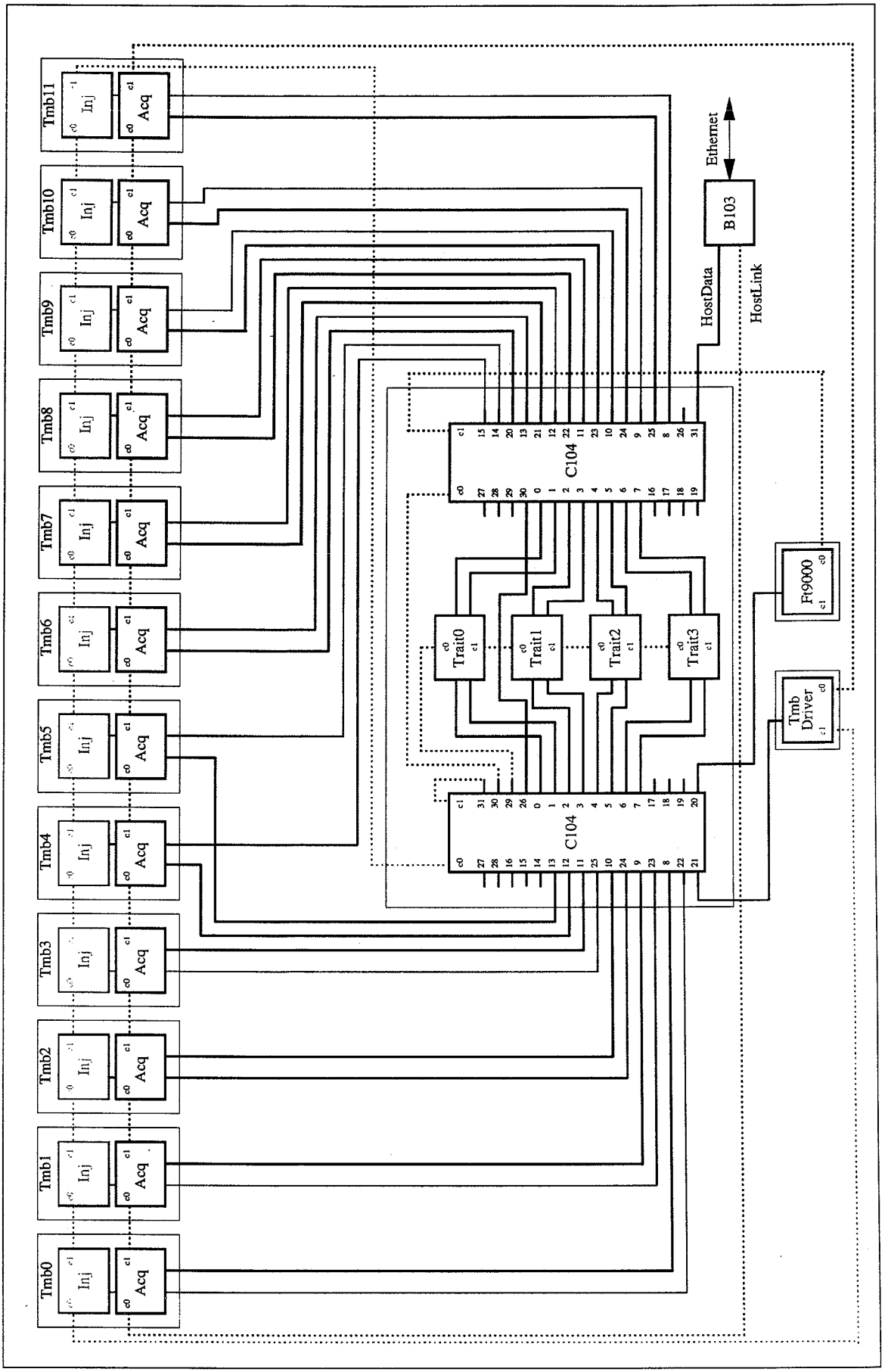


FIG. II.6 - *Schéma présentant l'interconnexion des différents* Transputer

La carte d'interconnexion a été conçue et réalisée au CERN.

II.7 L'interface *FASTBUS-Transputer*: le module *FT9000*

II.7.1 Présentation du module *FT9000*

Ce module est une adaptation du module *FT800* [32] à la famille *T9000*. Il est composé:

- d'un *Transputer T9000* dont le lien 0 est utilisé pour la communication avec un maître *FASTBUS* à travers le registre *CSR0*,
- d'une mémoire dynamique DRAM de 8 Moctets utilisée par le *Transputer*,
- d'une mémoire statique SRAM de 512 Koctets constituant l'espace mémoire partagé par le monde *Transputer* et *FASTBUS*. L'accès à cette mémoire est géré par un multiplexeur contrôlé par le registre *CSR0*.

La figure II.7 présente le bloc diagramme fonctionnel du module *Ft9000*.

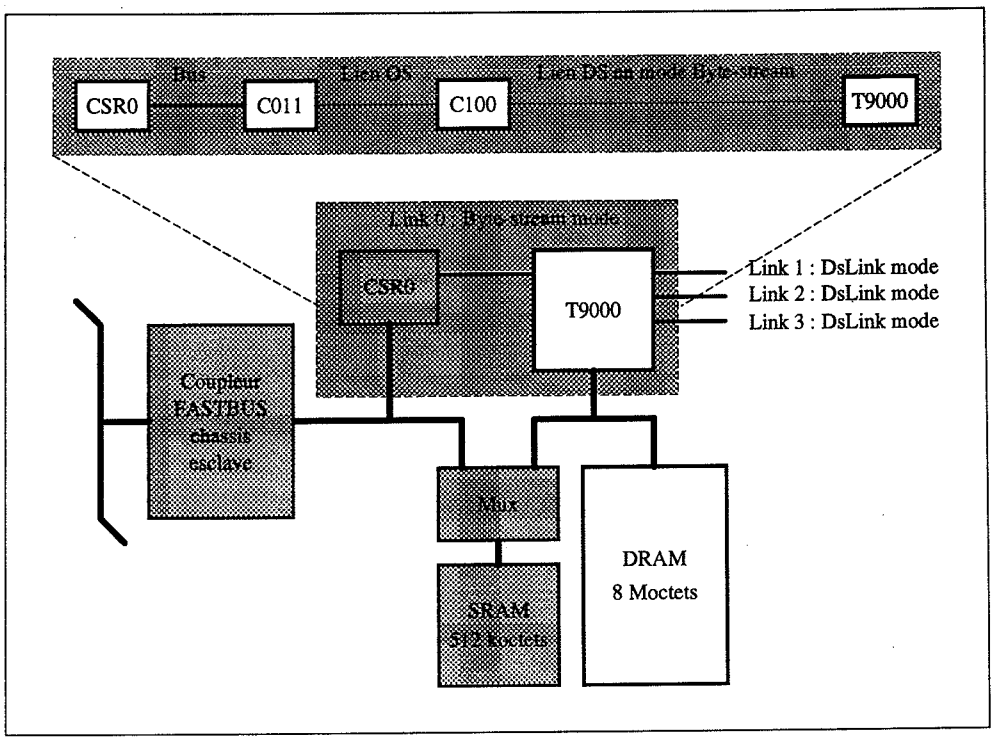


FIG. II.7 - *Schéma présentant le bloc diagramme fonctionnel du module* Ft9000

II.7.2 Principe de fonctionnement

L'accès à la mémoire partagée (*SRAM*) est géré par le monde *FASTBUS*. Cette mémoire est utilisée pour ne stocker qu'un seul événement. Lorsque la mémoire est vide le maître *FASTBUS* autorise l'accés à la mémoire pour le *Transputer* en positionnant un bit du registre *CSR0*. L'affection de ce bit:

- positionne le multiplexeur pour donner l'accès à la mémoire pour le *Transputer*,
- et transmet l'autorisation d'accès au *Transputer*.

Une fois le transfert de la mémoire multi–événements (*DRAM*) vers la mémoire partagée (*SRAM*) effectué, le *Transputer* transmet un acquittement qui positionne un bit dans le registre *CSR0*. Ce bit indique au maître *FASTBUS* qu'il y a un événement à lire et qu'il doit reprendre l'accès à la mémoire. Le *Transputer* se met alors en attente d'une nouvelle autorisation.

II.7.3 La synchronisation *FASTBUS–Transputer*

La synchronisation sur le module *Ft800* est obtenue par un interface appelé *Link Adaptor* mis en œuvre par le circuit *IMS C011* [17]. Il permet l'échange de données entre des liens série *OSLink* et un bus paralléle.

Une adaptation simple du module *FT800* au *Transputer T9000* consiste à utiliser le circuit *IMS C100* qui traduit le protocole *OSLink* en protocole *DSLink* et réciproquement.

Le circuit *IMS C100*[18] permet d'interfacer des réseaux *T8xx* avec des réseaux *T9000* selon plusieurs modes. Le mode choisi ici permet d'utiliser un système *T8xx* à partir d'un *Transputer T9000* sans modification de code sur le système *T8xx*. Ce mode est caractérisé comme suit:

- Les liens *T9000* connectés au *IMS C100* travaillent en *byte–stream mode*[7].
- Lorsque la communication s'effectue du système *T8xx* vers le *Transputer T9000*, le *Transputer T9000* doit transmettre auparavant la longeur du message qu'il souhaite recevoir. La longueur du message est transmise lorsque le *Transputer T9000* accède le canal travaillant en *byte–stream mode*: il faut donc s'assurer que le *Transputer T9000* est toujours prêt à recevoir le message avant que le système *T8xx* ne le transmette.
- Si un octet est émis par le système *T8xx* et que la longeur du message n'a pas été transmise par le *Transputer T9000*, un paquet spécifique est transmis permettant l'utilisation des *gardes*(cf Annexe A.1).

7. Protocole d'échange entre un *Transputer T9000* et un réseau *T8xx* interfacé par un *C100*

Chapitre III

Organisation logicielle distribuée du déclenchement de niveau–2

III.1 Choix logiciels

L'organisation générale du logiciel découle d'un certain nombre de choix logiciels en partie contraints par le cahier des charges expérimental (exposé au paragraphe II.1). D'autres choix propres au logiciel sont par ailleurs purement techniques visant soit à une meilleure structuration, modularité ou extensibilité du logiciel soit à une meilleure utilisation des possibilités de la technologie *Transputer*. Parmi les contraintes expérimentales, nous retiendrons les différents points qui intéressent le logiciel à savoir:

- la minimisation du temps de latence des événements,
- la restitution des événements dans l'ordre de déclenchement niveau-1,
- et le réarrangement des partitions combiné à l'assemblage de l'événement.

III.1.1 Choix d'un assemblage dynamique

Ce dernier point est essentiel et la première caractéristique du logiciel réside dans le choix de mettre en œuvre un assemblage dynamique. Cette technique s'oppose radicalement à ce que permettrait dans notre cas un assemblage logiciel opérant en deux temps: réception des données puis mise au format. Il est clair que notre premier souci est d'éviter lorsque cela est possible tout surcoût de temps purement logiciel qui pénalise le temps de latence de l'événement. Or dans ce type d'assemblage, les sources transmettent leurs données puis les blocs sont assemblés et mis au format de façon logicielle par les destinations.
L'assemblage dynamique permet quant à lui le transfert par les sources de blocs de données qui sont directement assemblés au fomat prédéfini dés leur réception par les destinations. Cette solution est d'autant plus souhaitable dans notre cas qu'elle est parfaitement adaptée à la technologie *Transputer*.

Le *transputer T9000* apporte en effet une solution nouvelle à la gestion des communications point à point à travers un réseau de processeurs. Du point de vue logiciel, des processus peuvent communiquer avec d'autres processus ne se trouvant pas sur un processeur voisin. Un nombre quelconque de liens virtuels entre processus pouvent être multiplexés sur un même lien physique. Ces possibilités de communications autorisent un assemblage dynamique où chaque partition émise depuis les mémoires *Tmbs* est directement assemblée et réceptionnée selon le format souhaité.

Il est par ailleurs intéressant d'exploiter au mieux les meilleures performances du *Transputer T9000*, notamment la gestion matérielle des canaux virtuels et des processus. Le contexte multitâche offre également de bonnes performances (le coût du changement de contexte reste inférieur à la microseconde) qui permettent à plusieurs processus de s'exécuter de façon concurrente sur un même processeur. Un processus en attente de communication est suspendu au profit d'un autre et repris lorsque la communication est achevée sans que le processeur n'ait été perturbé par les données transmises. Le *Transputer T9000* permet donc d'effectuer une synchronisation automatique avec les transfert de données. Le logiciel développé et présenté ici exploite ce mécanisme de la manière suivante.

Dans notre application, le point délicat se situe sur les sites destinations chargés de l'assemblage au format prédéfini. Ces derniers doivent réceptionner 76 partitions à chaque événement. On effectue la réception en parallèle afin d'optimiser la bande passante des liens de réception. Ainsi à chaque événement, le logiciel active une liste contenant autant de tâches qu'il y a de partitions à recevoir. Chaque tâche est associée directement à une partition et donc à un canal virtuel. Il convient de noter que les tâches sont allouées une fois pour toutes en début de programme et sont uniquement réactivées à chaque événement. Seul, un petit nombre de paramètres "événement dépendants" doit être positionné avant cette opération.
On opère de manière similaire sur les sites *source* (*Tmbs*).

Ce choix de programmation offrent les avantages suivants:

- La synchronisation de toutes les tâches avec le transfert de données est totalement transparente.
- La modularité et l'extensibilité vis à vis du nombre de partitions et des changements de format possibles sont optimales.
- Ce choix offre de bonnes performances que nous préciserons plus loin.

III.1.2 Gestion de la ferme de traitement

La gestion de la ferme de traitement recouvre plusieurs choix liés autant au mécanisme d'alimentation en événements des différentes unités qu'à l'optimisation du système. Différents mécanismes d'alimentation sont possibles qui peuvent être directement pilotés par le flux de données lui même (mécanisme "data driven") ou bien élaborés à partir d'un flux de contrôle (mécanisme "control driven").

Dans notre cas, minimiser le temps de latence de la réponse fournie à la TEC est l'une des contraintes fortes du système. Cette information relative à chaque événement est obtenue à l'issue du traitement de niveau-2. Ainsi, il est préférable de distribuer les événements dans l'ordre de leur arrivée vers les unités de traitement. L'ordre d'arrivée (ou de déclenchement de niveau-1) des événements est donc conservé au niveau des mémoires d'entrées *tmbs*; chaque événement doit ensuite être traité par l'unité la plus rapidement libre. Comme le temps de traitement est tres variable selon la nature de l'évenement, il préférable que unité envoie une demande dès qu'elle est libre. Par ailleurs, dès que cela est possible, on souhaite pouvoir alimenter en parallèle plusieurs unités disponibles au même moment pour le traitement des événements suivants.

D'un point de vue logiciel, dès que l'on privilégie une information de contrôle (comme ici l'information qu'une unité est libre), le mécanisme d'alimentation doit assurer la synchronisation des différents flux de données pilotés par cette information de contrôle vers la destination. Il s'agit bien entendu de respecter la cohérence des événements répartis sur l'ensemble des sources. La façon la plus simple et la plus fiable de gérer la demande simultanée de deux unités disponibles en assurant la cohérence des événements et en minimisant le temps de latence de chaque événement consiste à mettre en œuvre une gestion centralisée de la ferme par l'intermédiaire d'un site *serveur*. Celui-ci est alors chargé d'élaborer le flux de contrôle utile aux sites sources et d'alimenter la ferme en conséquence c'est à dire de piloter les flux de données de façon synchrone vers les destinations. Ceci est effectué en sérialisant les demandes simultanées des unités de traitement d'une part, puis en diffusant en parallèle ces demandes vers l'ensemble des sites *Tmbs* d'autre part.

Les temps de traitement par les algorithmes sont, nous l'avons vu, très variables d'un événement à l'autre. Bien que le temps de latence de chaque événement soit minimal, l'ordre de la réponse fournie par le niveau-2 est donc quelconque. La présence d'un site *serveur* permet également de restituer les événements dans l'ordre de déclenchement de niveau-1 sur l'interface de sortie.

III.1.3 Nécessité d'un protocole d'assemblage

Le logiciel d'assemblage des événements intègre un protocole rendu nécessaire par trois contraintes expérimentales.

La première est la présence de ports dont le nombre de mots est variable d'un événement à l'autre. Ainsi, certaines partitions à transmettre sont de longueur variable.

La seconde concerne le souci absolu d'assurer la cohérence des événements assemblés. Il a donc été décider à la demande des physiciens d'adjoindre au logiciel une vérification en ligne de la cohérence des événements.

Enfin, le niveau-2 peut contrôler que sur les ports fixes, le nombre de mots lus est bien celui attendu. Ceci constitue une vérification en ligne intéressante du bon fonctionnement du système amont. Ainsi, le logiciel d'assemblage véhicule pour chaque événement un bloc d'information relatif aux éventuelles erreurs de lecture par port.

La mise en place d'un protocole d'assemblage simple entre sources et destinations permet de satisfaire ces trois points. A chaque partition est associée un en-tête comprenant toutes les informations nécessaires au site de destination pour réceptionner les partitions et vérifier la cohérence et l'intégrité des informations qui vont être reçues (on se reportera à la description du protocole).

III.1.4 Définition et intérêt du logiciel *MBM*

Le logiciel *MBM* ("*Memory Buffer Manager*") est une librairie utilitaire conçue pour structurer l'unité de code présent sur chaque site *transputer* et qui définit l'organisation logicielle sur chaque processeur. Cet ensemble logiciel (décrit en détail en Annexe C) utilise une technique classique souvent retenue pour gérer des flux de données dans un environnement multi-tâches. Ses fonctions sont d'organiser les informations en mémoire et de gérer les différentes tâches qui opérent sur ces données. Ces tâches ou processus définissent des opérateurs pour le logiciel *"MBM"*. Son intérêt est de proposer :

- un outil de définition de la zone-mémoire utile,
- une procédure de déclaration des différents opérateurs,
- tous les mécanismes d'accès aux informations stockées dans la zone-mémoire (demande et restitution d'information)
- tous les mécanismes de synchronisation entre les différents opérateurs accèdant la zone-mémoire.

Les mécanismes de synchronisation sont élaborés une fois pour toutes et totalement transparents pour les couches logicielles supérieures. Le développement de cette librairie a permis de structurer efficacement le code et de manière reproductible sur chaque site.

III.1.5 Utilisation du transcodeur f2c

Les algorithmes de sélection des événements sont écrits en Fortran. Pour des raisons de compatibilité avec le logiciel utilisé hors ligne pour l'analyse des événements, les physiciens ne souhaitaient pas deux versions des mêmes algorithmes. Aussi il a été décidé de conserver le langage Fortran et d'utiliser le transcodeur Fortran-C *f2c* du domaine public (*GNU*). Le code C généré constitue un point d'entrée appelé par les opérateurs de traitement. Une légére otimisation relative aux manipulations de bits en entiers de 16 et 32 bits (abondamment usitée par les traitements en ligne) a été apportée à cet outil dont le portage sur *transputer* ne pose aucun problème et qui depuis satisfait nos besoins.

III.2 Organisation générale

On distingue trois types de sites correspondant aux trois étages qui constituent la structure du système de déclenchement de niveau–2 :

- les sites de mémorisation des données (*Tmbs*),
- les sites *unités de traitement*,
- le site d'interface avec le système d'assemblage central (*FT9000*).

Les données relatives aux événements transitent sur les différents sites du réseau. Sur chaque site, il faut gérer la présence de processus producteurs et consommateurs de données autour d'une zone de mémoire commune partitionnée et organisée en tampon (cf. figure III.1). Il est donc souhaitable que la gestion du flux de données soit identique sur l'ensemble des sites.

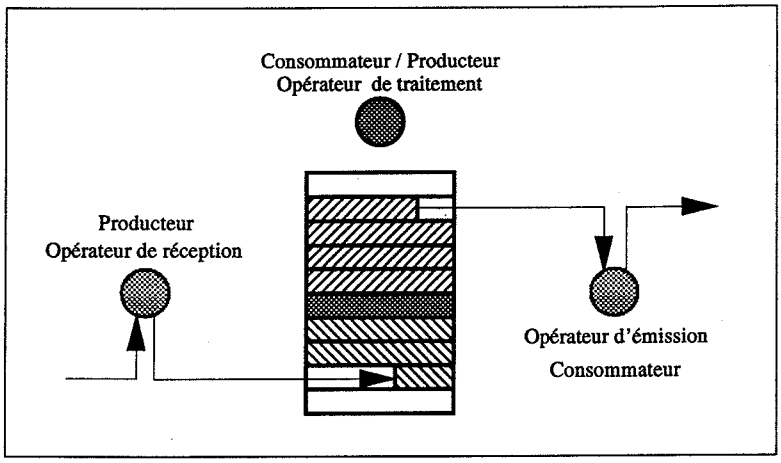


FIG. III.1 - *Schéma d'un gestionnaire de flux de données*

De manière générale, on définit un certain nombre d'opérateurs de réception, de traitement et d'émission correspondant aux différentes opérations à effectuer sur les données transitant sur chaque site. Les données sont stockées dans une zone mémoire–tampon de profondeur fixe définie à l'initialisation et partagée en emplacements de la taille d'un événement. L'ordre d'exécution des opérateurs définit un cycle opératoire. Chaque événement subit l'ensemble des opérations du cycle opératoire. En outre à chaque opérateur, on associe une ou plusieurs tâches. Ainsi:

- au niveau de chaque site, il existe une *structure pipeline* de profondeur la taille du cycle opératoire,
- au niveau de chaque opérateur, on autorise un *parallélisme réplicatif*. (A chaque opérateur est associé une ou plusieurs tâches concurrentes autorisant le cas échéant le traitement en parallèle de plusieurs événements).

La mise en œuvre de ce système nécessite le choix d'un mode de gestion de la zone mémoire–tampon. On peut choisir une gestion d'espace mémoire associative où chaque emplacement est associé à un événement modulo la profondeur du tampon. Cependant elle présente certaines restrictions ne permettant pas d'exploiter au mieux l'espace mémoire dès que plusieurs tâches sont associées à un même opérateur. En effet, dans ce cas les emplacements sont directement corrélés aux événements. Les temps de traitement pouvant varier d'un événement à l'autre, il peut se trouver que les emplacements en amont de l'emplacement p soient disponibles avant l'emplacement p lui–même. Il est alors impossible d'utiliser ces emplacements libres : d'où l'idée de décorréler totalement les emplacements des événements qui y transitent et de priviligier la disponibilité des emplacements. On associe pour cela à chaque opérateur une liste *FIFO* des références dont celui–ci peut disposer. Avec ce type de gestion, le *parallélisme réplicatif* est optimum car l'occupation de la zone mémoire en fonctionnement aux limites est maximale.

Un logiciel de gestion de flux de données *Mbm* a été développé dans cette optique. Il est décrit en détail dans l'annexe C.

III.3 L'assemblage de l'événement par le réseau de *Transputers*

L'assemblage consiste à lire des données réparties dans différentes mémoires et à les réordonner selon un format prédéfini dans une mémoire centrale. Dans notre cas, la mémoire centrale est

constituée par la mémoire des sites de traitement.

Pour le déclenchement de niveau–2, l'assemblage doit non seulement rendre les données accessibles aux *unités de traitement* mais aussi restituer les événements au système d'assemblage central dans leur ordre de déclenchement de niveau–1. La figure III.2 présente la structure du système d'assemblage et de filtrage du niveau–2.

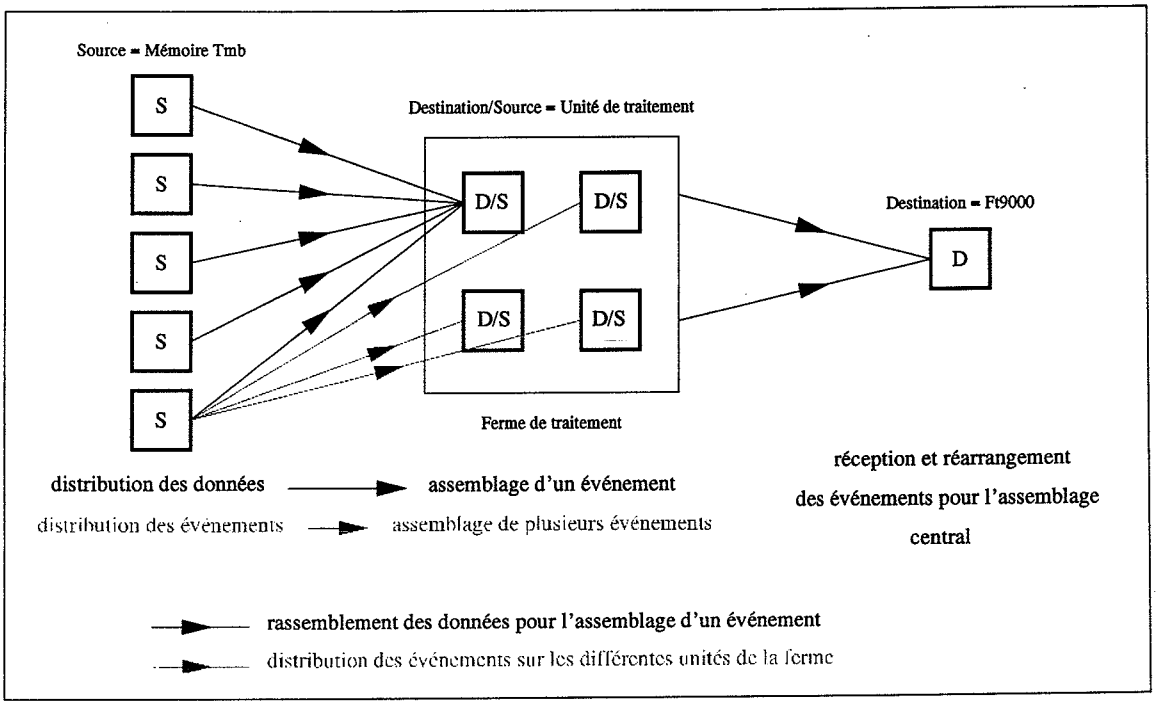


FIG. III.2 - *Structure d'assemblage d'événements*

Pour minimiser le temps de latence d'un événement et utiliser la bande passante de l'étage de traitement de façon optimale:

- sur chaque site, les événements sont gérés en *First In First Out.*
- le flux des données entre les sites *Tmb* et les sites *unité de traitement* est contrôlé par les unités de traitement. Ainsi dès qu'une unité de traitement est disponible, elle demande un nouvel événement aux sites *Tmb*.
- chaque site *Tmb* est capable de:
  - distribuer les événements dans leur ordre d'arrivée vers les unités de traitements,
  - distribuer en *parallèle* plusieurs événements distincts vers des unités de traitements distinctes.
- le flux des données entre les sites *unité de traitement* et le site *FT9000* est contrôlé par les données elles–mêmes. Ainsi dès qu'une unité de traitement a terminè l'ensemble de ces opérations, elle envoie les données au site *FT9000*.
- le site *FT9000* est capable de:
  - recevoir en *parallèle* les événements provenant de l'ensemble des unités de traitement,
  - présenter les événements au système d'assemblage central dans leur ordre d'arrivée sur les sites *Tmb*.

III.3.1 La structure de l'information et description de l'assemblage

Structure de l'information

L'information de déclenchement doit être assemblée selon le format précis attendu par les algorithmes de sélection. Ce format est constitué de différents blocs d'information, chacun des blocs contenant les données relatives à un détecteur.

Au niveau des ports d'entrée, les données issues d'un même détecteur peuvent être réparties sur plusieurs ports *Tmb*. De plus un même port est parfois utilisé pour acquérir les données provenant de plusieurs détecteurs.

L'information contenue dans les différents ports *Tmb* est donc partitionnée de façon à en extraire les données à destination d'un même bloc : ces données constituent une *partition*.

On distingue deux types de blocs :

- les blocs à nombre de mots fixes qui peuvent être constitués de plusieurs partitions réparties entre différents ports *Tmb*.
- les blocs à nombre de mots variables constitués d'une seule partition équivalent à un port *Tmb*.

Description de l'assemblage

Pour décrire l'assemblage sur un site de rassemblement, on énumère l'ensemble des partitions constituant l'événement et l'on indique pour chaque partition les informations suivantes:

- la source : numéro du module *Tmb*,
- la référence du bloc source : numéro du port dans le module *Tmb*,
- le décalage par rapport à l'origine du bloc source,
- la destination : référence du bloc destination dans le format de traitement,
- le décalage par rapport à l'origine du bloc destination,
- le nombre de mots contenu dans la partition s'il est connu,
- le nombre maximun de mots contenu dans la partition.

III.3.2 La cohérence et le réarrangement des événements

La cohérence d'un événement

La cohérence consiste à vérifier que les données des différentes partitions constituant un événement proviennent du même événement.

Elle exige:

- que l'on gère la demande simultannée de plusieurs unités de traitement [cohérence:1],
- que l'on assure une distribution synchrone des blocs d'événements des sites *Tmb* [cohérence:2].

Le réarrangement des événements

Le réarrangement des événements consiste à présenter les événements au système d'assemblage central dans leur ordre d'arrivée sur les sites *Tmb*.

Elle demande que sur le site *FT9000*:

- l'opérateur de rassemblement connaisse l'unité de traitement du prochain événement à recevoir [réarrangement:1],

- les opérateurs soient des opérateurs synchrones [réarrangement:2].

La condition [cohérence:1] est réalisée si les mémoires d'entrées *Tmb* sont utilisées comme des ressources partagées par les unités de traitement. Pour cela on définit un nouveau site: le site *Serveur*, dont le rôle est de sérialiser la demande des unitées de traitement puis de diffuser cette demande vers les mémoires d'entrées *Tmb*. Le contenu de cette demande est l'identificateur de l'unité de traitement.

La condition [réarrangement:1] est réalisée si le site *Serveur* transmet les demandes au site *FT9000* dans le même ordre qu'il les diffuse sur les sites *Tmb*.

III.3.3 Le principe de l'assemblage

Lorsque des unités de traitement sont disponibles, elle demandent l'accès à un événement par l'intermédiaire du site *Serveur*.

Celui-ci sérialise ces demandes puis les diffuse en *paralléle* sur les sites *Tmb* dans leur ordre de réception. Sur chaque site *Tmb*, on associe le même événement à la demande d'une unité de traitement puis on envoie un acquittement dès que cette association est effectuée. L'association {*demande d'événement*}-*événement* permet de réaliser la condition [cohérence:2], nous verrons plus loin le moyen permettant de réaliser cette condition.

Lorsque le site *Serveur* a reçu les acquittements des différents modules *Tmb*, il diffuse la demande d'événement vers le site *FT9000*. La figure III.3 présente le circuit de diffusion des demandes d'événements provenant des unités de traitement appelé flux de synchronisation.

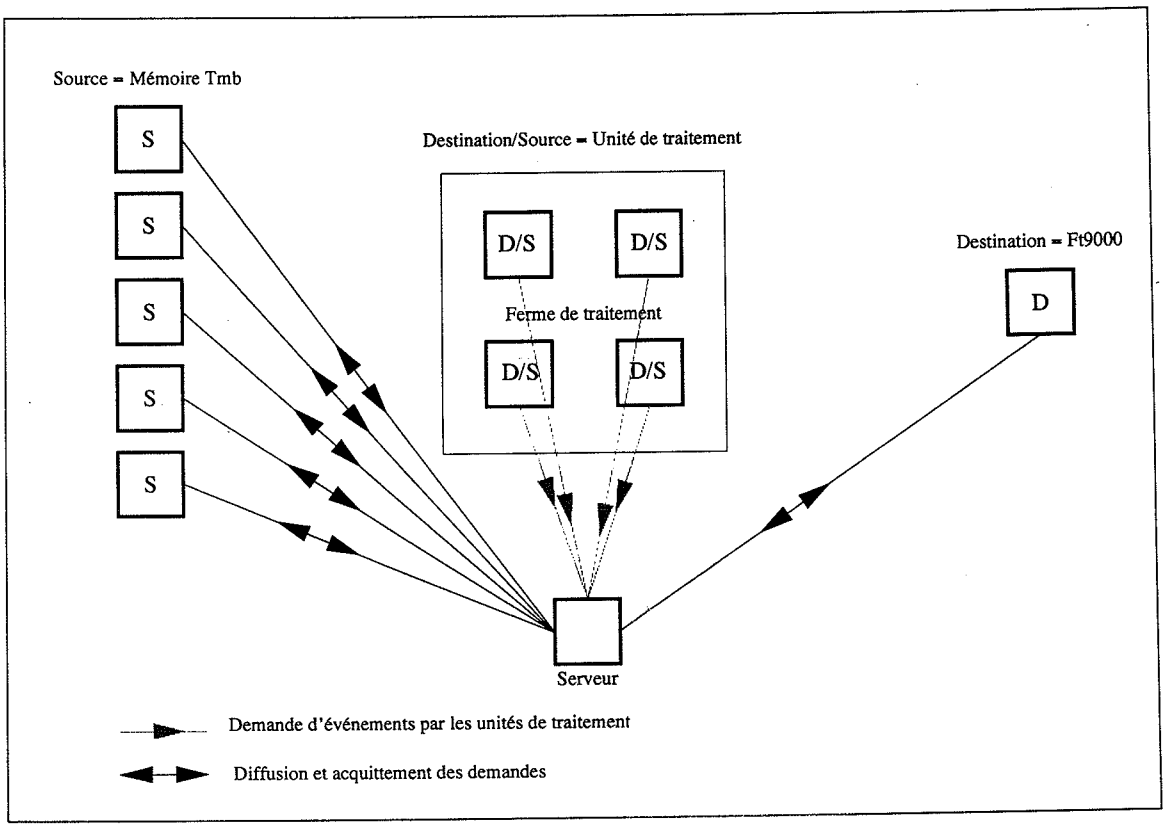


FIG. III.3 - *Schéma du flux de synchronisation.*

Sur chaque site *Tmb* une fois l'association {*demande d'événement*-}*événement* effectuée, on transmet en *parallèle* les différentes partitions vers l'unité de traitement.

Sur un site *unité de traitement*, dès que le serveur a acquitté la demande, on prépare la réception en *parallèle* des partitions constituant l'événement. Une fois l'assemblage de l'événement

effectué, l'événement est disponible pour l'opérateur de traitement. L'opération traitement comprend l'exécution d'algorithmes de sélection puis la transmission au site *Tmb Driver* du résultat du traitement. L'événement est ensuite disponible pour l'opérateur de diffusion vers le site *FT9000*.

Le site *FT9000* récupère l'identificateur de l'unité de traitement sur laquelle est localisé le prochain événement à transmettre au système d'assemblage central. Apres reception en *paralléle* des partitions le constituant, l'événement est assemblé puis transferé vers la mémoire *Fastbus–Transputer*.

III.3.4 Le protocole de l'assemblage

Les communications entre les différents sites sont effectuées par des canaux de communication externes. Ces canaux permettent le transfert de données d'un espace mémoire vers un autre espace mémoire.

Sur l'ensemble des sites de distribution et/ou de rassemblement de partitions, on associe à chaque partition un canal de communication externe.

Sur le site *Serveur*, on associe un canal de communication externe à chaque site *Tmb* à servir et à chaque unité de traitement pour recevoir la demande de celle–ci.

Pour que le transfert des partitions et la diffusion des demandes aient effectivement lieu en *parallèle* il faut associer un canal à une tâche. A chaque partition on associe un en–tête comprenant :

- un identificateur d'événement pour vérifier la cohérence,
- le nombre de mots de la partition pour permettre l'assemblage parallèle de l'événement.

Le protocole d'assemblage comprend :

- *sur les sites de distribution*:
  - l'émission des en–têtes d'assemblage,
  - l'émission des données.
- *sur les sites de rassemblement*:
  - la réception des en–têtes d'assemblage.
  - la vérification de la cohérence de l'événement,
  - la réception des données.

Organigramme général

La figure III.4 donne l'organigramme général du système d'assemblage. Il permet de voir, pour l'assemblage d'un événement:

- la séquence exécutée sur chaque site,
- le parallélisme entre les différents sites,
- les points de synchronisation entre ces sites.

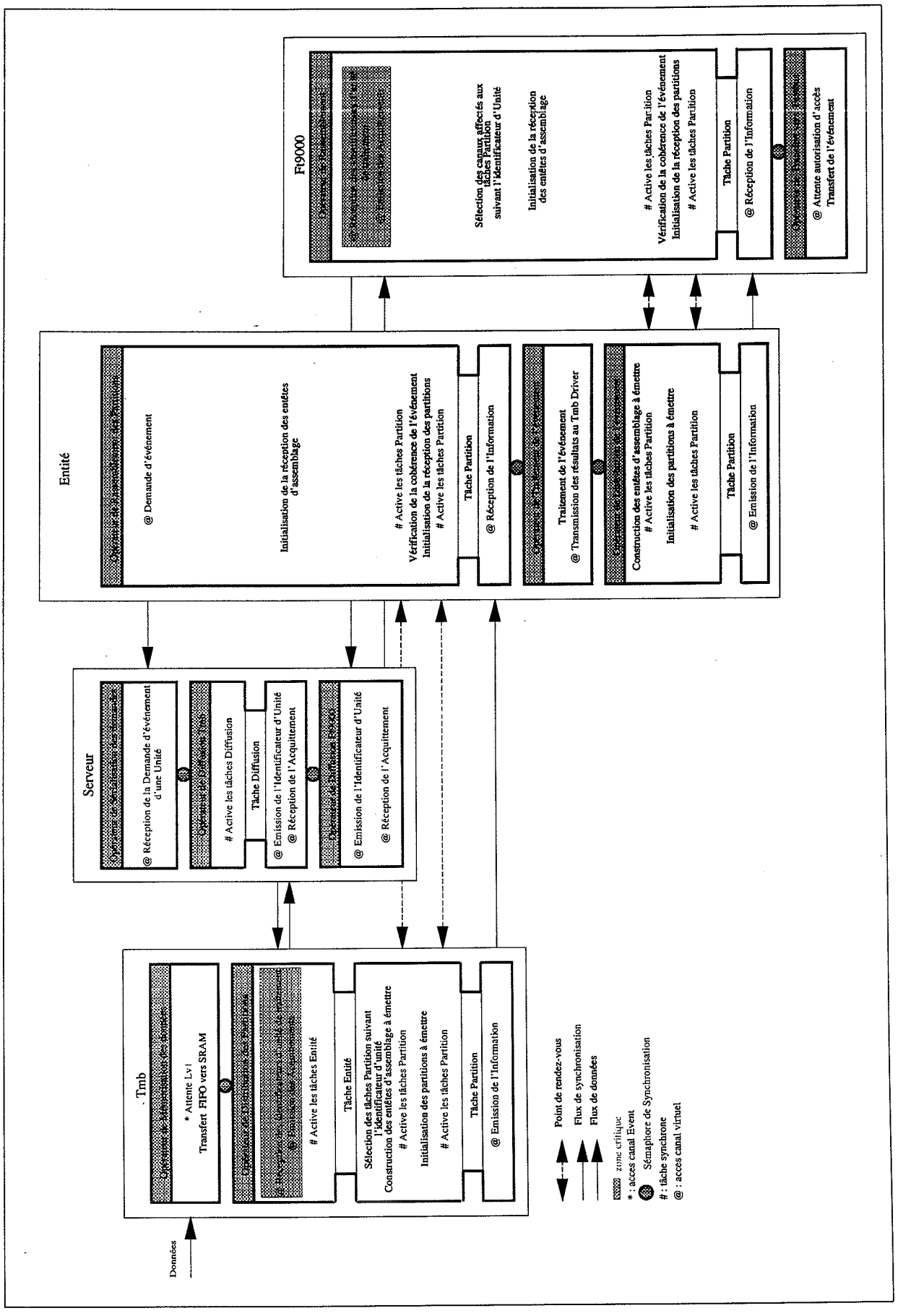


FIG. III.4 - *Organigramme du système d'assemblage.*

III.4 La structure logicielle sur les sites *Tmb*

Les fonctionnalités d'un site *Tmb* sont de:

1. transférer les données des différents ports actifs dans un emplacement mémoire en moins de 500 μs lorsque l'événement est validé par le déclenchement de niveau–1,
2. distribuer les données relatives à un même événement sur les différentes unités de traitement qui les demandent.

On définit alors sur chaque site *Tmb* un cycle opératoire composé de deux opérateurs correspondant respectivement aux deux fonctionnaliés décrites précédemment.

III.4.1 L'opérateur de mémorisation des données

Il constitue le premier opérateur du cycle opératoire. L'opération de transfert des données des différents ports actifs dans un emplacement mémoire utilise uniquement le bus de données du *Transputer*. Elle est effectuée par l'instruction **DevMove** mis en œuvre matériellement par le *Transputer*. Cette instruction ne permet pas la mise en place de parallélismes, on associe donc à cet opérateur qu'une seule tâche. Cette opération est critique pour l'expérience car elle ne doit pas introduire de temps mort supplémentaire. Pour cette raison, on souhaite que l'opération de mémorisation ne suspende pas son exécution sur un point de débranchement autre que le passage à zéro du sémaphore de synchronisation de sa liste de disponibilité indiquant que celle–ci est vide. Cette fonction est effectuée par un opérateur *haute priorité*. Lorsque le déclenchement de niveau–1 valide l'événenemt, la tâche de mémorisation est activée par la réception de la validation sur un canal d'interruption, sinon elle est inactive.

III.4.2 L'opérateur de distribution des partitions

Il constitue le dernier opérateur du cycle opératoire. L'opérateur de mémorisation ne possédant qu'une seule tâche, les événements sont disponibles dans l'ordre de leur arrivée sur le site *Tmb* pour l'opérateur de distribution. Celui–ci peut être un opérateur *asynchrone* et *basse priorité* contrairement à l'opérateur de mémorisation.

La distribution des partitions consiste à recevoir l'identificateur d'unité de traitement transmis par le serveur, de l'associer à un *bloc–événement*, puis de transmettre en parallèle ces partitions sur les canaux de communications relatifs à l'unité de traitement. On autorise la distribution en paralléle des partitions relatives à des événements distincts vers des unités de traitement distinctes. Ce parallélisme est réalisé en associant autant de tâches à l'opérateur que l'on autorise de distribution en paralléle d'événements distincts ($NbParEvent$). Ces tâches sont nommées tâches *Opérateur*.

Chaque tâche *Opérateur* doit pouvoir distribuer un événement sur les différentes unités de traitement. On définit une matrice d'interconnexion contenant les canaux de communications du site *Tmb* avec les unités de traitement dont l'élément de base est une liste de canaux profonde du nombre de partitions à transmettre par ce site *Tmb* vers une unité de traitement. On définit ensuite une matrice *Tâche* contenant les tâches associées à chacun des canaux de la matrice d'interconnexion. Chaque liste de la matrice *Tâche* est une ressource partagée par les tâches *Opérateur* dont l'accès est géré par un sémaphore binaire. Chaque tâche *Opérateur* utilise les éléments de la matrice *Tâche* comme des listes de tâches *synchrones*. La réception via le serveur de l'identificateur de l'unité de traitement permet la sélection de la liste adéquate. Le canal de réception de l'identificateur d'unité de traitement est une ressource partagée par les tâches *Opérateur* dont l'accès est géré par un sémaphore binaire.

Pour que la cohérence soit complète, il faut que la distribution d'événements fonctionne de façon synchrone entre les sites *Tmb* [cohérence:1.2]. Il s'ensuit que l'association d'un événement à un identificateur d'unité de traitement constitue une *zone critique* pour les tâches *Opérateur*.

L'accès a cette zone critique est géré par le sémaphore binaire associé au canal de réception de l'dentificateur d'unité de traitement.

III.4.3 La structure logicielle

La figure III.5 présente l'organisation logicielle sur un site *Tmb*.

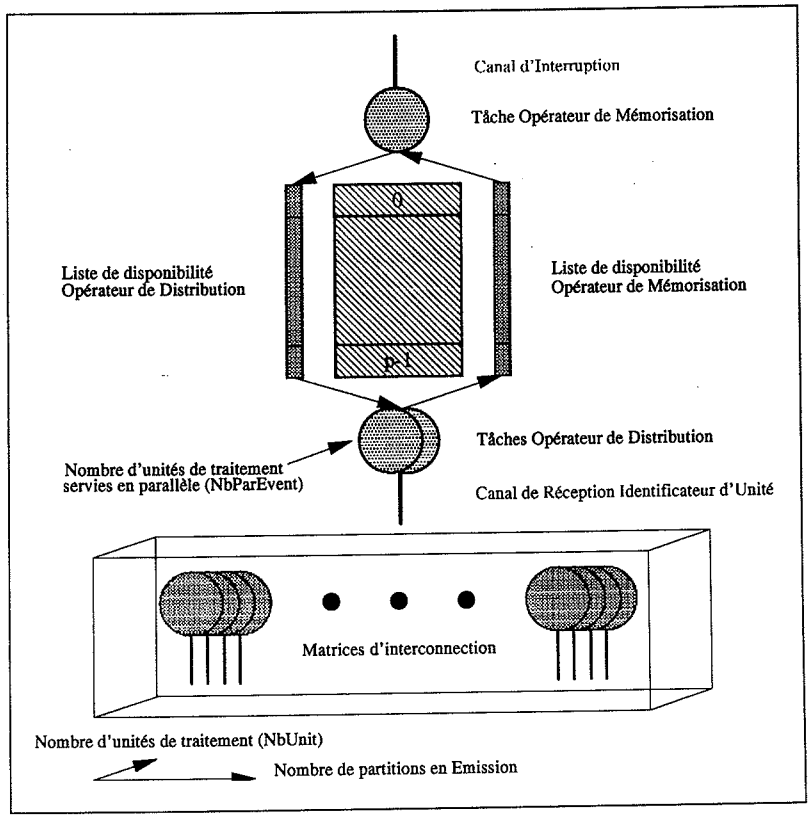


FIG. III.5 - *Structure logicielle sur un site* Tmb

Le tableau III.1 résume les caractéristiques du gestionnaire de flux de données sur les sites *Tmb*.

Caractéristiques du gestionnaire de flux de données			
Données échangées	Evénement		
Type de gestion	*First In First Out*		
Emplacement	$p \geq NbUnit$		
Opérateur	Type	Priorité	Nombre de Tâches
Mémorisation	asynchrone	Haute	1
Distribution	asynchrone	Basse	$1 \leq NbParEvent \leq NbUnit$

TAB. III.1 - *Caractéristique du gestionnaire de flux de données sur les sites* Tmb

III.5 La structure logicielle sur les sites *Unité de traitement*

Les fonctionnalités d'un site *Unité de traitement* sont:

1. de rassembler les informations nécessaires aux algorithmes de déclenchement selon un format prédéfini,
2. d'effectuer le traitement suivant le résultat du déclenchement de niveau–1, puis transmettre ce résultat aux processeurs de réduction de données du détecteur de traces (*Tec*) via le module *Tmb Driver*.
3. de transmettre l'événement au site *FT9000*.

Le cycle opératoire est composé de trois opérateurs correspondant aux trois fonctionnalités décrites précédemment. Ainsi on utilise de façon optimale les différentes ressources du *Transputer* permettant la mise en œuvre de parallélisme vrai sans augmenter le temps de latence de l'événement par du parallélisme concurrent. Les événements doivent subir les différentes opérations dans leur ordre d'arrivée sur le site. On n'autorise pas de parallélisme réplicatif au niveau des opérateurs car ceci induit du parallélisme concurrent et de plus cette opportunité est réalisée en augmentant le nombre d'unités de traitement. On définit chaque opérateur comme un opérateur de type *asynchrone* auquel est associé une seule tâche *Opérateur*.

III.5.1 L'opérateur de rassemblement des partitions

Il constitue le premier opérateur du cycle opératoire. L'opération consiste:

- à demander un événement via le site *Serveur* dès qu'il dispose d'un emplacement,
- à recevoir en paralléle les partitions réparties sur les différents sites *Tmb*s et vérifier la cohérence de l'événement.

L'événement est ensuite disponible pour l'opérateur de traitement.

Pour permettre la réception en parallèle des partitions, on définit une liste de canaux de communication profonde du nombre de partitions constituant un événement. On définit ensuite une liste de tâches de même profondeur pour permettre la réception en parallèle des partitions. Cette liste de tâches est utilisée comme une liste de tâches *synchrones* par la tâche *Opérateur*.

III.5.2 L'opérateur de traitement de l'événement

Il constitue le deuxième opérateur du cycle opératoire. L'opération de traitement consiste à appliquer un algorithme à un événement suivant le résultat du système de déclenchement de niveau–1. Une fois le traitement effectué, il transmet le numéro de l'événement et le résultat (accepté ou rejeté) au site *Tmb Driver*. L'événement est alors disponible pour l'opérateur de distribution.

III.5.3 L'opérateur de distribution de l'événement

Il constitue le dernier opérateur du cycle opératoire. L'opération de distribution consiste à transmettre l'événement au site *FT9000*.

On souhaite libérer le plus rapidement possible l'emplacement mémoire relatif à l'événement, on autorise donc le partage d'un même événement en partitions de taille égale que l'on transmet en parallèle. On définit une liste de canaux de communication avec le site *FT9000* de profondeur le nombre de partitions à transmettre. A cette liste, on associe une liste de tâches de même profondeur permettant le transfert en parallèle. Cette liste de tâche est utilisée comme une liste de tâches *synchrones* par la tâche *Opérateur*.

III.5.4 La structure logicielle

La figure III.6 présente l'organisation logicielle sur un site *Unité de traitement*. Pour obtenir

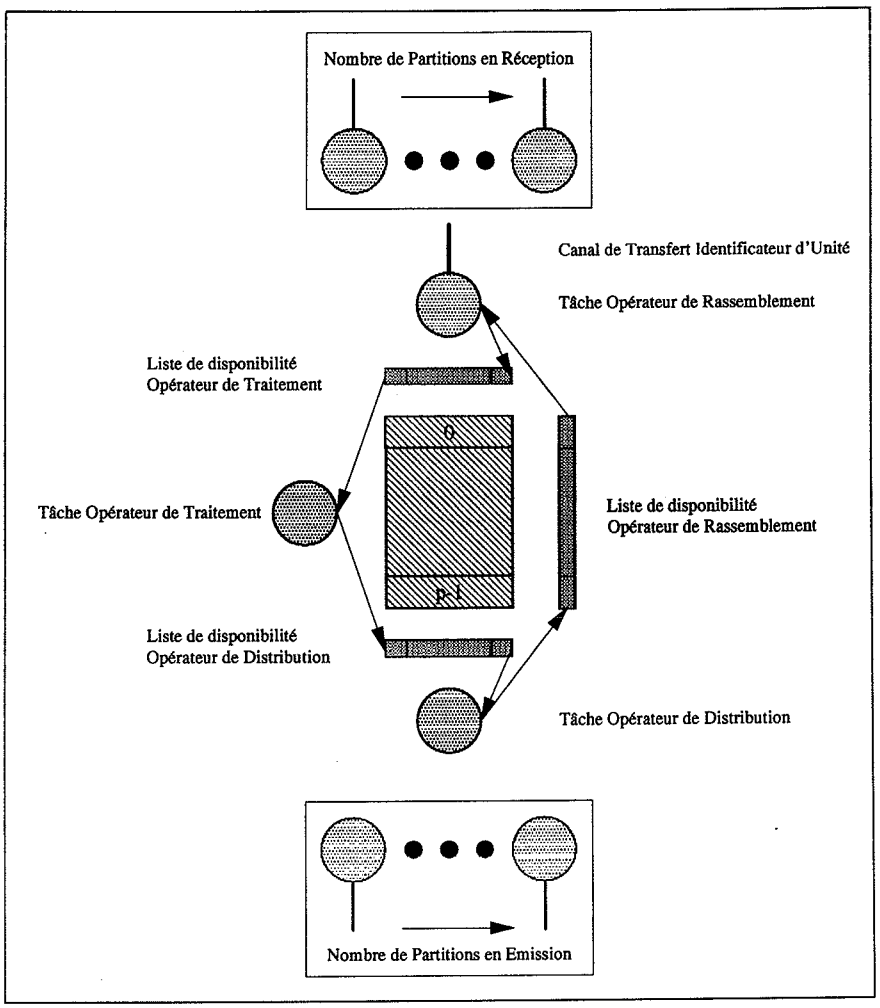


FIG. III.6 - *Structure logicielle sur un site* Unité de traitement

un temps de latence minimum pour chaque événement, il est souhaitable qu'une fois l'événement assemblé, l'opérateur de traitement devienne l'opérateur courant et ne soit pas suspendu par un autre opérateur. L'opérateur de traitement est un opérateur *haute priorité*. Ainsi l'opérateur de rassemblement des partitions ne pourra demander un nouvel événement seulement lorsque le traitement sera achevé. Un espace mémoire-tampon comprenant au minimum 2 événements permet d'assembler un nouvel événement et de tranférer celui qui vient d'être traité vers le site *FT9000* en *parallèle*.

Le tableau III.2 résume les caractéristiques du gestionnaire de flux de données sur les sites *Unité de traitement*.

III.6 La structure logicielle sur un site *Serveur*

Les fonctionnalités d'un site *Serveur* sont:

1. de sérialiser les demandes des unités de traitement,
2. de diffuser en parallèle ces demandes sur l'ensemble des sites *Tmb*,

Caractéristiques du gestionnaire de flux de données			
Données échangées	Evénement		
Type de gestion	*First In First Out*		
Emplacement	$p \geq 2$		
Opérateur	Type	Priorité	Nombre de Tâches
Rassemblement	asynchrone	Basse	1
Traitement	asynchrone	Haute	1
Distribution	asynchrone	Basse	1

TAB. III.2 - *Caractéristique du gestionnaire de flux de données sur les sites* Unité de traitement

3. de diffuser dans le même ordre que la diffusion sur les sites *Tmb*, les identificateurs des unités de traitement au site *FT9000*.

On utilise pour réaliser ce serveur le logiciel *Mbm* qui permet de rendre les fonctions énumérées précédemment asynchrones. On construit un cycle opératoire composé de trois opérateurs *basse priorité*

III.6.1 L'opérateur de sérialisation des demandes

Il constitue le premier opérateur du cycle opératoire auquel on associe autant de tâche *Opérateur* qu'il y a d'unités de traitement dans la ferme. A chaque tâche, on associe un canal de communication avec la tâche *Opérateur de rassemblement* de l'unité considérée. L'opération de sérialisation consiste à recevoir la demande d'un événement d'une unité dès qu'il dispose d'un emplacement. La sérialisation est effectuée lorsque l'opérateur libère l'emplacement lors de l'accès à la liste de disponibilité.

III.6.2 L'opérateur de diffusion *Tmb*

Il constitue le deuxième opérateur du cycle opératoire. C'est un opérateur *asynchrone* auquel on associe une seule tâche *Opérateur*. L'opération de diffusion *Tmb* consiste à transmettre en parallèle l'identificateur d'unité aux sites *Tmb*. L'identificateur est ensuite disponible pour l'opérateur de diffusion *FT9000*.

On définit une liste de canaux de communication profonde du nombre de sites *Tmb* à servir. A cette liste de canaux, on associe une liste de tâches de même profondeur pour le transfert en parallèle de l'identificateur. Cette liste de tâches est utilisée comme une liste de tâches *synchrones* par la tâche Opérateur.

III.6.3 L'opérateur de diffusion *FT9000*

Il constitue le troisième et dernier opérateur du cycle opératoire. L'opérateur de diffusion *Tmb* ne possédant qu'une seule tâche l'ordre des identificateurs est respecté, cet opérateur peut donc être un opérateur *asynchrone* auquel on associe une seule tâche *Opérateur*. L'opération de diffusion *FT9000* consiste à transmettre l'identificateur d'unité au site *FT9000* via un canal de communication.

III.6.4 La structure logicielle

La figure III.7 présente l'organisation logicielle sur un site *Serveur*.

Le tableau III.7 résume les caractéristiques du gestionnaire de flux de données sur le site *Serveur*.

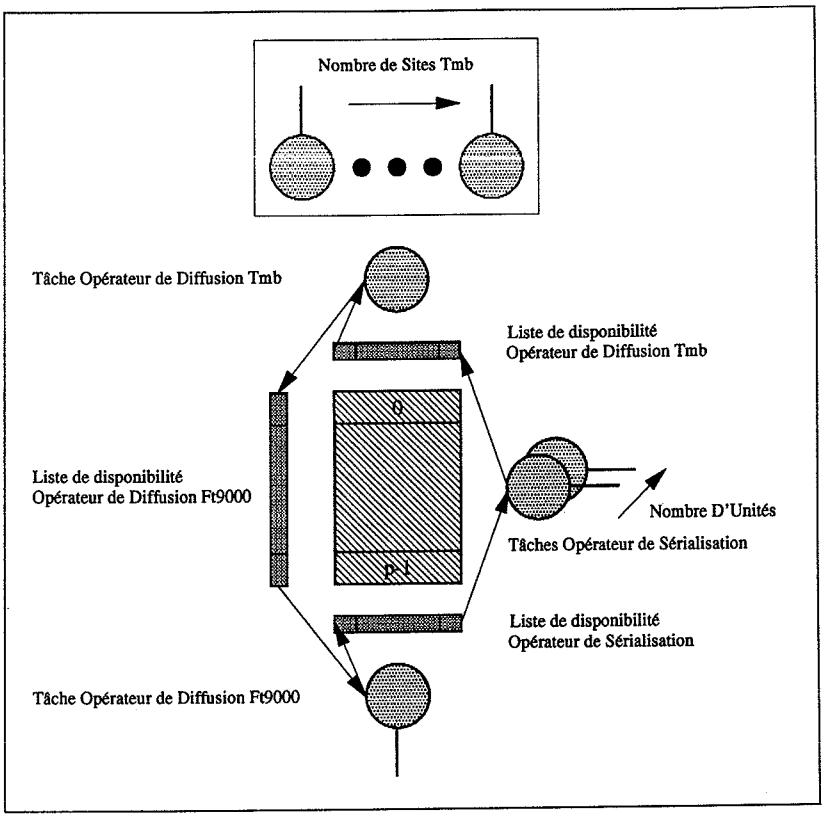


FIG. III.7 - *Structure logicielle sur un site* Serveur

III.7 La structure logicielle sur le site *FT9000*

Les fonctionnalités du site *FT9000* sont:

1. de recevoir les événements provenant des différentes unités de traitement,
2. d'envoyer les événements au système d'assemblage central dans l'ordre de leur arrivée sur les sites *Tmb* via la mémoire partagée *Fastbus–Transputer*.

On construit un cycle opératoire composé de deux opérateurs correspondant aux fonctionnalités décrites précédemment. Ces deux opérateurs sont des opérateurs *Basse priorité*. Le réarrangement des événements est réparti sur les deux opérateurs.

Caractéristiques du gestionnaire de flux de données			
Données échangées	Identificateur		
Type de gestion	*First In First Out*		
Emplacement	$p \geq NbUnit$		
Opérateur	Type	Priorité	Nombre de Tâches
Sérialisation	asynchrone	Basse	$NbUnit$
Diffusion *Tmb*	asynchrone	Basse	1
Diffusion *FT9000*	asynchrone	Basse	1

TAB. III.3 - *Caractéristique du gestionnaire de flux de données sur le site* Serveur

III.7.1 L'opérateur de rassemblement des événements

Cet opérateur constitue le premier opérateur du cycle opératoire. L'opération consiste à:

- mémoriser la référence de l'emplacement,
- recevoir l'identificateur de l'unité de traitement sur laquelle se situe le prochain événement à rassembler suivant l'ordre d'entrée des événements sur les sites *Tmbs*,
- à recevoir en parallèle les partitions constituant cet événement.

On souhaite pouvoir recevoir en paralléle les événements provenant des unités de traitement. Pour cela on associe à cet opérateur autant de tâches *Opérateur* qu'il y a d'unités de traitement dans la ferme. Pour que la condition [réarrangement:2] soit réalisée, il s'ensuit que la mémorisation de la référence de l'emplacement et la récuperation de l'identificateur de l'unité constitue une *zone critique* pour les tâches *Opérateur*.

A chaque tâche *Opérateur*, on associe une liste de tâches *Partition* profonde du nombre de partitions constituant l'événement. Cette liste est utilisée comme une liste de tâches *synchrones* par la tâche *Opérateur*. Ces tâches *Partition* doivent pouvoir s'interconnecter avec l'ensemble des unités constituant la ferme de traitement. On définit une matrice d'interconnexion contenant les canaux de communication du site *FT9000* avec les sites *Unité de traitement* dont l'élément de base est une liste de canaux profonde du nombre de partitions à recevoir provenant d'une unité de traitement. La réception de l'identificateur d'unité de traitement par une tâche *Opérateur* permet de sélectionner la liste de canaux adéquate et ensuite d'associer à chaque tâche *Partition* un canal pour permettre la réception en parallèle des différentes partitions.

Chaque liste de la matrice d'interconnexion est une ressource partagée par les tâches *Opérateur* dont l'accès est géré par un sémaphore binaire. De même le canal de réception de l'identificateur d'unité de traitement est une ressource partagée par les tâches *Opérateur* dont l'accès est géré par un sémaphore binaire. L'accès à la zone critique est géré par le sémaphore binaire associé au canal de réception de l'identificateur d'unité de traitement.

III.7.2 L'opérateur de transfert dans la mémoire *Fastbust–Transputer*

Il constitue le dernier opérateur du cycle opératoire. L'opération de transfert consiste à sélectionner la référence des emplacements dans l'ordre de leur association avec les identificateurs d'unité et à transférer l'événement dans la mémoire *Fastbust–Transputer* lorsque le maître *Fastbus* autorise l'accés à cette mémoire pour le *Transputer*. Pour que cet opérateur traite les événements dans le même ordre que l'opérateur de rassemblement, il faut que cet opérateur soit un opérateur *synchrone*. On associe une seule tâche à l'opérateur puisque le transfert s'effectue par l'intermédiaire du bus de données du *Transputer*.

III.7.3 La structure logicielle

La figure III.8 présente l'organisation logicielle sur le site *FT9000*.

Le tableau III.4 résume les caractéristiques du gestionnaire de flux de données sur le site *FT9000*.

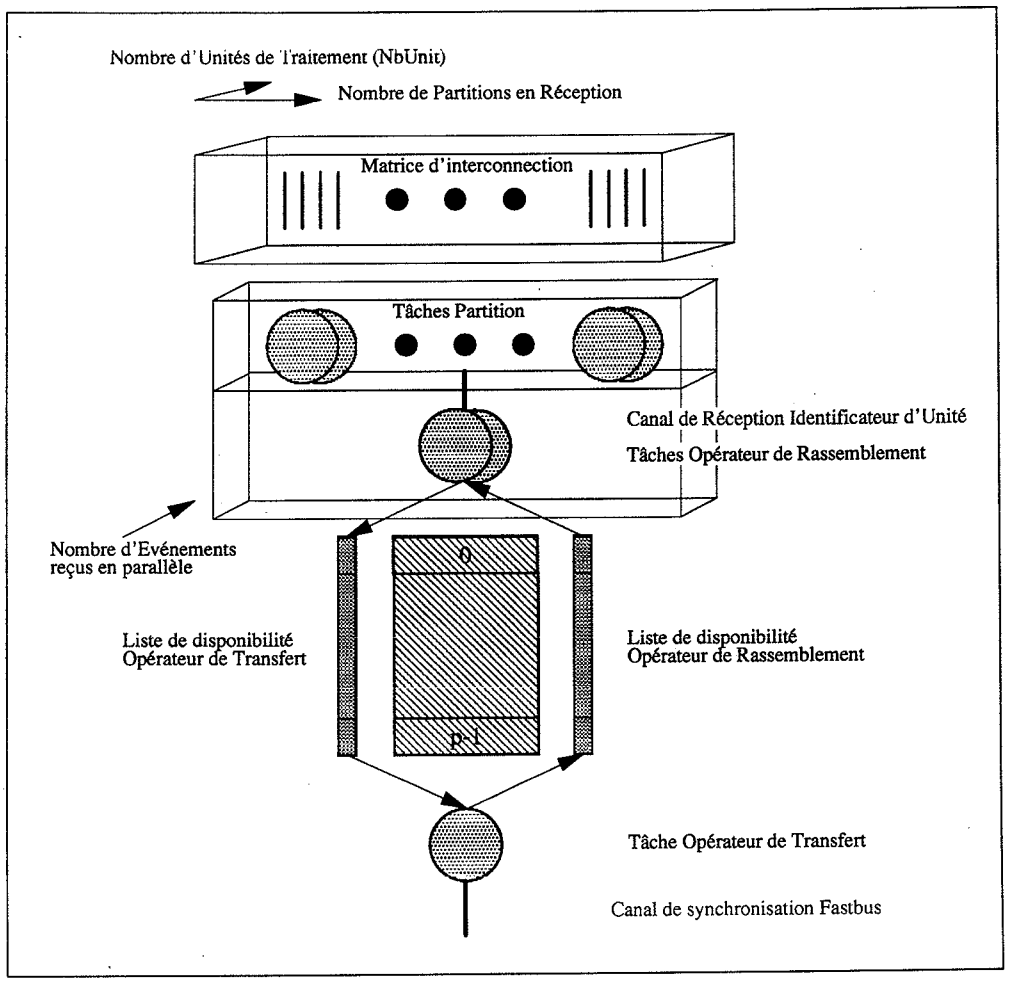


FIG. III.8 - *Structure logicielle sur du site* FT9000.

Caractéristiques du gestionnaire de flux de données			
Données échangées	Evénement		
Type de gestion	*First In First Out*		
Emplacement	$p \geq NbUnit$		
Opérateur	Type	Priorité	Nombre de Tâches
Rassemblement	asynchrone	Basse	$1 \leq NbParEvent \leq NbUnit$
Transfert	synchrone	Basse	1

TAB. III.4 - *Caractéristique du gestionnaire de flux de données sur le site* FT9000

III.8 Le parallélisme de traitement

Le traitement consiste à appliquer des algorithmes spécifiques suivant le résultat du déclenchement de niveau–1. Il valide l'événement s'il est à déclenchement multiple, ou applique l'algorithme suivant à la nature du déclenchement lors d'un déclenchement unique.

Le logiciel d'assemblage et de traitement a été conçu pour permettre le parallélisme. On envisage:

- de distribuer les algorithmes de sélection sur des *Transputer* distincts pour lesquels on assemble uniquement les données qui leurs sont nécessaires,
- d'appliquer les algorithmes relatifs au type de déclenchement à chaque événement sans augmenter le temps de latence.

On définit:

- une *unité de traitement* comme l'ensemble des algorithmes de sélection,
- une *entité de traitement* comme un élément d'une *unité de traitement* correspondant à un algorithme.

III.8.1 L'assemblage de l'événement avec parallélisme de traitement

Pour utiliser de façon optimale la bande passante de l'étage de traitement, le traitement d'un événement peut être distribué sur différentes unités selon la disponibilité de leurs entités. Le tableau III.5 donne un exemple de répartion d'événements sur un étage traitement constitué de quatres unités composées de deux entités.

événement	unité[0]	unité[1]	unité[2]	unité[3]
entité[0]	p	p+1	p+2	p+3
entité[1]	p+3	p+2	p+1	p

TAB. III.5 - *Exemple de répartition des événements sur un étage de traitement constitué de 4 unités contenant 2 entités.*

L'assemblage de l'événement s'effectue alors en deux étapes:

- assemblage partiel de l'événement sur chaque entité: assemblage *Tmb–Entité*.
- assemblage complet de l'événement réparti sur les différentes entités: assemblage *Entité–FT9000*.

Chaque entité travaille de façon autonome et récupère uniquement les données nécessaires à son traitement. Chaque site *Tmb* doit être capable de distribuer les données nécessaires aux différentes entités qu'il doit servir, ces données pouvant être différentes ou dupliquées si nécessaire (cas de l'en–tête de l'événement). De plus chaque site *Tmb* peut distribuer en *parallèle* les données relatives à un même événement vers les différentes entités qu'il doit servir. La figure III.9 présente la structure d'assemblage d'événement avec parallélisme de traitement.

Pour respecter les conditions [cohérence:1] et [réarrangement:1] des événements, on associe à chaque type d'entité un serveur qui sérialise les demandes des différentes unités, les diffuse aux sites *Tmb* associés à l'entité, puis au site *FT9000*.

La condition [cohérence:2] est respectée, si sur chaque site *Tmb* la réception des différents identificateurs d'unité de traitement relatifs aux entités à servir s'effectue dans la *zone critique* de l'opérateur de distribution des partitions.

La condition [réarrangement:2] est respectée, si sur le site *FT9000* la réception des différents identificateurs d'unité de traitement relatifs au prochain événement à assembler est effectué dans la *zone critique* de l'opérateur de rassemblement d'événements.

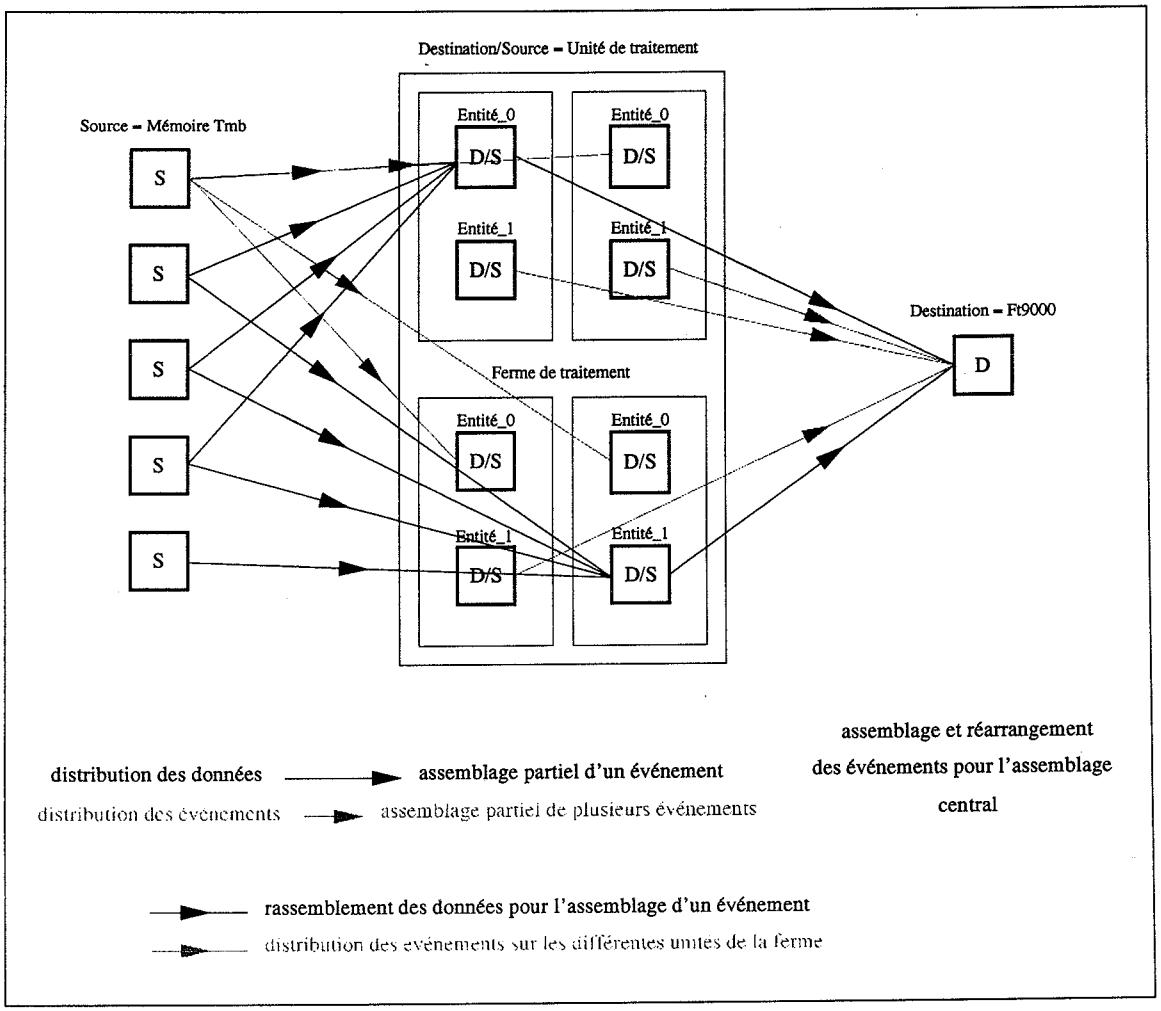


FIG. III.9 - *Structure d'assemblage avec parallélisme de traitement.*

III.8.2 La structure logicielle sur les sites *Tmb* et *FT9000*

Seule la structure de l'opérateur de distribution des partitions sur les sites *Tmb* et celle de l'opérateur de rassemblement sur le site *FT9000* sont étendues pour permettre le parallélisme de traitement.

La structure pour une *entité de traitement* est identique à celle d'une *unité de traitement* lors d'un assemblage sans parallélisme de traitement.

La structure d'un site *Serveur* est identique quel que soit le type d'assemblage, avec ou sans traitement parallèle.

L'opérateur de distribution des partitions du site *Tmb*

Sur chaque site *tmb*, une liste de canaux profonde du nombre d'entités à servir permet la réception des identificateurs d'unités de traitement. Cette liste de canaux est une ressource partagée par les tâches *Opérateur*. L'accés à cette liste est géré par un sémaphore binaire. L'association {*identificateurs d'unité*}–*événement* est effectuée dans la zone critique dont l'accés est géré par le sémaphore binaire associé aux canaux de réception des identificateurs d'unités de traitement.

A chaque tâche *Opérateur*, on associe une liste de tâches *synchrones* profonde du nombre d'entités de traitement à servir par le site *Tmb* considéré. Ces tâches sont nommées tâches *Entité*. On associe ensuite à chaque tâche *Entité* une matrice *Tâche* qui permet l'interconnexion avec l'ensemble des unités de traitement. La sélection d'un élément de la matrice *Tâche* est effectuée par l'identificateur d'unité transmis par le serveur d'entité. La figure III.10 présente la structure logicielle sur un site *Tmb* permettant le parallélisme de traitement.

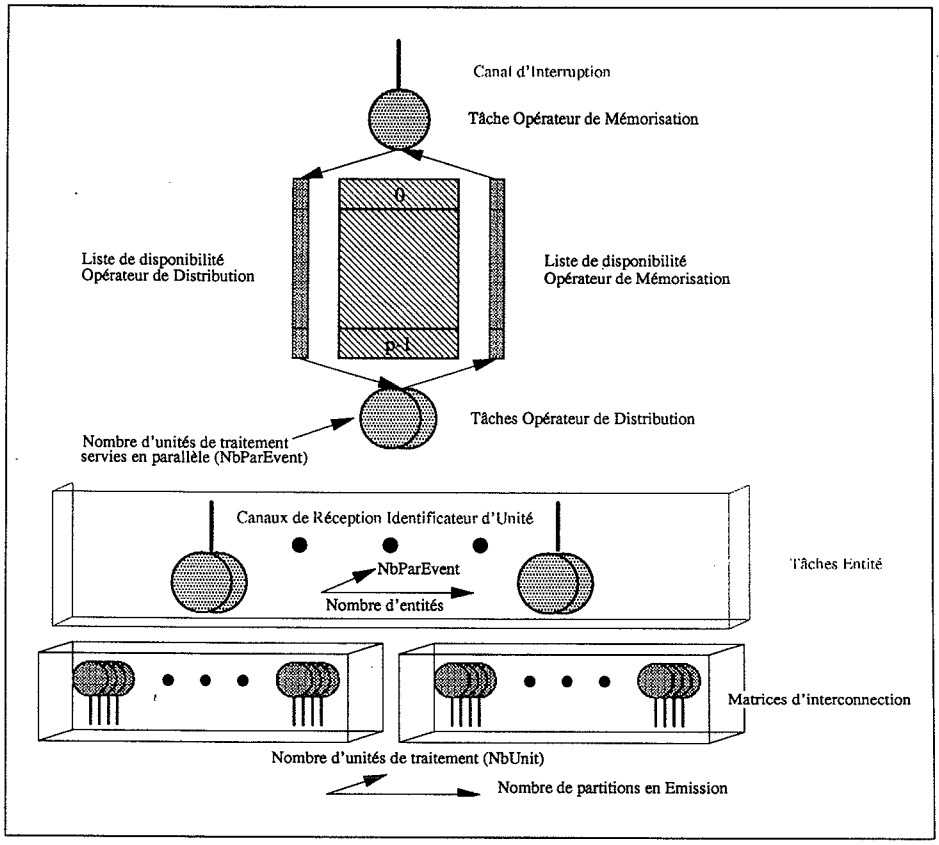


FIG. III.10 - *Structure logicielle sur un site* Tmb *permettant le parallélisme de traitement.*

L'opérateur de rassemblement du site *FT9000*

Sur le site *FT90000*, une liste de canaux profonde du nombre d'entité constituant une *unité de traitement* permet la réception des identificateurs d'unité où sont localisées les différentes entitées constituant l'unité de traitement pour le prochain événement à recevoir. Cette liste constitue une ressource partagée par les tâches *Opérateur*. L'accès à cette liste est géré par un sémaphore binaire.

Pour chaque type d'entité de traitement, on définit une matrice d'interconnexion qui permet la réception des partitions provenant des unités de traitement. Les identificateurs d'unités de traitement permettent d'affecter les canaux des matrices d'interconnexion aux tâches *Partition*. La figure III.11 présente la structure logicielle sur le site *FT9000* permettant le parallélisme de traitement.

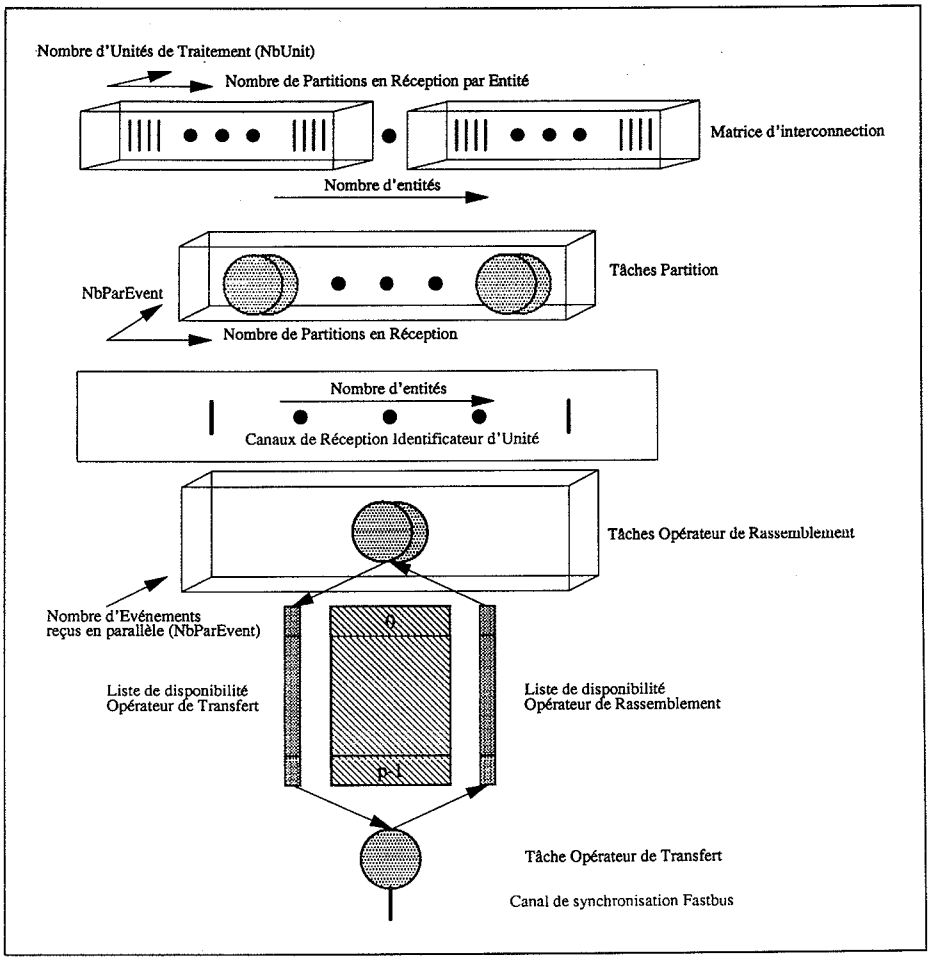


FIG. III.11 - *Structure logicielle sur un site* ft9000 *permettant le parallélisme de traitement.*

La structure logicielle complète du système d'assemblage est représentée sur la figure III.12.

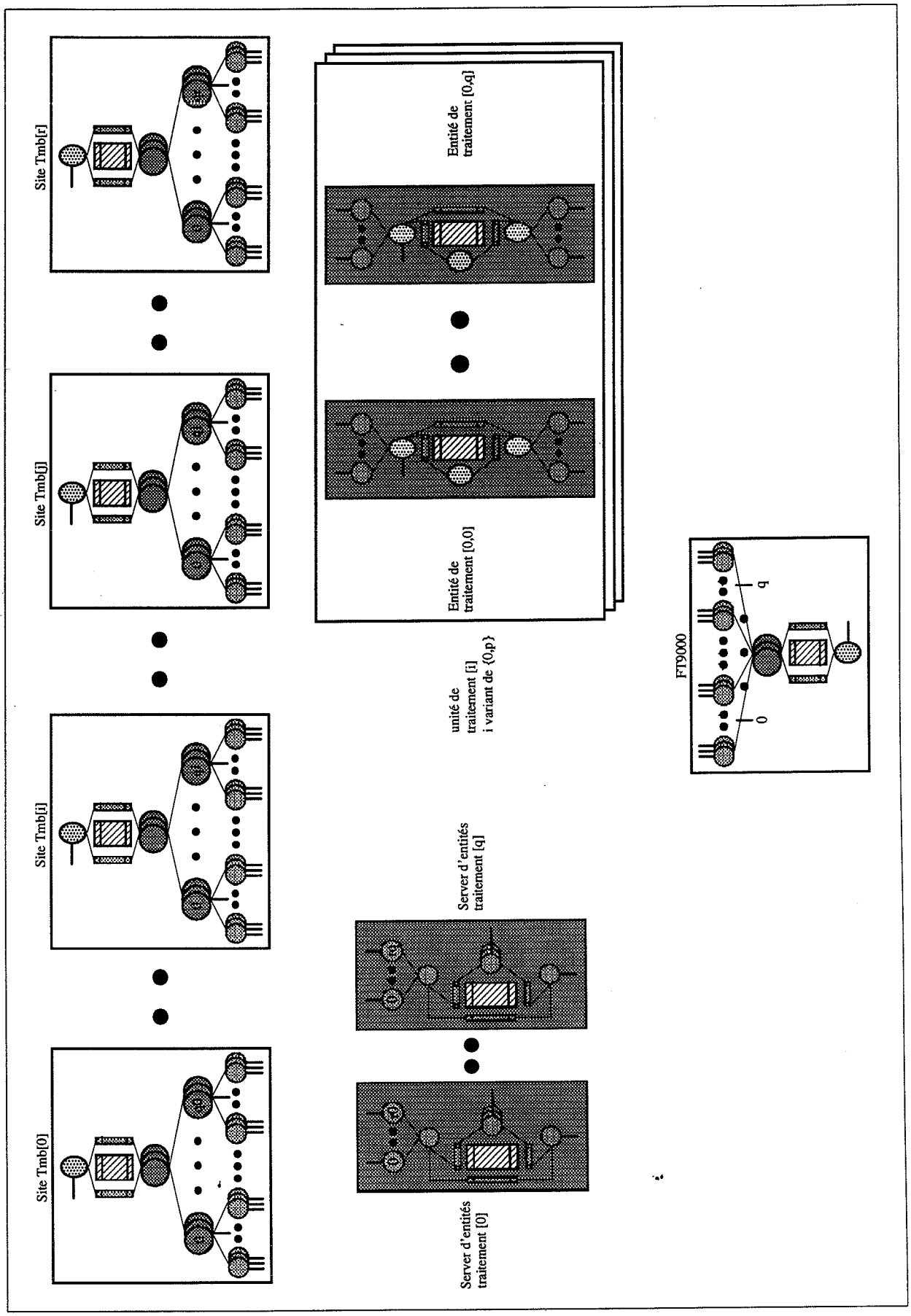


FIG. III.12 - *Structure logicielle du système d'assemblage permettant le parallélisme de traitement.*

Chapitre IV

Performance du système de déclenchement de niveau–2

IV.1 Introduction

Ce chapitre a pour objectifs d'établir les performances du systèmes en insistant sur les aspects suivants:

- montrer que le trafic sur le réseau de communication n'est pas perturbé par les logiciels mis en œuvre en vérifiant que la vitesse mesurée sur les liens est proche de la vitesse limite attendue.
- montrer que le système n'introduit pas de temps mort additionnel dans l'expérience en vérifiant que le temps mort par événement, appliqué à L3, ne dépend pas du taux de déclenchement.
- étudier le comportement du système en fonction du taux de déclenchement, du nombre de blocs à assembler et de leur taille.
- mesurer le temps d'initialisation logiciel qui est le paramètre le plus important dans l'assemblage de blocs de petite taille.
- mesurer la bande passante du système en nombre d'événements pour vérifier le cahier des charges et établir les limites du système.

Dans toutes les mesures qui suivent, le code utilisé est celui utilisé dans l'expérience L3, seuls les paramètres décrivant les données (nombre de blocs, longueur) ou controlant les fonctions (assemblage, traitement, transfert vers la mémoire de sortie) ont été modifiés

IV.2 Conditions de mesures et définition des temps mesurés

IV.2.1 Conditions de mesures

Au moment de l'installation, nous disposons de 32 *Transputers Revision D* et seulement 3 *Transputers Révision E*. Le *Transputer Révision E* est une version sans défauts majeurs pour le système de déclenchement de niveau–2 par oppposition aux *Transputers Revision D* dont les principaux défauts sont:

- Possibilité de corrompre les données lorsqu'elles sont reçues à travers au moins deux liens et que leur destination se situe dans un espace mémoire accessible en mode *cache*[Bug INSdi04069].
- Si l'on a 2 ou plusieurs canaux virtuels multiplexés sur un même lien physique, le processeur de canaux virtuels peut ne pas réactiver correctement les tâches communiquant par ces canaux [Bug INSdi04059].

L'ensemble des défauts d'un *Transputers Revision D* sont énumérés dans [20].

Pour l'ensemble des mesures, nous utilisons le réseau de *Transputers* présenté à la figure II.6 en précisant que:

- les *Transputers* dédicacés au contrôle des parties *Acquisition* et *Injection* sont des *Transputers Révision D*. On rappelle que 6 *Transputers* d'*Acquisition* ont un lien connecté sur le premier *C104*, les 6 autres sur le deuxième.

  La distribution des partitions et des événements s'effectue donc en *série* : on autorise qu'un seul canal virtuel à occuper un lien à chaque instant.
- la carte de traitement ne dispose actuellement que d'une seule carte *HTRAM* dont le *Transputer Révision E* est utilisé pour le multiplexage des entrées/sorties provenant des autres *Transputers*.

- la ferme de traitement est constituée par deux *Transputers Révision E* situés sur une carte *Tmb* ayant chacun un lien de connecté vers chaque *C104*.

  La réception des partitions s'effectue en *paralléle*.

  Lors de la transmission de l'événement vers le module *FT9000*, l'événement est divisé en deux blocs égaux transmis en *série*.

- le *Transputer* dédicacé au module *FT9000* est un *Transputers Révision D*. Il a un lien connecté au premier *C104*,

- les *Transputers* travaillent à $20MHz$, les *C104* à $30MHz$.

- les liens de données sont programmés à la vitesse de 100 Mbits s^{-1}.

IV.2.2 Définition des temps sur les différents sites

On définit les temps suivants correspondants aux différentes opérations s'effectuant sur les différents sites:

Tmb :

1. Mémorisation des données:
   - T_{tmb_M}: Temps de mémorisation des données,
     Début : Réception *Lv1*.
     Fin : lorsque l'emplacement est disponible pour l'opérateur de distribution.
2. Distribution des partitions:
   - T_{tmb_psh}: temps pour préparer et émettre les en–têtes.
   - T_{tmb_psd}: temps pour préparer et émettre les données.
   - $T_{tmb\_pshd} = T_{tmb\_psh} + T_{tmb\_psd}$: temps global de l'opération de distribution,
     Début : réception de l'identificateur d'unité de traitement.
     Fin : lorsque l'emplacement est à nouveau disponible pour l'opérateur de mémorisation.

Unité de traitement :

1. Rassemblement des partitions:
   - T_{entity_prh}: comprend l'initialisation et la réception des en–têtes,
     Début : Réception de l'identificateur d'unité de traitement.
     Fin : lorsque l'ensemble des en–têtes ont été reçus.
   - T_{entity_prd}: comprend l'initialisation, la réception des données et la synchronisation avec l'opérateur suivant,
     Début : Après réception de l'ensemble des en–têtes.
     Fin : lorsque l'emplacement est disponible pour l'opérateur suivant.
   - $T_{entity\_prhd} = T_{entity\_prh} + T_{entity\_prd}$: temps global de l'opération de rassemblement
2. Traitement de l'événement:
   - T_{entity_T}: correspond au temps de traitement,
     Début : Dès que l'événement est disponible pour l'opération de traitement.
     Fin : lorsque l'emplacement est disponible pour l'opérateur suivant.
3. Distribution de l'événement:
   - T_{entity_psh}: temps pour préparer et émettre les en–têtes.

- T_{entity_psd}: temps pour préparer et émettre les données.
- $T_{entity\_pshd} = T_{entity\_psh} + T_{entity\_psh}$: correspond au temps pour transmettre l'événement au site *FT9000*,

  Début : Dès que l'événement est disponible pour l'opération de distribution.

  Fin : lorsque l'emplacement est disponible pour l'opérateur suivant.

Serveur :

1. Sérialisation des demandes:
   - T_{server_s}: comprend la réception et la sérialisation de la demande.
2. Diffusion *Tmb*:
   - T_{server_dtmb}: comprend la diffusion aux sites *Tmb*s, la réception des acquittements émis par chaque site *Tmb*.
3. Diffusion *FT9000*:
   - $T_{server\_dft9000}$: comprend la diffusion au site *FT9000*, la réception de l'acquittement émis par le site *FT9000*.
4. Seveur:
   - T_{server}: correspond au temps durant lequel l'unité de traitement n'est pas disponible.

     Début : lorsque l'ensemble des sites *Tmb*s ont transmis leurs acquittements indiquant qu'ils disposent tous d'un événement à assembler.

     Fin : lorsque l'unité de traitement émet une nouvelle demande d'événement.

FT9000 :

1. Rassemblement des évévements:
   - T_{ft9000_prh}: comprend l'initialisation et la réception des en–têtes.
   - T_{ft9000_prd} comprend l'initialisation, la réception des données et la synchronisation avec l'opérateur suivant.
   - $T_{ft9000\_prdh} = T_{ft9000\_prh} + T_{ft9000\_prd}$: correspond au temps pour recevoir et ordonner les événements,

     Début : lorsque l'identificateur d'unité de traitement du prochain événement à receptionner est reçu.

     Fin : lorsque l'emplacement est disponible pour l'opérateur suivant.
2. Transfert *Transputer–FASTBUS*:
   - T_{ft9000_T2F}: correspond au temps pendant lequel l'événement est présent sur l'opérateur de transfert dans la mémoire *Transputer–FASTBUS*. Il comprend l'attente de la synchronisation avec l'assemblage central.

     Début : lorsque l'événement est disponible pour l'assemblage central.

     Fin : lorsque l'emplacement est disponible pour l'opérateur suivant.

IV.3 Caractérisation de système d'assemblage complet: *Tmb–Entité–Ft9000*

IV.3.1 Les conditions de mesure et Mesures

Pour caractériser le système d'assemblage complet, on varie le nombre de partitions transmis par chaque source et la taille des partitions. Chacunes des sources émet le même nombre de partitions, les partitions ayant la même taille. On se place à une fréquence d'injection pour laquelle on s'assure que le système fonctionne à vide (mémoires d'entrée vides).

Les sources sont constituées par les mémoires *Tmb* qui transmettent 1, 2, 3, 4 partitions dont la taille prend les valeurs 2, 256, 512, 1024 octets. Pour chaque point on relève:

- le temps d'assemblage de l'événement donné par T_{server}.
- le temps d'assemblage des données T_{entity_prd}.
- le temps de transfert de l'événement donné par le minimum entre T_{entity_prhd} et T_{ft9000_prhd}.

La table IV.1 donne la configuration logicielle utilisée sur les différents sites.

Site	Opérateur	Nb Tâches	Précision
12 *Tmbs*	Mémorisation	1	
	Distribution	1	Transmission des partitions en *série*
	16 emplacements situé dans la *SRAM 32 bits non–cache* Code situé dans la *DRAM 32 bits cache 16 KOctets*		
1 *Unité*	Rassemblement	1	Réception des partitions en *parallèle*
	Distribution	1	Transmission des partitions en *série*
	2 emplacements situé dans la *DRAM 32 bits cache 16 KOctets* Code situé dans la *DRAM 32 bits cache 16 KOctets*		
1 *FT9000*	Rassemblement	1	
	64 emplacements situé dans la *DRAM 64 bits cache 16 KOctets* Code situé dans la *DRAM 64 bits cache 16 KOctets*		

TAB. IV.1 - *Conditions de mesure pour la caractérisation du système complet*

La figure IV.1 présente:

- le temps d'assemblage en fonction de la taille de l'événement assemblé et du nombre de partitions.
- le graphe du $Min(T_{ft9000\_prhd}, T_{entity\_pshd})$ en fonction de la taille de l'événement.

On remarque que:

- pour une taille d'événement supérieure à 10 KOctets, le temps d'assemblage est proportionnel à la taille et que les courbes relatives aux différents nombre de partitions (24, 36, 48) sont parallèles. On en déduit:
  - qu'il existe un temps d'assemblage logiciel proportionnel aux nombres de partitions assemblées.
  - que l'on a saturation du mécanisme de réception soit par blocage, soit par la vitesse limite des liens lorsque la taille de l'événement augmente.
- le temps de transfert vers le module *FT9000* est proportionnel à la taille de l'événement: on a donc saturation du mécanisme de transmission.

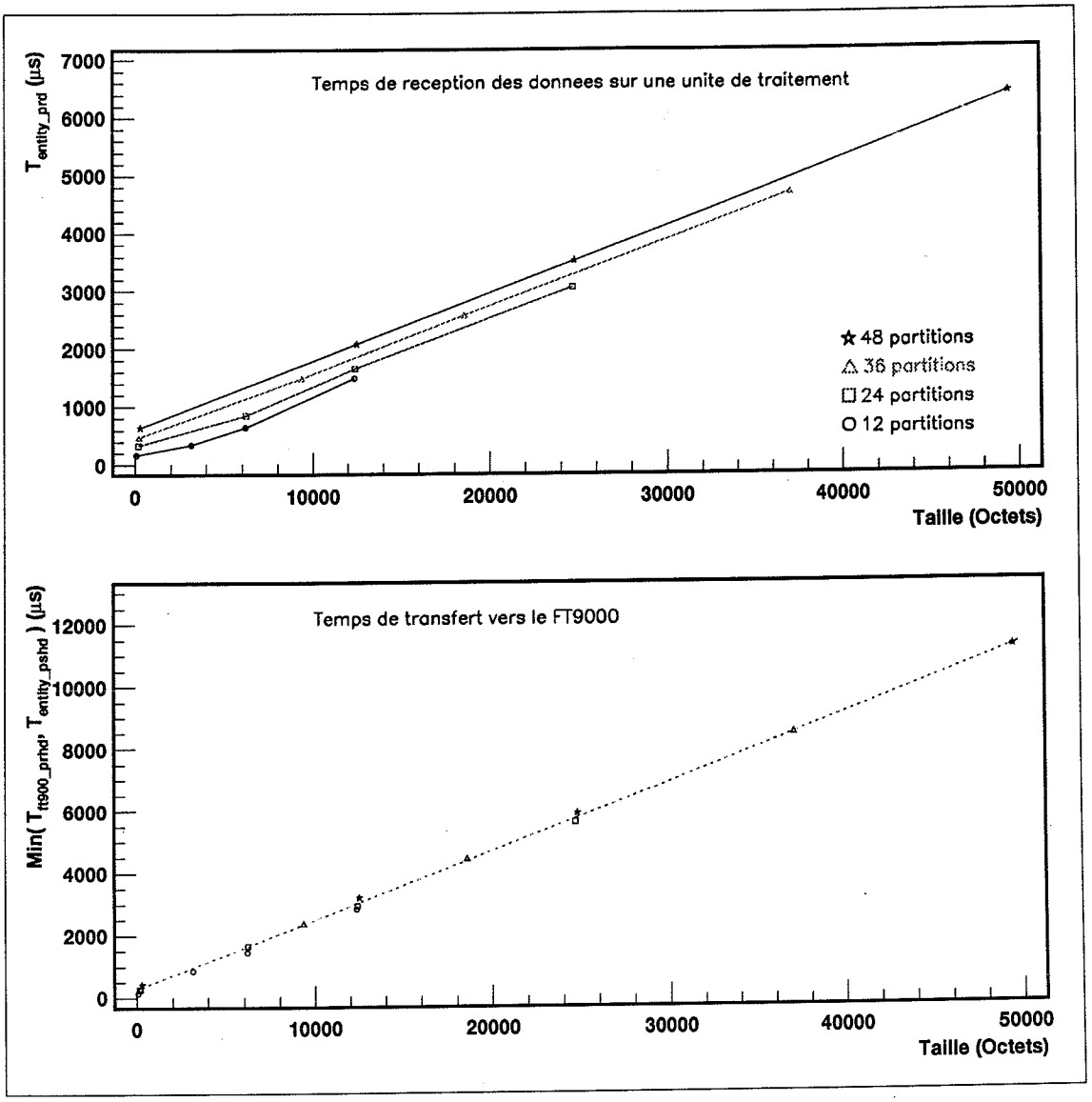


FIG. IV.1 - *Temps d'assemblage des données et Temps de transfert vers le* FT9000 *en fonction de la taille d'un événement*

IV.3.2 Temps logiciel d'assemblage

Lorsque les en-têtes ont été reçus par l'opérateur de rassemblement, celui-ci:

1. vérifie la cohérence de l'événement,
2. initialise la réception des données,
3. active la liste de tâches *Partition* avec l'instruction *ProcParList*.

pendant que sur chaque source:

1. on initialise les partitions à émettre,
2. on active la liste de tâches *Partition* avec l'instruction *ProcParList*.

Si on estime que les sources effectuent 12 fois moins d'opérations que les unités, elles sont en état de transmettre leurs premiers paquets pendant que l'unité de traitement finit ces opérations

d'initialisation. Les paquets sont alors disponibles sur l'unité de traitement lorsque les tâches *Partition* arrivent au rendez–vous. Le temps d'assemblage correspond alors uniquement à du temps logiciel pour des partitions dont la taille est inférieure à 32 octets.

La figure IV.2 reporte le temps d'assemblage en fonction du nombre de partitions lorque l'on transmet 2 octets par partition. On s'aperçoit que T_{entity_prd} varie linéairement en fonction

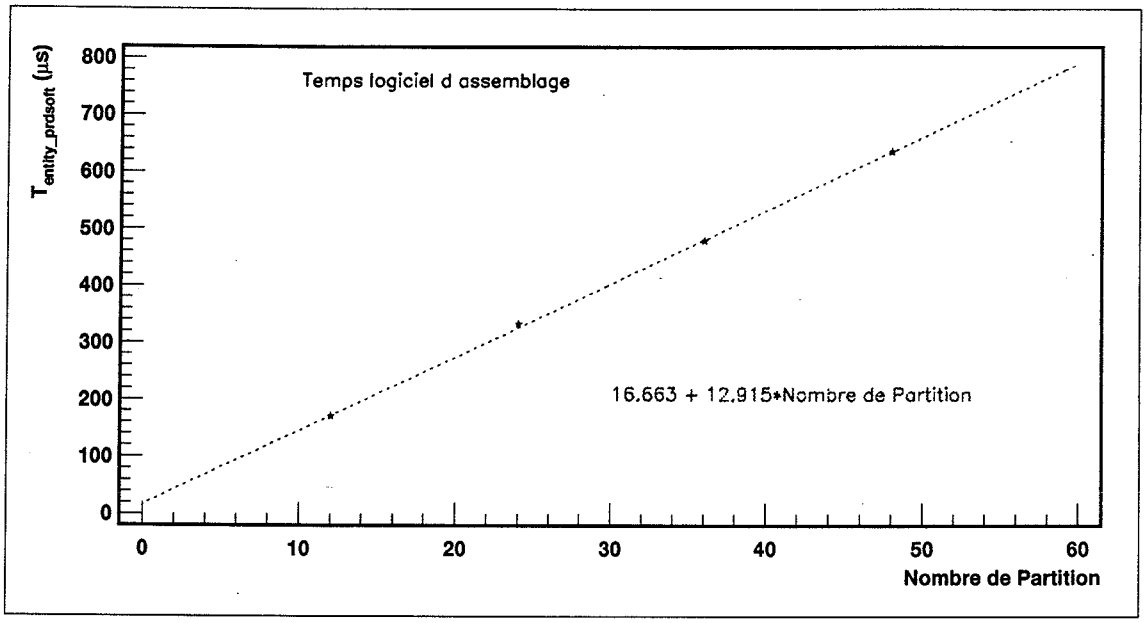


FIG. IV.2 - *Temps logiciel d'assemblage en fonction du nombre de partitions où chaque partition transmet 2 octets*

du nombre de partitions assemblées. On définit $T_{entity\_prdsoft}$ comme étant égale à T_{entity_prd} sans transfert sur les liens. T_{entity_prd} s'exprime alors

$$T_{entity\_prd} = T_{entity\_prdsoft} + T_{entity\_prdlink} \tag{IV.1}$$

où

$$T_{entity\_prdsoft} = C_{prdsoft} * NombredePartitions, \tag{IV.2}$$

avec $C_{prdsoft}$: constante d'initialisation et d'activation des partitions, et

$$T_{entity\_prdlink} = C_{link_2} * Taille, \tag{IV.3}$$

avec C_{link_2}: vitesse de réception des données sur 2 liens.

Expérimentalement, on trouve:

$$T_{entity\_prdsoft}(\mu s) = 16.7 + 12.92 * NombredePartitions \tag{IV.4}$$

Le décalage à l'origine dans l'expression de $T_{entity\_prdsoft}$ est du au fait que T_{entity_prd} contient la synchronisation avec l'opérateur suivant.

Lorsque l'on affine les tests, la constante $C_{prdsoft}$ s'exprime

$$C_{prdsoft} = C_{pdsoft} + C_{ProcParList}.$$

avec

- C_{pdsoft}: constante d'initialisation des partitions,

- $C_{ProcParList}$: constante de synchronisation entre la tâche *maître* gérant l'emplacement et les tâches *Partition* réalisée par la fonction *ProcParList*.

On en déduit alors:

$$C_{pdsoft} \approx 4.5\mu s/partition,$$

$$C_{ProcParList} \approx 8.5\mu s/partition.$$

IV.3.3 Bande passante de réception des données

La réception des données s'effectue à travers deux liens. Chaque lien de réception reçoit le même nombre de blocs. Le temps d'assemblage des données variant linéairement avec la taille de l'événement, la pente de la droite, C_{link_2}, correspond à la vitesse de réception des données sur les 2 liens. Elle permet de déduire la bande passante limite corespondant à la bande passante disponible sur ces 2 liens.
D'après les équations IV.1, IV.3, on déduit

$$C_{link_2} = 0.1165\mu s/octet$$

soit une bande passante limite sur les deux liens de réception de 8.58 Moctets s^{-1}. La figure IV.3 présente le temps d'assemblage des données T_{entity_prd}, son approximation par l'équation IV.5 et la bande passante corespondante.

Le temps de réception des données T_{entity_prd} sur deux liens s'exprime alors:

$$T_{entity\_prd}(\mu s) = 16.7 + 12.92 * NombredePartitions + 0.1165 * Taille(octets). \quad \text{(IV.5)}$$

La bande passante de 8.58 MOctets s^{-1} correspond à un transfert caractérisé par:

- une source où le code est situé dans une mémoire *DRAM 32 bits cache 16 KOctets* et les données dans une mémoire *SRAM 32 bits*.
- une destination où le code et les données se partage une mémoire *DRAM 32 bits cache 16 KOctets*.
- un transfert sur 2 liens physiques à travers un *C104* avec 6 canaux virtuels par lien phisyque.

D'après [10], pour un transfert entre deux mémoires externes 32 bits non–cache, la bande passante sur deux liens avec 10 canaux vituels approche 9.5 MOctets s^{-1}. La différence peut s'expliquer par la différence de configuration matérielle.

la figure IV.4 présente la bande passante d'une unité de traitement correspondant à la réception des en–têtes et des données. La réception des en–têtes altère légèrement la bande passante de réception des données pour une fréquence d'injection $\approx$ 50 Hz. Nous en expliquerons les raisons dans les paragraphes suivants.

IV.3.4 Bande passante *FT9000*

Le temps de transfert d'un événement sur le *FT9000* varie linéairement en fonction de la taille de l'événement, il s'exprime:

$$Min(T_{ft9000\_prhd}, T_{entity\_pshd}) = T_{init} + C_{link_1} * Taille \quad \text{(IV.6)}$$

avec:

- T_{init}: temps d'initialisation, d'activation et de synchronisation des tâches *Partition*,

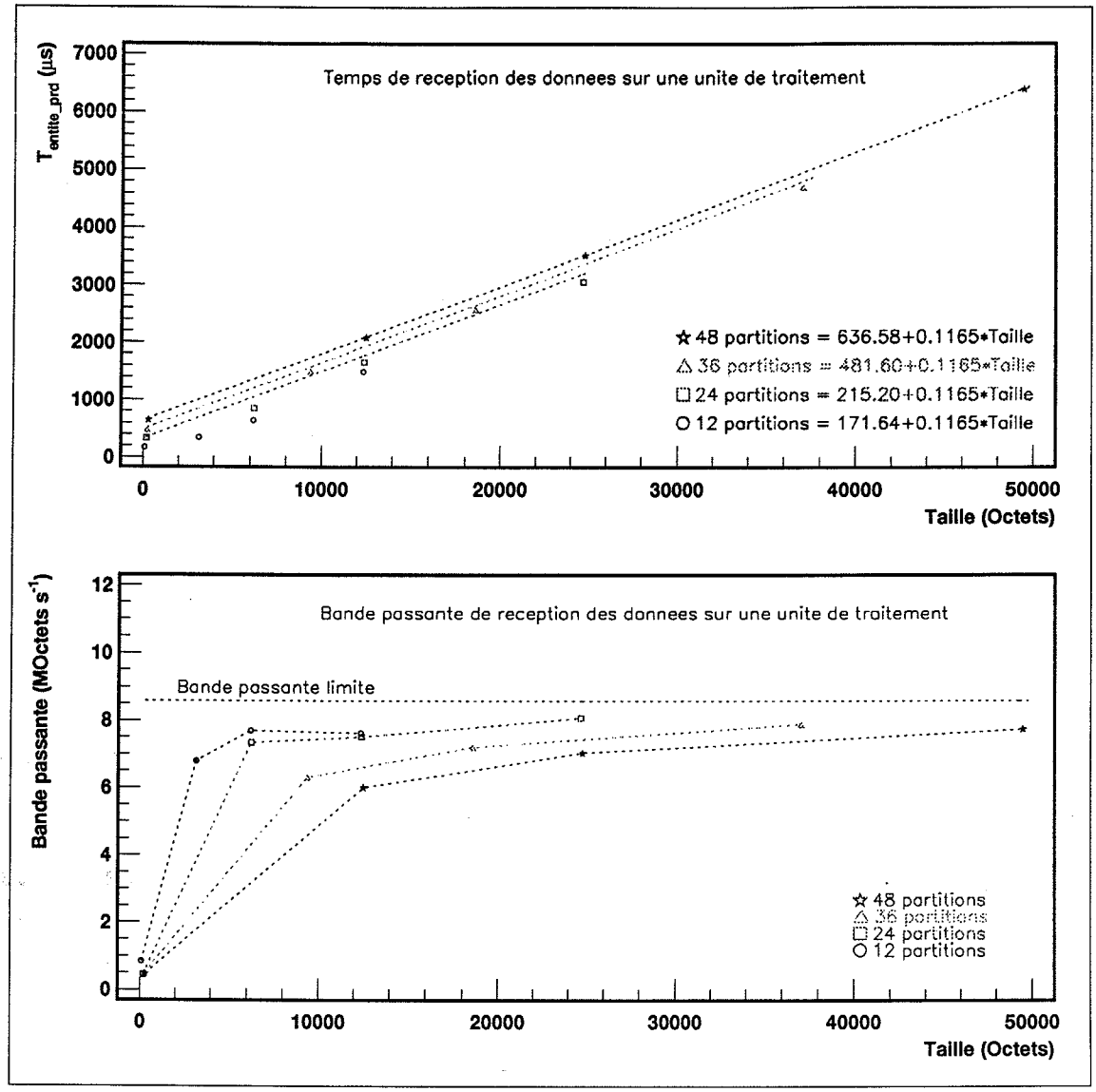


FIG. IV.3 - *Temps d'assemblage des données et bande passante correspondante*

- C_{link_1}: vitesse de réception des donnés sur un lien.

La figure IV.5 présente le temps de transfert et la bande passante correspondant.

La dispersion entre les points pour des tailles voisines n'est pas vraiment bien comprise. Il semblerait que le nombre de partitions à assembler ait une influence sur la bande passante, ceci pourrait se justifier par le fait que l'opération de rassemblement et l'opération de distribution sur le *FT9000* s'effectue en parallèle.
$Min(T_{ft9000\_prhd}, T_{entity\_pshd})$ peut être ajusté par:

$$Min(T_{ft9000\_prhd}, T_{entity\_pshd}(\mu s) = 180 + 0.225 * Taille(octet) \quad \text{(IV.7)}$$

Lors de transfert de très petite taille $Min(T_{ft9000\_prhd}, T_{entity\_pshd}) = T_{ft9000\_prhd}$, c'est à dire que les tâches *Partition* du *FT9000* se présentent les dernières au rendez–vous, le temps $T_{init_{ft9000}}$ comprend alors:

1. La détermination des canaux virtuels relatifs à l'unité sur laquelle se situe le prochain événement

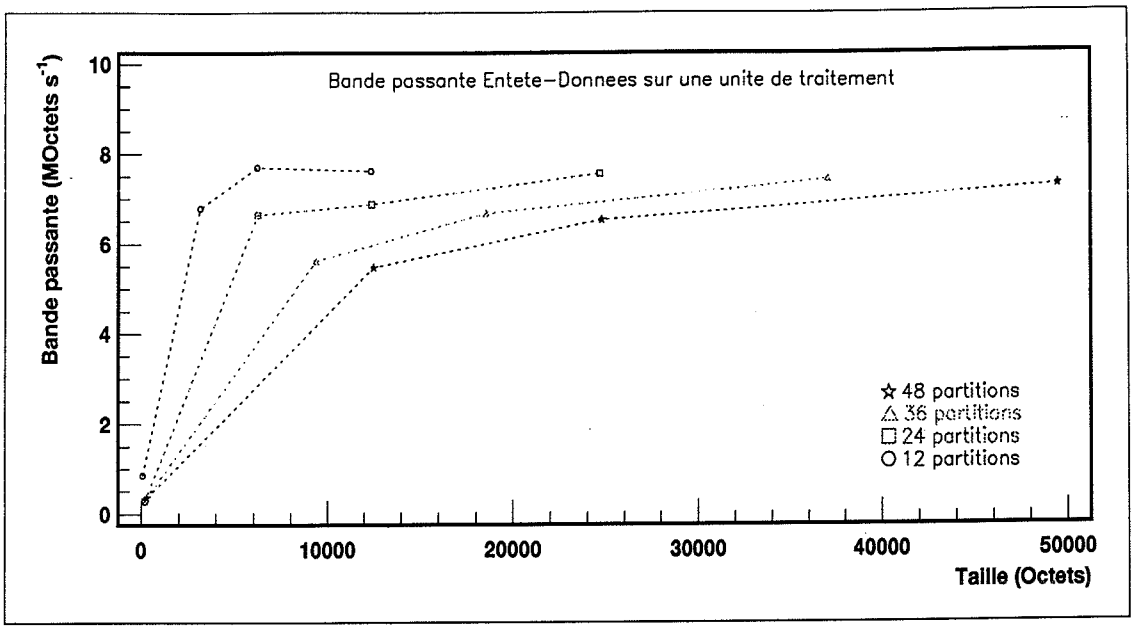


FIG. IV.4 - *Bande passante de l'assemblage complet* {en–tête}–données

2. l'initialisation des partitions et la réception des en–têtes,
3. la vérification de la cohérence de l'événement,
4. l'initialisation des partitions et la réception des donnée,

Tandis que si $Min(T_{ft9000\_prhd}, T_{entity\_pshd}) = T_{entity\_pshd}$, c'est à dire que les tâches *Partition* du *Ft9000* se présentent les premières au rendez–vous, le temps $T_{init_{entity}}$ comprend alors:

1. l'initialisation des partitions et l'émission des en–têtes,
2. la vérification de la cohérence de l'événement,
3. l'initialisation des partitions et la réception des données,

La bande passante limite est alors de 4.44 MOctets s^{-1} pour un transfert caractérisé par:

- une source où le code et les données résident dans une mémoire *DRAM 32 bits cache 16 KOctets*,
- une destination où le code et les données résident dans un mémoire *DRAM 64 bits cache 16 KOctets*,
- par un tranfert sur un lien physique à travers un *C104* sur un seul canal virtuel.

D'après [10], La bande passante pour un transfert entre mémoire *DRAM 32 bits cache 8KOctet* est de 4.57 MOctets s^{-1}.

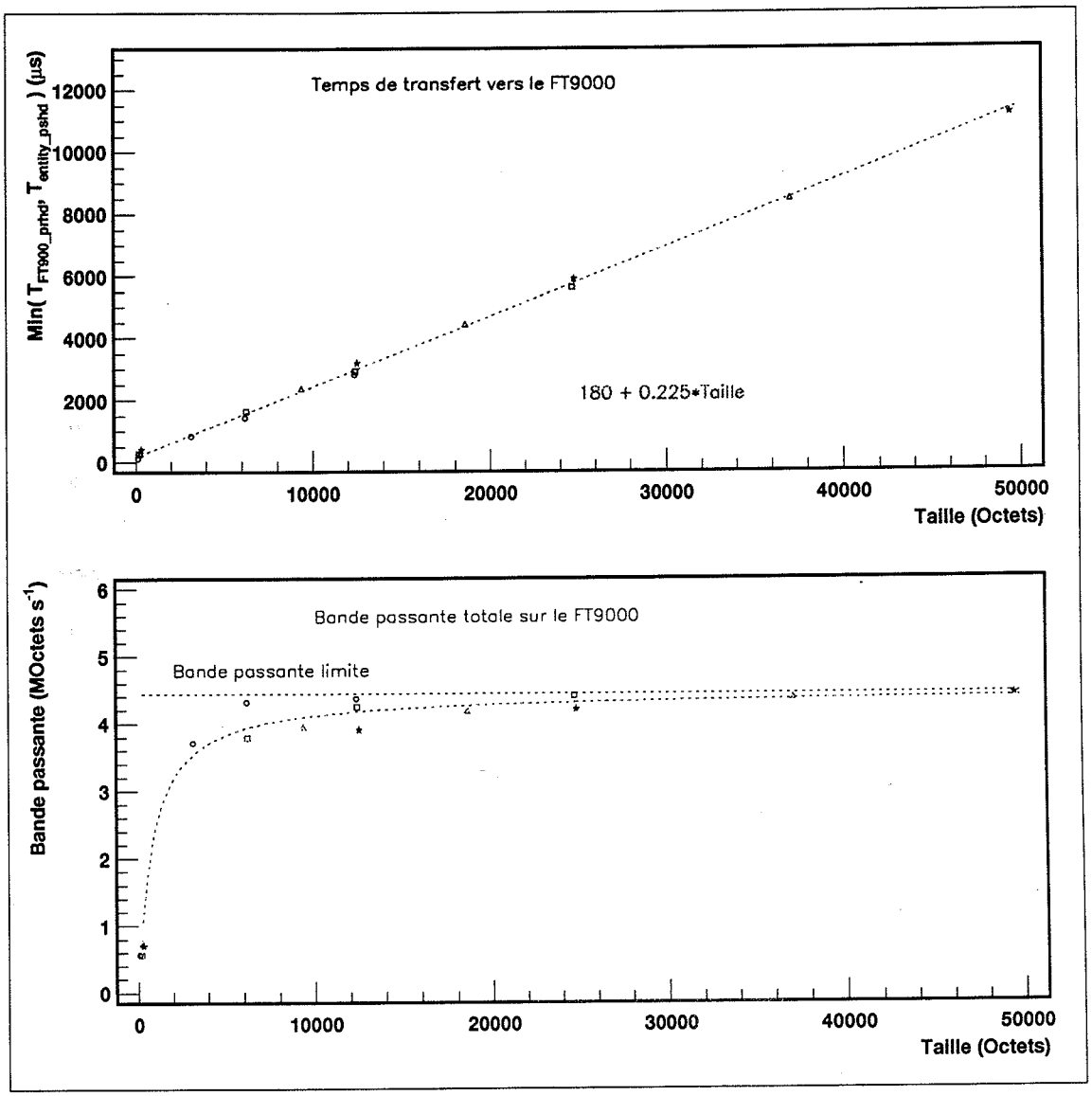


FIG. IV.5 - *Temps de transfert et Bande passante de réception du module* FT9000

IV.4 Conditions expérimentales pour l'assemblage d'un événement pour le système de déclenchement de niveau–2

La description de l'assemblage d'un événement pour le déclenchement de niveau–2 est donnée dans les tableaux D.1, D.2. L'assemblage d'un événement consiste donc:

- sur les unités de traitement à recevoir 76 partitions dont la taille varie de 8 à 512 octets.
- sur les sites *Tmb* à transmettre de 3 à 24 partitions.

IV.5 Temps de Mémorisation des données T_{tmb_M}

L'opération de mémorisation consiste à transférer les données présentes dans les mémoires *Fifo* dans un espace mémoire profond de 32 événements situé dans la mémoire statique à accès *non–cache*.

Le tableau IV.2 donne le temps de mémorisation expérimental pour les différentes mémoires *Tmb*. La figure IV.6 présente la distribution du temps de mémorisation suivant la taille à mémoriser.

Numéro *Tmb*	Taille (Octets)	Temps de Mémorisation $T_{tmb\_M}(\mu s)$	Temps *Fifo–SRAM* $T_{tmb\_FS} = Tdev_{Sram32non-cache} * Taille(\mu s)$
0	392	145.81	58.8
1	1024	237.82	153.6
2	884	216.66	132.6
3	424	138.08	63.6
4	2.12	80.92	0.318
5	2.07	80.92	0.31
6	6.74	83.25	1.01
7	21.74	88.26	3.26
8	16	86.50	2.4
9	1168	249.80	175.2
10	19.25	76.88	2.89
11	19.37	76.93	2.90

TAB. IV.2 - *Temps de mémorisation expériementaux*

Cette distribution est linéaire en fonction de la taille, l'ajustement donne:

$$T_{tmb\_M}(\mu s) = 78 + 0.150 * Taille(octet)$$

où 0.150 correspond au temps de transfert par octets de la mémoire *Fifo* vers la mémoire statique *non–cache*. En effet le transfert s'effectue de la façon suivante:

- lecture d'un mot de 2 octets dans la mémoire *Fifo*,
- puis écriture du mot de 2 octets dans la mémoire statique.

Le temps de transfert $Tdev_{Sram32non-cache}$ s'exprime alors

$$Tdev_{Sram32non-cache} = (Tcycle_{Fifo} + Tbus_{Fifo} + cycle_{Sram} + Tbus_{Sram})/2$$

soit d'après l'annexe B, $Tdev_{Sram32non-cache} = 150ns/octet$.

IV.6 Performance Mémorisation–Assemblage $\{Tmb\text{–}Entité\}$

IV.6.1 Introduction

Le tableau IV.3 présente les conditions de mesure. On ajuste seulement deux paramètres:

- *NbUnité*: le nombre d'unités présentes dans la ferme de traitement,

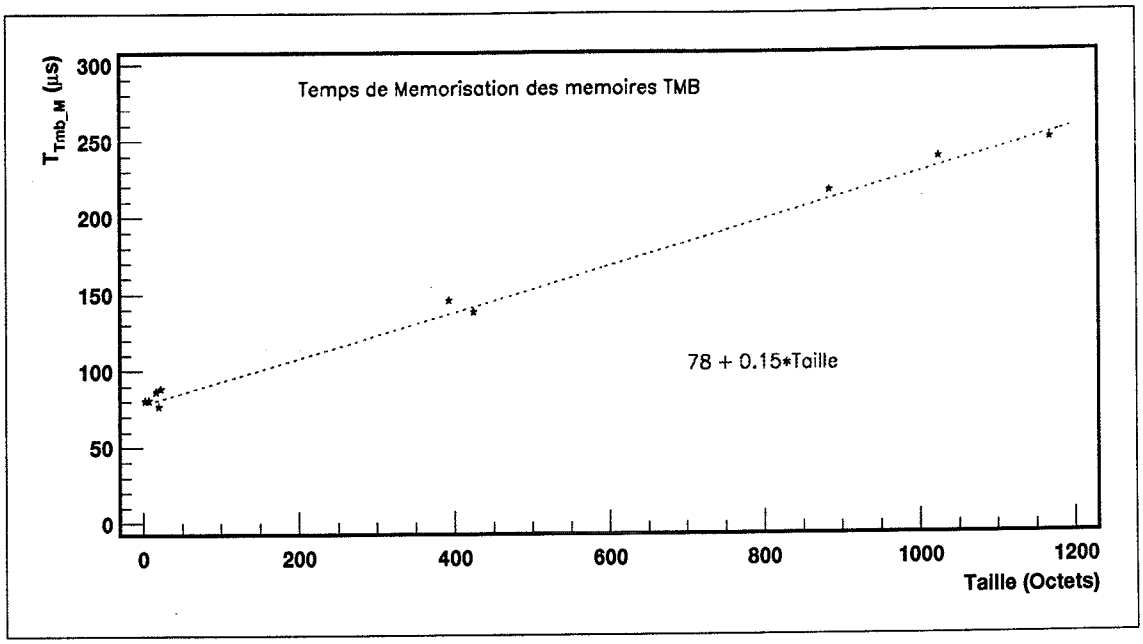


FIG. IV.6 - *Distribution du temps de mémorisation*

Site	Opérateur	Nb Tâches	Précision
12 *Tmbs*	Mémorisation	1	
	Distribution	*NbParEvent*	Transmission des partitions en *série*
	32 emplacements situé dans la *SRAM 32 bits non-cache* Code situé dans la *DRAM 32 bits cache*		
NbUnit Unité	Rassemblement	1	Réception des partitions en *parallèle*
	2 emplacements situé dans la *DRAM 32 bits cache* Code situé dans la *DRAM 32 bits cache*		

TAB. IV.3 - *Conditions de mesure des performances Mémorisation–Assemblage* {Tmb–Entité}

- *NbParEvent*: le nombre d'événements distribués en *paralléle* sur les sites *Tmbs*.

Pour ces mesures, on varie la fréquence de réception d'événement et on mesure pour chaque fréquence:

- le temps moyen d'assemblage complet $< T_{server} >$ suivant le nombre d'unités *NbUnit*.
- le temps moyen d'assemblage des données $< T_{entity\_prd} >$ suivant le nombre d'unités *NbUnit*.
- le temps mort introduit dans l'expérience. Le temps mort est le temps pendant lequel le niveau–2 n'est pas disponible pour recevoir le prochain évémement: il correspond à la durée du signal *Overflow*. On rappelle que celui–ci est positionné sur la réception du signal *Lv1* et est enlevé lorsque l'ensemble des *FIFOs* sont vides.
- la bande passante. Elle est égale au produit de la taille d'un événement par la fréquence d'acceptation d'injection. La fréquence d'acceptation d'injection Fz_{accept} est donnée par le système d'injection.

La taille d'un événement ayant pour valeur moyenne 4164 octets, le nombre de partition 76, on estime d'après l'équation IV.5

$$T_{entity\_prdsoft} = 355.66 + 642.96 = 998.62\mu s$$

et

$$T_{entity\_prdlink} = 485.1\mu s,$$

soit

$$T_{entity\_prd} = 1483.72\mu s.$$

La description d'un événement pour L3 correspond à un cas ou le temps logiciel est prépondérant par rapport au temps de transfert des données sur les liens. De plus pour la réception des en–têtes, comme l'émission est *série* sur les modules *Tmb*, seuls 12 en–têtes sont émis, il faut que les tâches *Partition* de réception soient activées pour que les autres en–têtes soient transmis. Dans ce cas, il y a re–synchronisation entre les sources et la destination, tandis que dans le cas de l'émission *parallèle* l'en–tête constituant un message de 8 octets, l'ensemble des en–têtes sont transmis sans re–synchronisation en dehors de l'acquittement indiquant que les tâches *Partition* se sont présentées au rendez–vous. Cependant la réception des en–têtes comprend l'activation des tâches *Partition* donc

$$T_{entity\_prh} \geq C_{ProcParList} * 76 = 642.96\mu s.$$

IV.6.2 Performance avec une unité de traitement

La figure IV.7 présente le déroulement d'un assemblage selon deux états des mémoires d'entrées:

vides : la demande d'un événement et l'initialisation de la réception des en–têtes sont cachées par l'attente d'un événement.
T_{server} s'exprime alors

$$T_{server} = T_{tmb\_psh} + T_{entity\_prd}.$$

saturées : il faut assembler un événement afin de libérer un emplacement sur les *Tmb*s pour la mémorisation de celui qui est dans les *Fifo*. Une fois l'emplacement libéré, l'opérateur de mémorisation devient actif. Pendant ce temps une nouvelle demande transite sur le serveur tandis que l'unité de traitement prépare la réception des en–têtes.

Le tableau IV.4 présente les mesures obtenues.

Fréquence Hz	$< Tserver >$ μs	$< Tentity\_prd >$ μs	Temps mort μs	Bande Passante $Moctets\ s^{-1}$
100.1	1780	1450	355	0.416
201.8	1780	1450	355	0.840
301.9	1780	1450	355	1.256
405.3	2340	1450	383	1.686
420.4	2002	1450	948	1.716
454.4	2000	1450	1272	1.745
505.4	2012	1450	1354	1.745
616.2	1996	1450	1539	1.746
817.9	1997	1450	1740	1.746
1046.1	1997	1450	1868	1.745
2142.2	1997	1450	2113	1.746

TAB. IV.4 - *Mesures Mémorisation–Assemblage* {Tmb–Entité} *à une unité de traitement*

La figure IV.8 représente la variation du temps mort, la bande passante et la distribution d'occupation des emplacements sur les mémoires d'entrée en fonction de la fréquence d'injection. La distribution d'occuapation des emplacements sur les mémoires d'entrée permet de mieux comprendre comment le système passe d'un régime non–saturé à un régime saturé.

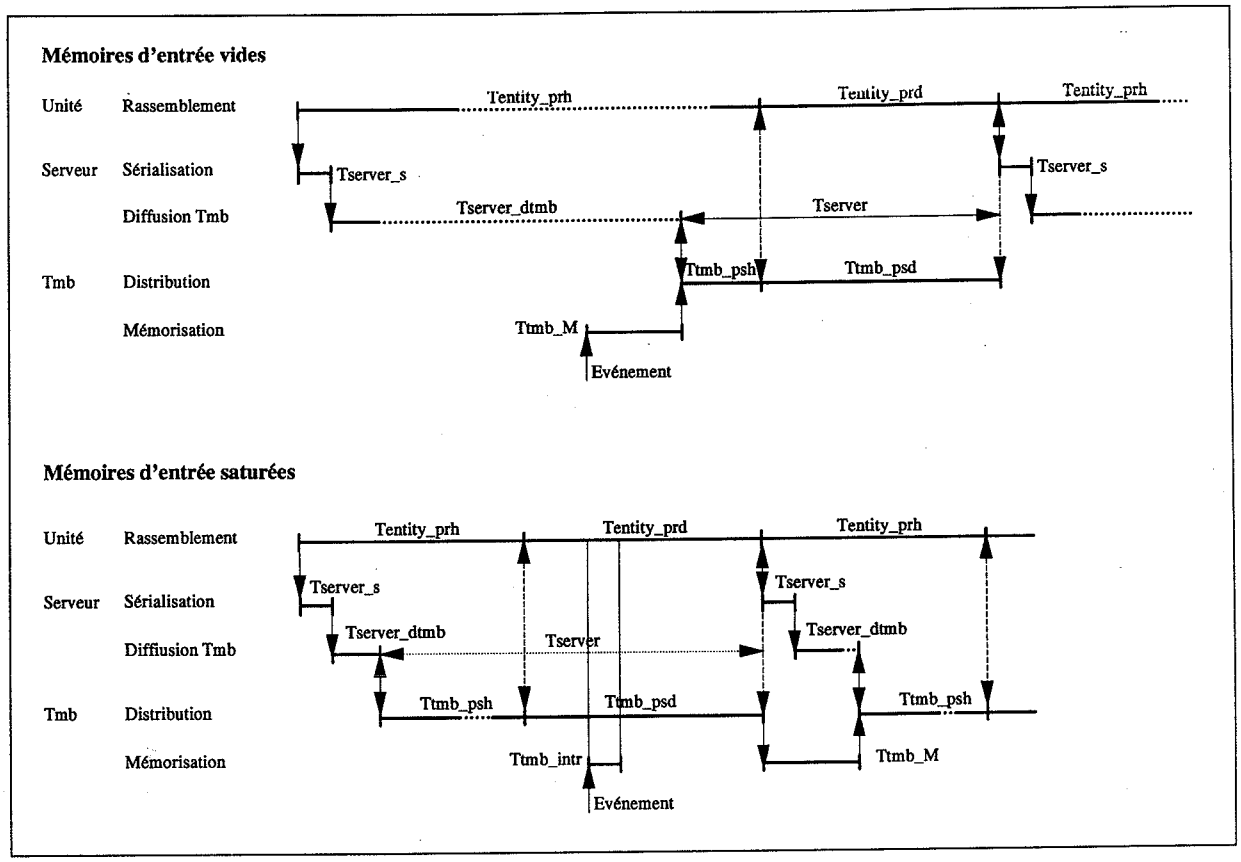


FIG. IV.7 - *Chronogramme Mémorisation–Assemblage* {Tmb–Entité} *à une unité de traitement*

On constate que:

- la bande passante croît linéairement avec la fréquence d'injection jusqu'à $420Hz$ pour atteindre la valeur maximale et limite de $1.746Moctets\ s^{-1}$. Cette valeur limite est contrainte par le temps d'activation des tâches *Partition* pour la réception des en–têtes.

- le temps mort est constant et égal à $355\mu s$ pour des fréquences d'injection inférieures à $400Hz$, et qu'il croît ensuite et tend vers une valeur limite.

- $T_{entity\_prd} = 1450\mu s$, peu différent de la valeur estimée $1483.72\mu s$ est constant quel que soit la fréquence d'injection.

- T_{server} prend trois valeurs correspondants aux trois états possibles pour les mémoires d'entrées:

  vide : lorsqu'un seul emplacement est utilisé, $T_{server} = 1780\mu s$ et correspond à $T_{tmb\_psh} + T_{entity\_prd}$ d'ou $T_{tmb\_psh} = 330\mu s$.

  intermédiaire : lorsque plusieurs emplacements sont utilisés mais certains restent disponibles. Dans ce cas l'assemblage d'un événement est interrompu par l'opération de mémorisation: T_{server} augmente, pour la fréquence $405Hz$ il reste presque toujours un emplacement de disponible, cf figure IV.8(a), on a alors $T_{server} = 2340\mu s$.

  saturé : tous les emplacements sont utilisés. T_{server} correspond à $T_{tmb\_psh} + T_{entity\_prd}$ plus l'attente de re–synchronisation pour l'échange des en–têtes.

- le protocole {*en-tête*}-*données* introduit un temps supplémentaire T_{tmb_psh} correspondant uniquement au temps de transmission des *Tmb*s tant que le système n'est pas saturé. Ensuite, c'est la mise en place de tâches de réception sur l'unité de traitement qui augmente ce temps.

Bande passante

La bande passante limite correspond à $T_{Fz_{limit}} = T_{entity\_prh} + T_{entity\_prd}$. Si l'on mesure T_{entity_prh} à la fréquence d'injection de $827Hz$, on a $T_{entity\_prh} = 786\mu s$ d'ou une fréquence limite $Fz_{limit} \cong 448Hz$. La différence avec $420Hz$ peut avoir diverses origines:

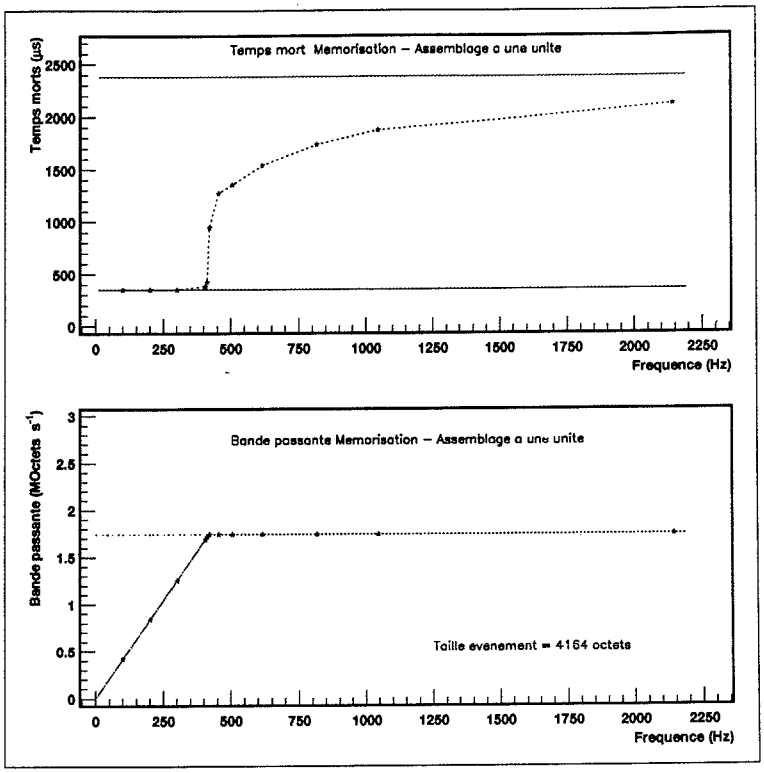
1. le temps T_{tmb_psh}. En effet lorsque les mémoires d'entrée sont saturées, le décalage de réception du signal *Lv1* sur les modules $Tmb_{9,10,11}$ est caché, ainsi l'ensemble des en-têtes sont émis simultannément tandis que en mode non-saturé les autres modules peuvent commencer à émettre leurs en-têtes pendant que les modules $Tmb_{9,10,11}$ exécutent leur opération de mémorisation.

2. le mécanisme de synchronisation entre les opérateurs.

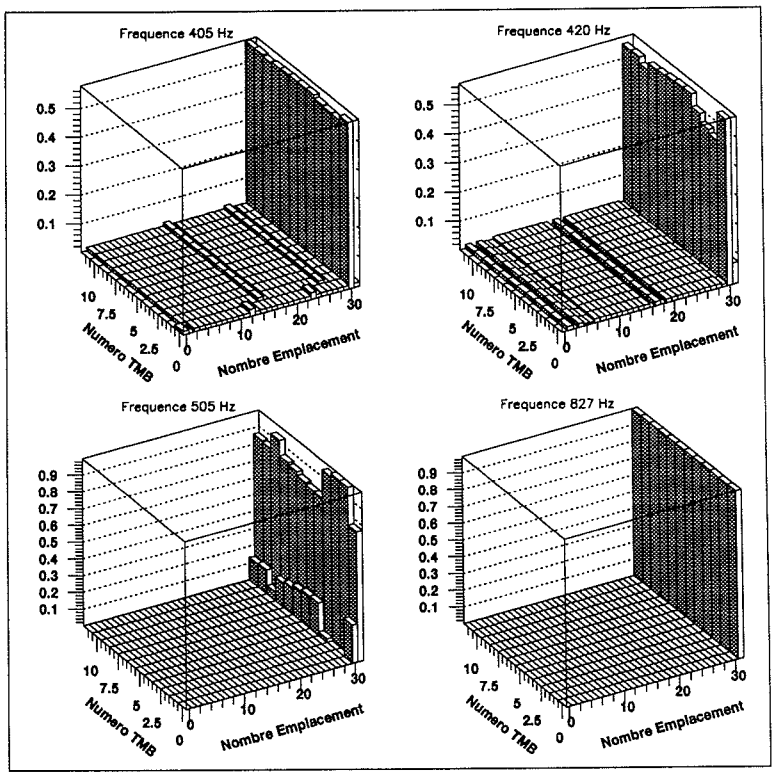
3. le mécanisme de synchronisation avec l'expérience qui n'est plus caché.Celui-ci s'effectue par l'intermédiaire d'une tâche *haute priorité* qui reçoit la synchronisation expérimentale puis la transmet à la tâche de l'opérateur de mémorisation.

Temps mort

Pour un taux de déclenchement inférieur à $400Hz$, le temps mort appliqué est introduit par l'opération de mémorisation. Il est égal à la somme du temps de transfert *Fifo-Sram*,$T_{tmb\_FS_9}$, et du retard pour la réception du signal *Lv1* par rapport aux autres modules (voir figure B.1) soit $180 + 175 = 355\mu s$.
Le temps mort maximal appliqué correspond à $1/Fz_{max}$. Le système tend vers cette valeur limite sans jamais l'atteindre.



(a) Temps mort et bande passante



(b) Distribution d'occupation des emplacements sur les *Tmbs*

FIG. IV.8 - *Mesures à une unité de traitement*

IV.6.3 Performance avec deux unités de traitement

La ferme de traitement est maintenant constituée de deux unités mais on autorise la distribution uniquement d'un seul événement par les mémoires d'entrées.

La figure IV.9 présente le déroulement d'un assemblage selon deux états des mémoires d'entrées, vides ou saturées. Le gain apporté par un fonctionnement à deux unités consiste essentiellement à cacher une partie de T_{entity_prh}: il permet d'augmenter la fréquence maximale. La partie du temps caché vaut

$$T_{entity\_prh} - T_{tmb\_psh} = 456\mu s$$

d'ou une fréquence maximale attendue de

$$Fz_{max} \cong (2381 - 456)^{-1} \cong 520Hz.$$

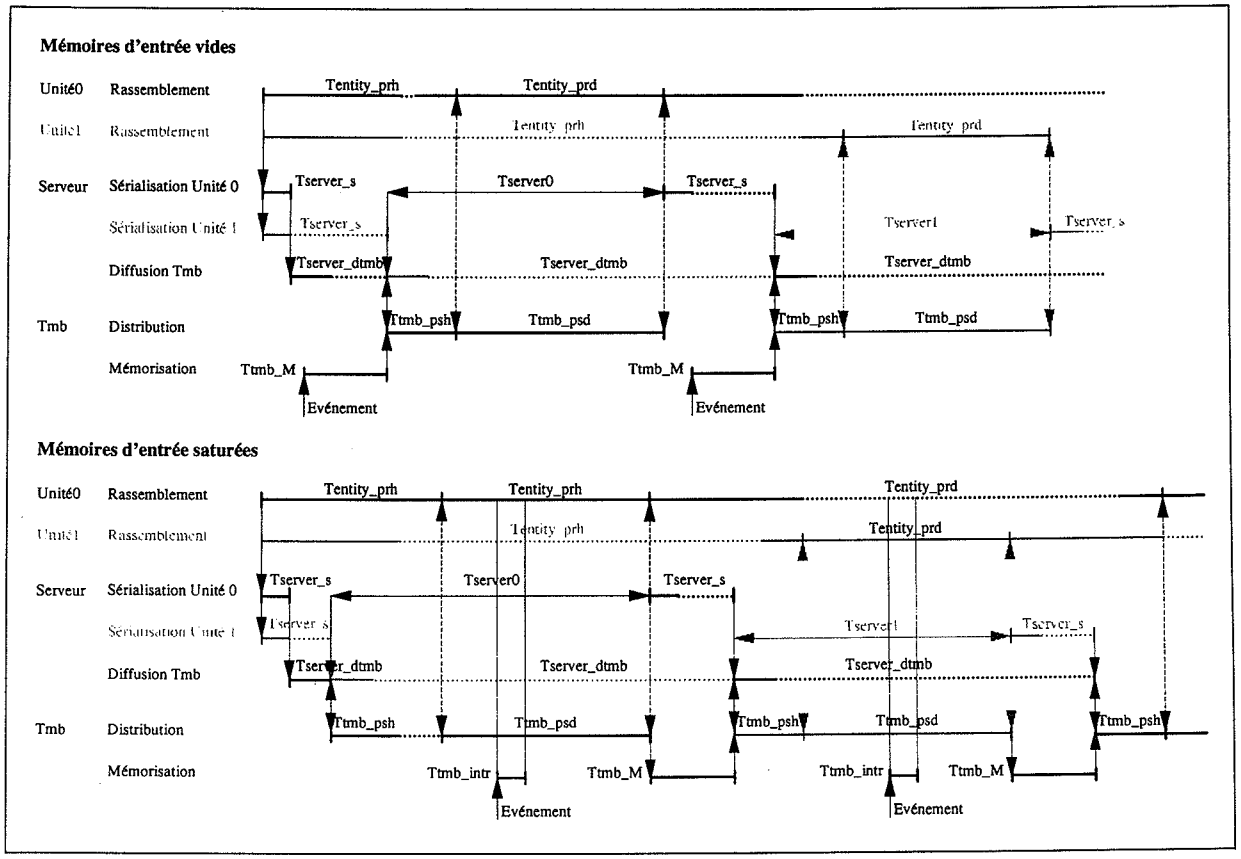


FIG. IV.9 - *Chronogramme Mémorisation-Assemblage* {Tmb–Entité} *à deux unités de traitement*

Le tableau IV.5 présente les mesures obtenues uniquement pour quelques points intéressants.

La figure IV.10 représente la variation du temps mort, la bande passante et la distribution d'occupation des emplacements sur les mémoires d'entrée en fonction de la fréquence d'injection.

On constate que:

- la fréquence $Fz_{max} = 529Hz$ peu différente de la fréquence attendue $520Hz$.

Fréquence Hz	$< Tserver >$ μs	$< Tentity\_prd >$ μs	Temps mort μs	Bande Passante $Moctets\ s^{-1}$
100.1	1785	1455	356	0.416
405.3	1785	1455	356	1.687
505.4	2000	1558	356	2.151
529.2	1915	1470	356	2.202
541.9	1900	1460	1305	1.839
2142.2	1910	1455	1997	1.836

TAB. IV.5 - *Mesures Mémorisation–Assemblage* {Tmb–Entité} *à deux unités de traitement*

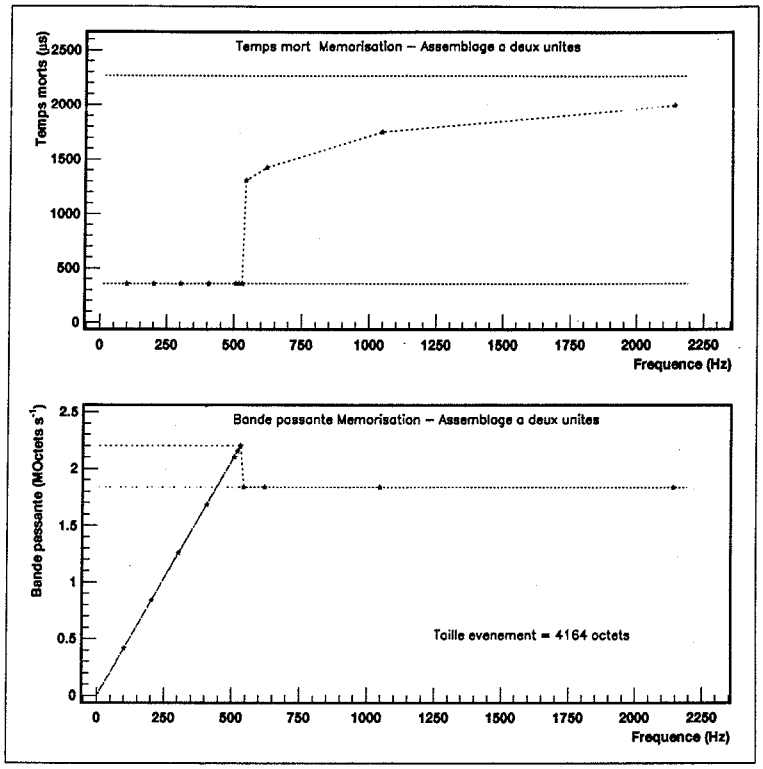
- la bande passante croît linéairement avec la fréquence d'injection jusqu'à Fz_{max} pour atteindre la valeur maximale de $2.2 Moctets\ s^{-1}$. Puis lorsque les mémoires d'entrée sont saturées, la bande passante devient égale à la valeur limite de $1.84 Moctets\ s^{-1}$ quel que soit la fréquence d'injection $Fz \geq Fz_{max}$. Cette valeur limite correspond à une fréquence limite d'injection $Fz_{limite} \cong 443 Hz$.

- le temps mort est constant et égal à $356 \mu s$ pour des fréquences d'injection inférieures à Fz_{max}, puis qu'il croît vers une valeur limite.

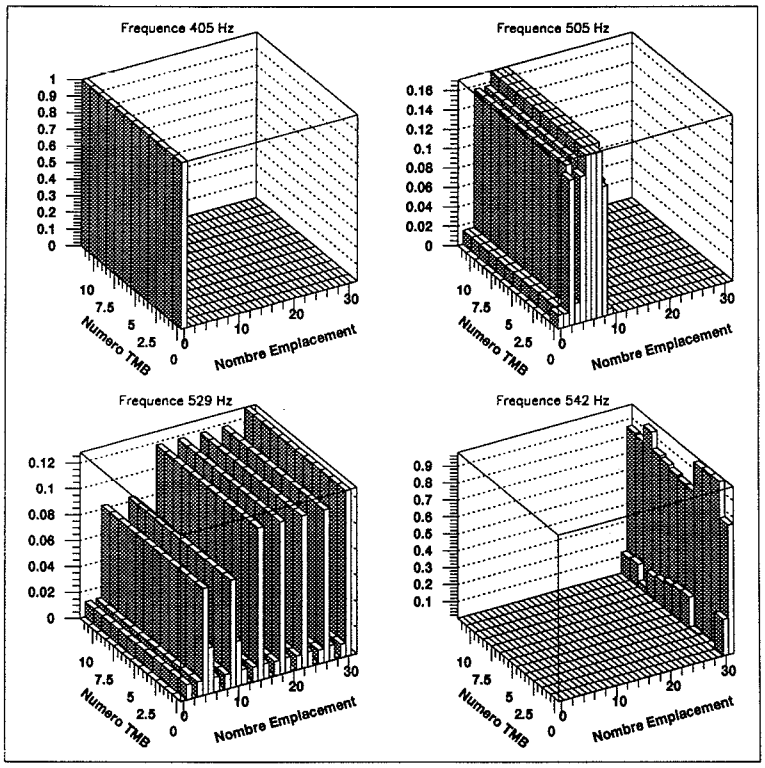
- $T_{entity\_prd_{unit_0}} \cong T_{entity\_prd_{unit_1}}$. $< T_{entity\_prd} > \cong 1455 \mu s$ sauf pour $Fz = 505 Hz$ où $T_{entity\_prd} = 1558 \mu s$. Pour cette fréquence, les opérations de mémorisation et de distribution sur les *Tmb*s s'effectuent en *parallèle*, cf figure IV.10(b). A ceci près que l'opération de mémorisation est une opération *haute priorité,* elle peut donc *interrompre* ou *ralentir* l'opération de distribution et ainsi augmenter T_{entity_prd}.

- $T_{server_{unit_0}} \cong T_{server_{unit_1}}$. On s'attendait à avoir un temps $< T_{server} >$ constant égal à $\cong 1780 \mu s$ pour l'état vide et saturé des mémoires d'entrée. Or il s'avère que pour l'état saturé, on a $< T_{server} > \cong 1900 \mu s$ ceci pour les raisons énumérées précédemment.

L'intérêt du fonctionnement à deux unités avec une seule tâche de distribution sur les *Tmb*s est que s'il existe toujours une unité de traitement disponible, on peut masquer une partie du temps correspondant à l'initialisation de la réception des en–têtes. On a alors une bande passante maximale supérieure à la bande passante limite. La bande passante limite est obtenue lorsque les mémoires d'entrée sont saturées: la limitation se situe alors sur les sources.



(a) Temps mort et bande passante



(b) Distribution d'occupation des emplacements sur les *Tmbs*

FIG. IV.10 - *Mesures à deux unités de traitement et un événement distribué à la fois par les* Tmb*s*

IV.6.4 Performance avec deux unités de traitement avec multi–distribution

La ferme de traitement est maintenant constituée de deux unités mais on autorise la distribution simultannée de deux événements par les mémoires d'entrées en imposant la restriction suivante:

- a chaque instant, il ne peut y avoir que les en–têtes ou les données d'un même événement qui ont accès au lien.

Ceci permet:

- sur chaque site *Tmb* d'utiliser le temps d'initialisation T_{entity_pdsoft} pour transmettre les en–têtes vers l'autre unité de traitement.
- d'utiliser la distribution temporelle entre les *Tmb*s: un module peut avoir finit de transmettre les données vers une unité de traitement et commencer à transmettre les en–têtes de l'événement suivant vers l'autre unité pendant que certains modules continuent de transmettre leurs données. Cependant comme le nombre et la taille des partitions varient entre les modules *Tmb*s , la distribution temporelle est limitée par le module qui prend le plus de temps à transmettre ces partitions.
- de ne pas augmenter T_{entity_prd} et d'utiliser pleinement la bande passante du lien de sortie des mémoires em Tmb pour transmettre l'événement.

On s'attend alor à une fréquence maximale telle que

$$1/Fz_{max} = T_{entity\_prd} + T_{tmb\_M_{max}}$$

soit $Fz_{max} = (1450 + 250)^{-1} \cong 588Hz$.

Le tableau IV.6 présente les mesures obtenues uniquement pour quelques points intéressants. Lorsque l'on augmente la fréquence d'injection au delà de $616Hz$ le système d'assemblage *casse* pour des raisons que nous ne connaissons pas et que nous n'avons pas pu déterminer par manque de temps.

Fréquence Hz	$< Tserver >$ μs	$< Tentity\_prd >$ μs	Temps mort μs	Bande Passante $Moctets\ s^{-1}$
405.3	1790	1469	356	1.686
505.4	3836	1653	356	2.102
529.2	–	–	356	2.202
541.9	–	–	356	2.254
569.4	3395	1463	356	2.369
584.0	3832	1455	1203	1.988
616.0	3775	1455	1234	2.013

TAB. IV.6 - *Mesures Mémorisation–Assemblage* {Tmb–Entité} *à deux unités de traitement avec multi–distribution par les* Tmb*s*

La figure IV.11 représente la variation du temps mort, la bande passante et la distribution d'occupation des emplacements sur les mémoires d'entrée en fonction de la fréquence d'injection.

On constate que:

- la fréquence $Fz_{max} = 569Hz$ autorisant une bande passante maximale de $2.37Moctets\ s^{-1}$. La différence observée avec Fz_{max} attendue peut être liée au mécanisme de synchronisation avec l'expérience.
- la bande passante limite est de $2.0Moctets\ s^{-1}$ correspondant à $Fz_{limite} \cong 480Hz$.

- $T_{entity\_prd_{unit_0}} \cong T_{entity\_prd_{unit_1}} . < T_{entity\_prd} >$ est constant et vaut $\cong 1455\mu s$. Ce qui signifie que l'on transfert bien l'en–tête de l'événement suivant pendant le temps T_{entity_pdsoft} et sans augmenter le temps d'assemblage de l'événement puisque $< T_{entity\_prd} >$ est constant.

- $T_{server_{unit_0}} \cong T_{server_{unit_1}} . < T_{server} >$ n'est plus indépendant de l'état des mémoires d'entrée:

  vide : $< T_{server} > \cong 1780\mu s$ est équivalent au à $< T_{server} >$ sans multi–distribution.

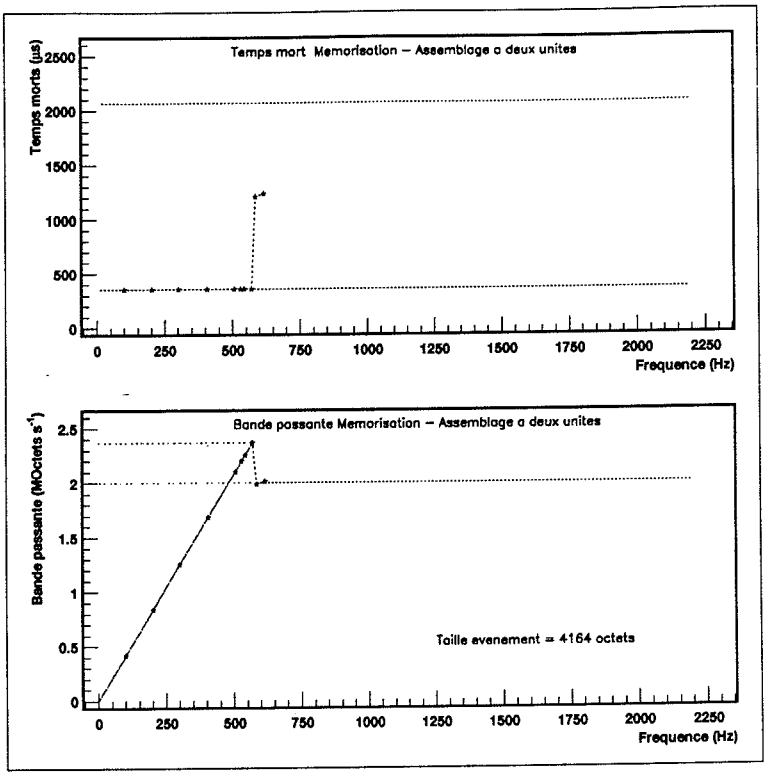
  intermédiaire, saturé : $3395\mu s \leq < T_{server} > \leq 3836\mu s$. Cette augmentation de $< T_{server} >$ est du au fait qu'il y ait deux tâches de distribution sur les *Tmb*s et que l'accès au lien est sérialisé pour les événements.

La multi–distribution d'événements sur un même lien en sérialisant l'accès à ce lien nous permet obtenir la bande passante maximale de $2.37 Moctets\ s^{-1}$ tout en conservant un temps d'assemblage de $\approx 1.8ms$. Lorsque l'on autorise autant d'événements à être assemblés en paralléle qu'il y a d'unités de traitement dans la ferme, le temps de demande d'événements n'est plus masqué.
Il serait intéressant d'effectuer les tests suivants:

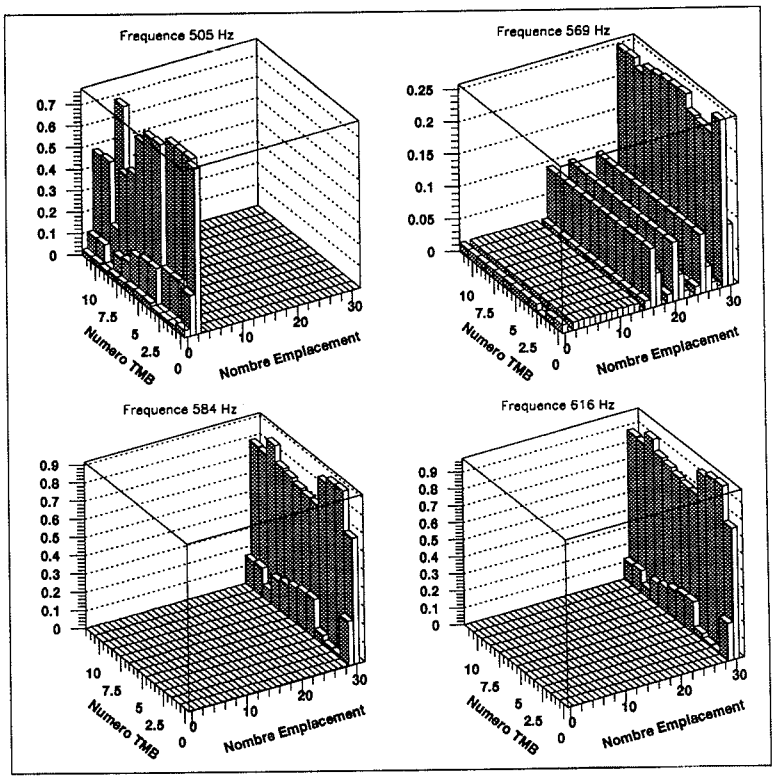
1. d'augmenter le nombre d'unités de traitement avec un cœfficient de multi–distribution prenant pour valeur $NbParEvent \in \{1, NbUnit - 1, NbUnit\}$ et ainsi vérifier que l'on peut masquer le temps de demande si le coefficient de multi–distribution est inférieur au nombre d'unités de la ferme.

2. d'essayer de réduire la contention sur le circuit de routage *C104* en utilisant des techniques de *distribution de traffic* sur les mémoires *Tmb*s pour optimiser l'occupation des liens d'émission sur les *Tmb*s.

3. d'effectuer la multi–distribution d'événements sur des liens distincts pour vérifier le parallélisme géométrique. On espère ainsi multiplier par un facteur ≈ 2 la bande passante maximale et conserver le temps de latence constant.



(a) Temps mort et bande passante



(b) Distribution d'occupation des emplacements sur les *Tmb*s

FIG. IV.11 - *Mesures à deux unités de traitement avec multi-distribution d'événements par les* Tmb*s*

IV.6.5 Optimisation de l'assemblage

On a vu précédemement que

$$T_{entity\_prd} = C_{prdsoft} * NombredePartitions + C_{link_2} * Taille.$$

L'optimisation de l'assemblage peut donc s'effectuer de différentes façons:

1. de diminuer les deux constantes en positionnant la mémoire-tampon sur les unités de traitement dans une mémoire *non-cache* indépendante du code.

2. en conservant le même nombre de partitions mais en essayant de diminuer $C_{prdsoft}$ par la mise en œuvre d'autres mécanismes de synchronisation.

3. de diminuer le nombre de partitions en mettant en place un assemblage logiciel sur les unités de traitement

4. en conservant l'assemblage dynamique, canaux virtuels, mais en activant seulement autant de tâches qu'il y a de mémoires *Tmb*. Et d'utiliser une boucle pour recevoir les partitions à l'intérieur de ces tâches pour recevoir les partitions.

Influence de la localisation de l'espace mémoire-tampon sur les unités de traitement

On utilise la configuration à une unité et suivant la localisation de l'espace mémoire-tampon, on relève T_{server}, T_{entity_prd} et Fz_{max}. Le tableau IV.7 présente les mesures obtenues.

Localistion espace mémoire-tampon	*DRAM cache 32 bits*	*SRAM non-cache 32 bits*
T_{server} (μs)	1997	1640.5
T_{entity_prd} (μs)	1450	1232.3
Fz_{max} (Hz)	420	495

TAB. IV.7 - *Performance de l'assemblabe à une unité suivant la localisation de l'espace mémoire-tampon*

En autorisant l'accès à la mémoire *cache* uniquement pour le code, la fréquence d'injection maximale des données passe de $420Hz$ à $495Hz$ et permet de diminuer les deux constantes puisque $\Delta T_{server} \geq \Delta T_{entity\_prd}$.
Le fait d'utiliser la même mémoire cache pour le code et les données détériore le temps d'assemblage.

Constante de synchronisation

Avec deux unités de traitement à reconstruire simultannément des événements, on obtient une fréquence Fz_{max} telle que:

$$Fz_{max} = (T_{entity\_prd} + T_{tmb\_M_{max}})^{-1}.$$

Cette fréquence maximale est donc limitée par T_{entity_prd}. Une façon pour essayer de diminuer ce temps liè à l'activation d'une liste de tâches *synchrones* est:

- d'activer cette liste de tâches une seule fois,

- d'utiliser des *sémaphores* ou des *canaux de communication internes* pour synchroniser les tâches *Partition* avec la tâche *maître* gérant l'emplacement.

La mise en œuvre de ce nouveau mode de synchronisation s'effectue de la façon suivante:

tâche *maître* :

1. initialisation et reveil des tâches *Partition* pour la réception des en–têtes: *SemSignal* ou *ChanOutChar*.
2. attente réceptions des en–têtes: *SemWait* ou *ChanInChar*.
3. initialisation et reveil des tâches *Partition* pour la réception des données: *SemSignal* ou *ChanOutChar*.
4. attente réceptions des données: *SemWait* ou *ChanInChar*.

tâche *Partition* :

1. attente initialisation réception en–tête: *SemWait* ou *ChanInChar*
2. réception en–tête.
3. signale la réception: *SemSignal* ou *ChanInChar*.
4. attente initialisation réception données: *SemWait* ou *ChanInChar*
5. réception données.
6. signale la réception: *SemSignal* ou *ChanInChar*.

Conceptuellement ce mode de synchronisation offre plus de *parallélisme* puisque dès qu'une tâche *Partition* est réveillée, elle peut recevoir l'information pendant que la tâche *maître* continue de réveiller les suivantes. Cependant il existe une différence fondamentale entre une synchronisation avec un *sémaphore* et une synchronisation avec un *canal interne*:

sémaphore : l'instruction *SemSignal* ne constitue pas un point de débranchement, seule l'instruction *SemWait* constitue un point de débranchement uniquement si la tâche qui l'exécute trouve un sémaphore de valeur nulle.

canal interne : les instructions *ChanInChar* et *ChanOutChar* constituent des points de débranchement systématiques si la tâche qui les exécute se présente la première au rendez–vous.

La différence fondamentale entre ces trois modes de synchronisation se situe dans l'ordonnancement des tâches *Partition* par rapport à la tâche *maître*.

Comme l'initialisation et le réveil sont imbriquées, on détermine la valeur de $C_{prdsoft}$ à partir de l'équation

$$T_{entity\_prd} = C_{prdsoft} * NombredePartitions + C_{link_2} * Taille$$

en mesurant T_{entity_prd}. Le tableau IV.8 présente les mesures et les résultats obtenus. Ces mesures sont obtenues avec la configuration à une unité de traitement pour une fréquence d'injection de $1000Hz$, fonctionnement en mode saturé quel que soit le type de synchronisation.

Synchronisation	*ProcParList*	*Sémaphores*	*Canaux internes*
T_{server} (μs)	2100.25	2421.5	4694.5
T_{entity_prh} (μs)	842	961.9	1808.9
T_{entity_prd} (μs)	1552	1631.1	2715.38
Taille $(octets)$	4414	4406	440.2
Fz_{max} (Hz)	407	344	207
$C_{prdsoft}$ $(\mu s/partition)$	13.43	14.49	28.7

TAB. IV.8 - *Estimation de la constante d'initialisation et d'activation des partitions suivant le mode de synchronisation*

On constate que:

- $Min(C_{prdsoft})$ est obtenue lorsque la synchronisation est effectuée en activant pour chaque événement les tâches *Partition* pour la réception des en–têtes et des données.
- la synchronisation avec *Sémaphores* est presque équivalente à la synchronisation avec *ProcParList*.
- la synchronisation avec *canaux internes* diminue d'un facteur ≈ 2 les performances du systéme d'assemblage.

La solution la moins pénalisante pour la synchronisation entre les tâches *Partition* et la tâche *maître* consiste à activer les tâches *Partition* pour la réception des en–têtes et des données.

Assemblage logiciel sur les unités de traitement

Lorsque l'on effectue de l'assemblage logiciel sur les unités de traitement, on envisage de transférer soit une partition par *Tmb*, soit une partition par port *Tmb*. Ces partitions sont stockés dans un espace temporaire puis copiés par *BlockMove* dans leur emplacement exact. Quel que soit le type d'assemblage, le temps d'occupation des liens est le même. Le nombre minimum de blocs reçus est égal aux nombres de sources. Le temps T_{entity_move} a été estimé

Nombre de Partitions	$T_{entity\_prdlink}$ μs	$T_{entity\_prdsoft}$ μs	T_{entity_move} μs	T_{entity_prd} μs
12	486	$8.5 * 12 + 4.5 * 76$	$1100_{32} * 0.4$	≈ 1370
48	486	$8.5 * 48 + 4.5 * 76$	$1100_{32} * 0.4$	≈ 1676
76	486	$8.5 * 76 + 4.6 * 76$	0	1486_{DRAM}
	–	–	0	1233_{SRAM}

TAB. IV.9 - *Estimation du temps d'assemblage*

pour une mémoire ayant un temps d'accès de 200 ns/cycle.
Dans les conditions expérimentales L3, le temps d'assemblage est du même ordre de grandeur si l'on transfert les 76 blocs sans assemblage logiciel ou si l'on transfert uniquement 12 blocs, correspondant au nombre de source, et que l'on effectue ensuite un assemblage logiciel.

La limitation du temps d'assemblage se situe, dans notre cas, sur le nombre de tâches à activer pour recevoir en *parallèle* les différentes partitions. Il resterait a faire le test en activant autant de tâches qu'il y a de sources, chaque tâche recevant en *série* les différentes partitions relative à la source. Mais par manque de temps et de disponibilité du site, ce test n'a pas pu être effectué.

IV.7 Performance Mémorisation–Assemblage {*Tmb–Entité*}–Traitement

IV.7.1 Introduction

Par ces mesures, on cherche à comprendre comment le système d'assemblage se comporte lorsque l'on introduit du temps de traitement. Ensuite, on souhaite déterminer pour une fréquence maximale d'injection de $50Hz$ le temps de traitement maximum permis.

Le tableau IV.10 présente les conditions de mesures.

On ajuste seulement deux paramètres:

- le nombre d'unités présentes dans la ferme de traitement *NbUnité*,

Site	Opérateur	Nb Tâches	Précision
12 *Tmbs*	Mémorisation	1	
	Distribution	1	Transmission des partitions en *série*
	32 emplacements situé dans la *SRAM 32 bits non–cache* Code situé dans la *DRAM 32 bits cache*		
NbUnit Unité	Rassemblement	1	Réception des partitions en *parallèle*
	Traitement	1	Tâche *Haute Priorité*, Temps de Traitement T_{entity_T}
	2 emplacements situé dans la *DRAM 32 bits cache* Code situé dans la *DRAM 32 bits cache*		

TAB. IV.10 - *Conditions de mesure des performances Mémorisation–Assemblage* {Tmb–Entité} *–Traitement*

- le temps de traitement simulé sur les unités de traitement T_{entity_T}.

et on relève pour chaque point le temps mort et la fréquence d'acceptation d'injection Fz_{accept}.

IV.7.2 Temps de traitement de 10 ms

La figure IV.12 présente le chronogramme des opérations pour une unité.

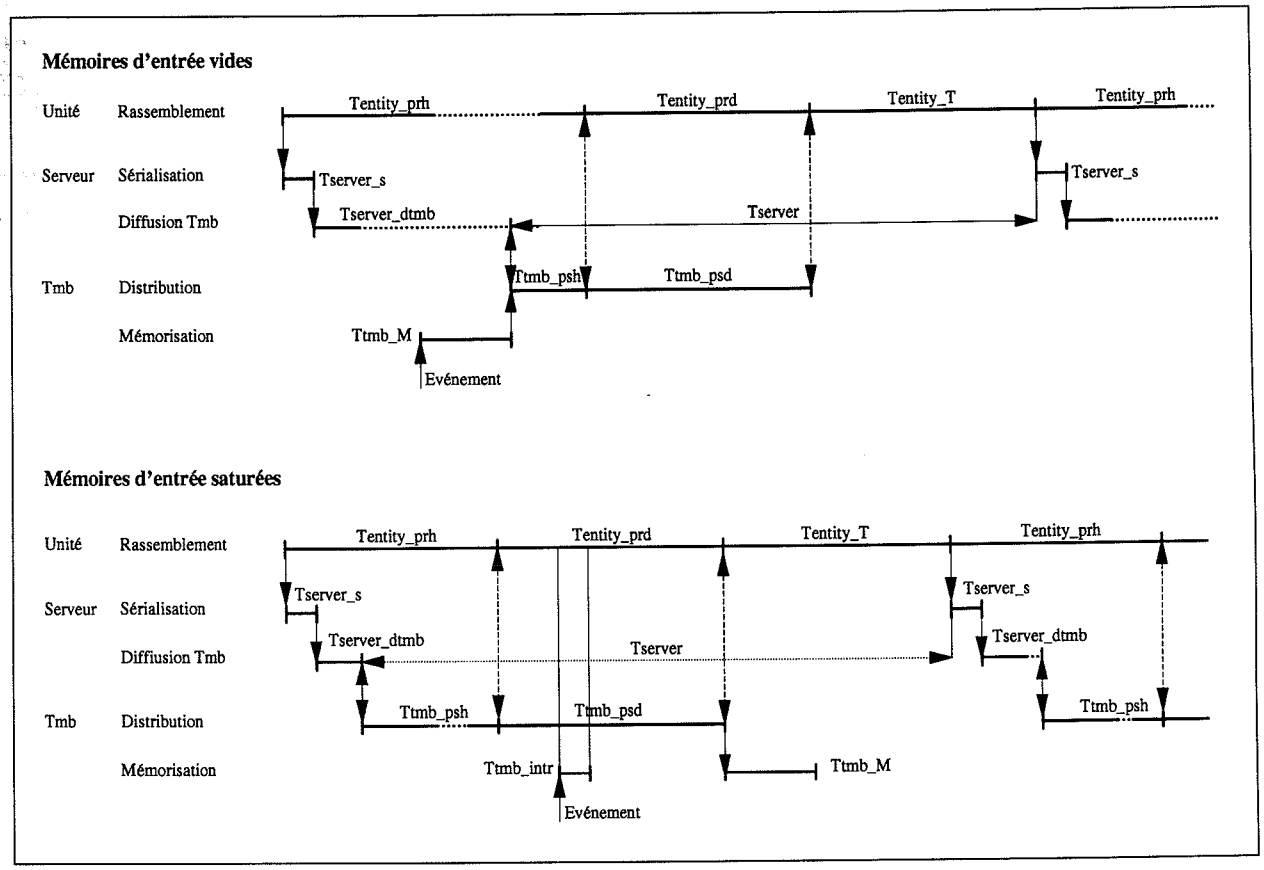


FIG. IV.12 - *Chronogramme Mémorisation–Assemblage* {Tmb–Entité}*–Traitement à une unité*

Le tableau IV.11 donne les mesures obtenues pour une ferme contenant une unité de traitement.

Le temps de traitement des algorithmes de sélection est compris entre 2 et 10 ms au maximun. On voit donc clairement que pour le déclenchement de niveau–2, une ferme de traitement

Fréquence (Hz)	Fz_{accept} (Hz)	Temps mort (μs)
19.95	19.95	356
40.06	40.02	356
60.06	60.04	356
80.18	77.55	2338
100.18	75.75	8152

TAB. IV.11 - *Mesures Mémorisation-Assemblage* {Tmb-Entité}*-Traitement à une unité pour un temps de 10 ms*

constituée d'une seule unité satisfait le cahier des charges puisque la fréquence maximale est de $75Hz \geq 50Hz$.

Le tableau IV.12 donne les mesures obtenues pour une ferme contenant deux unités de traitement.

Fréquence (Hz)	Fz_{accept} (Hz)	Temps mort (μs)
20.02	20.02	356
100.18	100.18	356
120.06	120.60	356
201.71	153.68	3654

TAB. IV.12 - *Mesures Mémorisation-Assemblage* {Tmb-Entité}*-Traitement à deux unités pour un temps de 10 ms*

On constate que pour deux unités, la fréquence maximale est de $153Hz$ et équivaut donc à $\approx 2 * 75$. On peut accroître la bande passante du système en augmentant le nombre d'unités dans la ferme de traitement.

IV.7.3 Temps de traitement disponible avec deux unités de traitement pour une fréquence d'injection de $50Hz$

Pour deux unités, $Fz_{max} = T_{entity\_T}/2 + T_{entity\_prhd}$ soit $T_{entity\_prhd} = 1536\mu s$. Donc pour $Fz_{max} = 50Hz$, le temps de traitement disponible est de $\cong 18ms/unite$. Le tableau IV.13 donne les mesures obtenues pour une ferme contenant deux unités de traitement avec $T_{entity\_T} = 36ms$.

Fréquence (Hz)	Fz_{accept} (Hz)	Temps mort (μs)
19.98	19.98	356
40.06	40.06	356
50.04	50.04	356
60.06	51.60	7450

TAB. IV.13 - *Mesures Mémorisation-Assemblage* {Tmb-Entité}*-Traitement à deux unités pour un temps de 36 ms*

La fréquence maximale est de $51.60Hz$, donc le temps de traitement disponible pour le traitement avec deux unités est de $36ms$ sur chacune des unités.

IV.8 Performance Mémorisation–Assemblage{ *Tmb–Entité* }–FT9000

IV.8.1 Introduction

Par ces mesures, on souhaite comprendre comment le système se comporte lors que sur les unités de traitement on n'effectue aucun traitement et que l'on transfert l'événement complet sur le module *FT9000*. Le lien entre la ferme de traitement et le module *FT9000* à une longueur de 10 m.
Le tableau IV.14 présente les conditions de mesures.

Site	Opérateur	Nb Tâches	Précision
12 *Tmbs*	Mémorisation	1	
	Distribution	1	Transmission des partitions en *série*
	16 emplacements situé dans la *SRAM 32 bits non-cache* Code situé dans la *DRAM 32 bits cache 16 KOctets*		
NbUnit Unité	Rassemblement	1	Réception des partitions en *parallèle*
	Distribution	1	Transmission des partitions en *série*
	2 emplacements situé dans la *DRAM 32 bits cache 16 KOctets* Code situé dans la *DRAM 32 bits cache 16 KOctets*		
1 *FT9000*	Rassemblement	*NbParEvent*	
	64 emplacements situé dans la *DRAM 64 bits cache 16 KOctets* Code situé dans la *DRAM 64 bits cache 16 KOctets*		

TAB. IV.14 - *Conditions de mesure de performances Mémorisation– Assemblage*{Tmb–Entité}–*FT9000*

On ajuste les paramètres suivants:

- le nombre d'unités présentes dans la ferme de traitement: *NbUnit*,
- le nombre d'événements reçus en *parallèle* sur le site *FT9000*: *NbParEvent*.

on relève pour chaque mesure:

- le temps moyen d'assemblage complet $< T_{server} >$,
- le temps moyen d'assemblage des données $< T_{entity\_prd} >$,
- les temps T_{ft9000_prhd} etT_{entity_pshd} pour la transmission vers le module *FT9000*,
- le temps mort et la fréquence d'acceptation d'injection Fz_{max}.

IV.8.2 Performance à une unité de traitement

Le tableau IV.15 donne les mesures obtenues. Comme l'on dispose de 2 emplacements, les opérations de rassemblement et de distribution vers le *Ft9000* peuvent s'effectuer en parallèle. La tâche de rassemblement peut effectué une nouvelle demande d'événement pendant que la tâche de distribution transfert l'événement vers le site *FT9000*.

On constate que:

- le temps mort est constant tant que le taux de déclenchement est inférieur à $Fz_{max} = 275.7Hz$.
- le taux de déclenchement maximal est limité par le module *FT9000* puisque pour $Fz \geq Fz_{max}$ le minimum de entre T_{entity_pshd} et T_{ft9000_prhd} est donné par T_{ft9000_prhd}.
- pour $Fz \leq Fz_{max}$, T_{server} correspond au temps d'assemblage complet, puis pour $Fz \geq Fz_{max}$ T_{server} inclut le temps d'assemblage complet et une partie du temps de transfert vers le *FT9000*.

Fréquence (Hz)	T_{server} (μs)	T_{entity_prd} (μs)	T_{entity_pshd} (μs)	T_{ft9000_prhd} (μs)	Fz_{accept} (Hz)	Temps mort (μs)
19.98	1849	1540	2036	3227	19.98	356
40.06	1849	1540	2071	3195	40.06	356
80.18	1846	1540	2092	3270	80.21	356
160.1	1856	1542	2117	3281	160.1	356
240.6	2441	1616	4130	3102	240.6	356
323.6	3519	1504	3610	2574	275.7	2046

TAB. IV.15 - *Mesures Mémorisation-Assemblage* {Tmb-Entité}*-Ft9000 à une unité*

Intérêt du parallélisme sur l'unité de traitement

Si sur l'unité de traitement on ne dispose que d'un seul emplacement, les opérations de rassemblement et de distribution sont alors séquentielles. Dans ce cas le taux de déclenchement maximun est de $242.5Hz$, il correspond au chaînage de l'ensemble des opérations. Si l'on estime la somme entre:

- le temps de mémorisation maximal: $250\mu s$,
- le temps d'assemblage de l'évément donné par T_{server} à vide: $1850\mu s$
- le temps de transfert vers le *FT9000* donné dans ce cas par T_{entity_pshd} à vide: $2036\mu s$

on obtient un temps de $4136\mu s$ peut différent de $1/242 = 4132\mu s$.
Le fait d'autoriser le parallélisme sur l'unité de traitement entre les opérations de rassemblement et de distribution apporte un gain en temps de $500\mu s$.

IV.8.3 Performance à deux unités de traitement

Lorsque la ferme de traitement est constituée de deux unités, nous avons autorisé le site *FT9000* à recevoir simultannément les événements provenant des deux unités. Nous avons pu vérifier que le système accepte un taux de déclement d'au moins $323Hz$ sans introduire de temps mort supllémentaire aux $336\mu s$ de létage d'entrée. Cependant pour des taux de déclenchement supérieur, le système casse pour des raisons que le manque de temps et disponibilité du site ne nous ont pas permis d'approfondir.

IV.9 Conclusions

Concernant l'assemblage d'événements:

- La bande passante de transfert mesurée sur les liens est de 4.44 Moctets/s sur un lien, de 8.58 Moctets/s sur 2 liens. La bande passante mesurée sur les liens est toujours supérieure à plus de 90% de la valeur maximale attendue. Elle ne dépend ni du nombre de blocs assemblés ni de leur taille, ce qui confirme le *"bon fonctionnement"* du routeur *C104*.
- le temps d'assemblage est constitué de 3 composantes:
  1. le temps de réception des en–têtes ($330\mu s$) fonction du nombre de sources et du nombre de partitions sur les sources,
  2. le temps d'initialisation logiciel ($1ms$) fonction du nombre de partitions à assembler,
  3. le temps de transfert sur les liens ($491\mu s$) fonction de la taille de l'événement.
- le système d'assemblage, pour 76 blocs de données correspondant à un événement de 4400 octets, sature à
  - 420 Hz pour une ferme constitué d'une seule unité sans effectuer de traitement,
  - 530 Hz pour une ferme constitué de deux unités sans effectuer de traitement,
- le gestionnaire de flux de données *Mbm* fonctionne correctement et permet l'utilisation des différents parallélismes présents sur un *Transputer*.
- Le code d'assemblage est écrit en langage C. Il est structuré:
  - pour permettre différentes possibilités de multi–distribution et s'adapter à des traffics de données plus élévés que celui de l'expérience L3.
  - pour permettre l'assemblage de différents formats d'événements construit à partir de pointeurs

  Les paramètres d'assemblage sont gérés par un tableau, une modification de format se résume au changement d'éléments dans ce tableau.
- La technique d'assemblage est optimisée, toutefois il est possible d'améliorer les temps:
  - en utilisant deux mémoires sans zone commune (*cache*) l'une affectée aux données, l'autre réservée au code, on a montré qu'on réduit le temps d'assemblage de presque 20 %,
  - en réduisant le nombres de processus montés en parallèle,
  - en essayant de masquer le temps logiciel d'initialisation.
- La comparaison des performances avec celles mesurées dans l'environnement *XOP*[4] montre:
  1. le temps de transfert sur 2 liens du *Transputer* est voisin du temps d'occupation sur le bus *FASTBUS* en mode *pipeline*.
  2. la structure en tranches de processeurs *XOP* permet de masquer le temps logiciel d'initialisation ainsi que l'envoi préalable du nombre de mots. Dans les mesures effectuée, le temps logiciel d'initialisation n'est pas masqué ce qui explique que *XOP* est 3 à 4 fois plus rapide.
     L'assemblage multi–événements permet de masquer une partie du temps logiciel d'initialisation, cependant les mesures doivent être approfondies.

  Toutefois, on compare un microcode spécialisé modifiable uniquement par un expert à un code en langage C beaucoup général et très souple.

Concernant le déclenchement de niveau–2

- le temps mort susceptible d'être introduit dans l'expérience est indépendant du taux de déclenchement jusqu'à saturation et vaut $356\mu s$. Comme il est inférieur au temps mort des autres détecteurs ($\geq 500\mu s$), le système de déclenchement de niveau–2 n'introduit pas de temps mort dans l'expérience.
- les mesures montrent que les temps de mémorisation, d'assemblage, de traitement et de transfert vers l'interface *FT9000* normalisés par événement sont indépendants du taux de déclenchement jusqu'à saturation.
- le taux de déclenchement supporté par la ferme de traitement est proportionnel au nombre d'éléments tant que ce taux de déclenchement reste inférieure au taux de déclenchement maximum de $275Hz$ supporté par l'interface *Ft9000*.
- Le système est entièrement piloté par des langages de haut niveau, C pour l'assemblage, FORTRAN pour les algorithmes, qui apportent la souplesse de fonctionnement aussi bien pour modifier les algorithmes que pour adapter la configuration.
- Le tableau IV.16 précise les performances du système de déclenchement de niveau–2 en rappelant le cahier des charges qu'il doit satisfaire.

Ferme de Traitement	Temps	Taux de déclenchement supporté sans ajouter de temps mort		Cahier des charges
		1 Unité	2 Unités	
Entrée *FIFO*	$11/12\mu s$	$45/90KHz$		$45/90KHz$
Mémorisation	$356\mu s$	–		$\leq 500\mu s$
Assemblage	$\cong 1.8ms$	$420Hz$	$440/530Hz$	$\leq 100Hz$
Traitement	$1-10ms$	$75Hz$	$150Hz$	$\leq 100Hz$
FT9000	$\cong 2ms$	$275Hz$	$\geq 323Hz$	$\leq 100Hz$

TAB. IV.16 - *Performances du système de déclenchement de niveau–2*

- Le système installé satisfait le cahier des charges de l'expérience L3.

Chapitre V

Intégration du déclenchement de niveau–2

V.1 Intégration dans le système d'acquisition de L3

L'intégration au sein de l'ensemble de la configuration expérimentale doit permettre deux modes de fonctionnement:

1. un fonctionnement en mode global qui permet d'assurer le contrôle du système durant les périodes de prise de données.
2. un fonctionnement en mode local destiné à effectuer les calibrations et les tests matériels

Mode global Dans ce mode, le système est intégré au sein du système d'acquisition de l'expérience. Celui-ci comprend de nombreuses tâches s'exécutant sur différentes machines d'un "*cluster*" VAX. Chaque tâche est dédiée à une partie du système d'acquisition qui peut être un sous-detecteur ou un niveau de déclenchement. En mode de fonctionnement global, le contrôle de toutes ces tâches est centralisé et s'effectue depuis un contrôleur général. Celui-ci est une tâche intéractive qui permet de choisir les conditions de la prise de données, de générer toutes les commandes qui permettent d'initier et de synchroniser la prise de données. Les commandes sont transmises vers les superviseurs des différents sous-systèmes qui gèrent les réponses et messages. L'ensemble est hiérarchisé de sorte que le niveau-2 dispose d'une tâche de contrôle en liaison avec le contrôleur du système de déclenchement.

Mode local En dehors des périodes de prises de données, chaque sous-détecteur peut fonctionner en mode local pour effectuer selon les besoins des calibrations ou des tests matériels. Ainsi le déclenchement de niveau 2 peut effectuer localement et de façon autonome un certain nombre de tests qui garantissent son bon fonctionnement, indépendamment des processeurs situés en amont et en aval. Dans ce mode, le système de communications tâche à tâche avec le contrôleur général est désactivé et les différentes tâches de contrôle dédiées à chaque partie de l'appareillage sont pilotées localement.

Le contrôle en ligne d'un système recouvre à la fois son pilotage et sa surveillance. Par pilotage, on entend la partie du logiciel de contrôle en ligne qui permet de soumettre le système matériel aux différentes phases de l'acquisition: démarrage à froid, initialisation, début et fin de la prise de données mais aussi d'adapter son comportement aux conditions de la prise de données qui peuvent varier. Pour le déclenchement de second niveau, certains paramètres peuvent être modifiés comme la référence du code chargé dans les *Transputers*, l'activation des algorithmes de sélection, les facteurs d'échantillonnage (notamment l'échantillonnage d'événements qui devraient être rejetés) etc...

Dans ce mode, le système est capable:

- d'une part, de verrouiller l'information générée en amont,
- d'autre part, d'injecter une information simulée, contrôlée par le système.

Ces deux fonctions sont assurées au niveau des modules *Tmbs* permettant d'effectuer les tests *in sitú*.

V.2 Contrôle en ligne du niveau 2

Le contrôle en ligne du Niveau 2 est basé sur une architecture qui respecte le cahier des charges du contrôle général de l'expérience et reprend pour l'essentiel la technique d'intégration antérieure. Il intégre à la fois les fonctionalités de pilotage et de surveillance du système.

Pour l'essentiel, il s'agit de détecter les commandes émises par le contrôleur général de prise de données (`CREATE`, `COLD`, `INIT`, `START`, `STOP`). Selon les cas, ces commandes sont traitées par des tâches Unix implantées sur la station *l3sunlv2* (initialisation, chargement du réseau de

Transputers) ou bien transmises au réseau de *Transputers* (démarrage et arrêt de la prise de données).

Il est prévu deux modes de fonctionnement global et local utilisables selon que le Niveau 2 est intégré ou non dans le système d'acquisition de l'expérience. Le mode local permet de piloter le Niveau 2 depuis la station *l3sunlv2*.

Dans le contexte du nouveau Niveau 2 (cf figure V.1), la tâche chargée d'opérer sur le système est un processus Unix (noté `J_RUNT2A`). En mode global, une tâche de contrôle VAX/VMS dédiée au niveau 2 (notée `J_RUNL2A`) est chargée de détecter les commandes reçues par le contrôleur du système de déclenchement `J_RUNCO` et de les transmettre au serveur Unix *j2svr* vers la tâche de contrôle du Niveau 2. A ce niveau, on utilise une communication entre tâches de type client–serveur servie par le protocole TCP [1].

En mode local, les commandes sont émises par une tâche Unix intéractive et transmises à la tâche de contrôle *j_runt2A*. Cette dernière est commune aux deux modes de fonctionnement. Elle est relayée par un service *j_t2run*, chargé de communiquer avec l'ensemble des processus dédiés au pilotage qui se trouvent répartis sur le réseau de *Transputers*.

Dans le monde Unix, les communications sont assurées par les *IPC*s [2] classiques ici des tubes nommés.

L'architecture des logiciels de contrôle et de surveillance du réseau est distribuée et identique sur tous les sites *Transputers*. Elle est basée sur le protocole AServer qui permet d'établir les communications nécessaires entre le monde *Transputer* et le monde UNIX. Par système distribué, on entend que sur chaque *Transputer*, un processus est dédié l'un au contrôle (échange commandes–réponses avec le contrôleur de run), l'autre à la surveillance (détection et renvoie des erreurs et statistiques). Côté *Transputer*, ces processus sont des clients AServer. Côté UNIX, deux services sont respectivement dédiés au contrôle et à la surveillance.

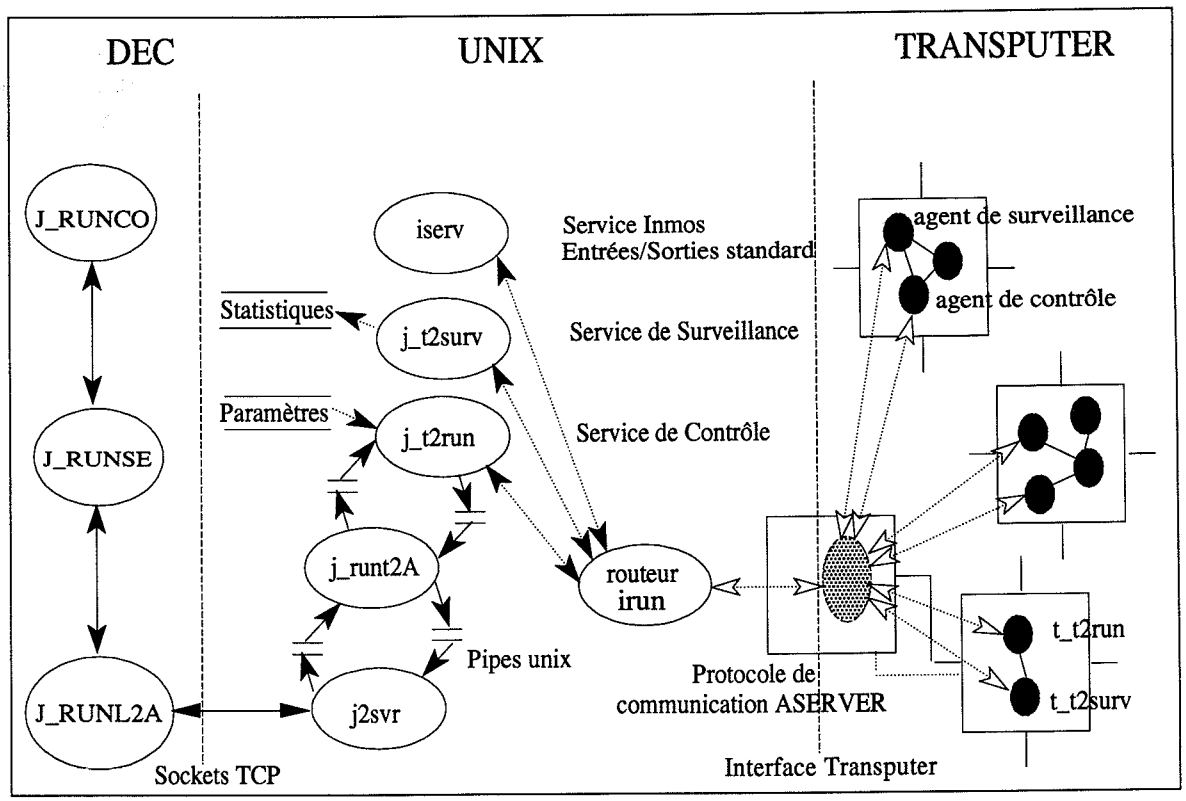


FIG. V.1 - *Intégration et Architecture du logiciel de contrôle en ligne (mode global)*

1. Transmission Control Protocol
2. Inter Process Communication

Le logiciel de contrôle en ligne est bâti au dessus d'une couche de communications qui sert les fonctionnalités de contrôle. La couche applicative proprement dite assure l'interprétation des commandes en ligne et l'éxécution des actions correspondantes.

Deux protocoles de communication coéxistent au sein du logiciel: les communications entre processus unix et les communications vers le système de *Transputer*s assurées par le protocole AServer.

V.3 Communications entre processus Unix

Elles sont assurées par le mécanisme de base que sont les tubes nommés au dessus desquels on définit une connexion adaptée aux besoins de notre application.

Une connexion est constituée de deux tubes (*pipes*) unidirectionnel, le choix de deux *pipes* unidirectionnels permet de séparer la voie réservée aux commandes de celle réservée aux réponses.

Chaque processus doit établir les connexions qui lui sont utiles pour communiquer. Deux processus qui communiquent ensemble partagent bien évidemment les mêmes tubes nommés. Si d'un côté le tube est considéré en écriture, de l'autre, c'est un *pipe* de lecture et vice versa.

Les messages qui circulent à travers ces tubes respectent le protocole défini par un ensemble de fonctions de base. La routine d'initialisation *cp_init_connection* permet d'associer deux tubes nommés pour constituer une connexion entre deux processus. Les routines *cp_open_client* et *cp_open_service* permettent respectivement au processus client situé en amont et au processus service situé en aval d'ouvrir la connexion. On dispose ensuite des routines de base pour effectuer des transmissions formatées ou non selon les besoins. Les routines de transmission formatée tiennent compte du format des commandes et des réponses en différents champs. Les routines de transmission non formatées permettent quant à elle l'échange de messages quelconques. On dispose également d'une routine de fermeture de connexion *cp_close_connection_pipe*.

V.4 Communication Unix-*Transputer*s: le logiciel *AServer*

Le protocole de communication AServer fourni par Inmos permet d'établir les communications nécessaires entre le monde *Transputer* et le monde hôte.

Le logiciel AServer comprend une collection de programmes, librairies et protocoles qui ensemble permettent aux applications *Transputer*s d'accèder des services d'une façon consistante et extensible (cf figure V.2). AServer inclut notamment:

- les spécifications du protocole de communication,
- deux routeurs de messages AServer (côté hôte et côté *Transputer*) ,
- deux librairies d'interface logicielle vers les routeurs AServer,
- un mécanisme d'appel de procédures distantes (*RPC*),
- un format de représentation de données (*ASDR*),
- un outil de compilation du protocole (*iasgen*),
- un convertisseur du protocole de communication antérieur (*isconv*),
- ainsi qu'un certain nombre de services côté Unix: *autoiserver* (pour la compatibilité des développements antérieurs), *iserv* pour les entrées-sorties standards et un service de déverminage.

Il s'agit d'un logiciel constructeur ouvert qui autorise le développement de services Unix spécifiques, adaptés aux besoins de clients implantés dans le monde *Transputer*. On utilise pour cela les deux librairies de développement fournies qui permettent, de chaque côté, d'interfacer des

processus–utilisateur vers chacun des routeurs de messages AServer. Ces processus permettent d'interfacés les liens vers l'exterieur du réseau et jouent le rôle de multiplexeur–démultiplexeur bidirectionnel de messages AServer. Ils sont également chargés de router les communications AServer vers l'endroit approprié: vers un autre processus ou un autre routeur. En plus de cette fonctionalité, il faut ajouter que le routeur présent sur le système hôte permet le démarrage et le contrôle réseau. Sur le réseau de *Transputers*, les processus utilisent des canaux virtuels pour communiquer avec le routeur. Sur la machine hôte, les communications avec le routeur sont assurées par les mécanismes de communication entre processus offerts par le système hôte, ici les *IPCs* d'Unix.

L'ensemble logiciel AServer a permis le développement des deux services dédiés au pilotage et à la surveillance du Niveau 2. Dans le monde *Transputer*, chaque fonction est distribuée. Ainsi chaque *Transputer* abrite un processus client, véritable "agent" vis à vis de la fonction recherchée. Le même type d'architecture est utilisé pour le système de déverminage proposé par Inmos. De façon similaire, on trouve sur chaque *Transputer* un processus client: agent de déverminage. Côté Unix, un service AServer joue le rôle de contrôleur de l'ensemble du réseau en servant et controlant tous les processus agents. Dans notre cas, les mécanismes de pilotage et de surveillance bien que similaires sont indépendants.

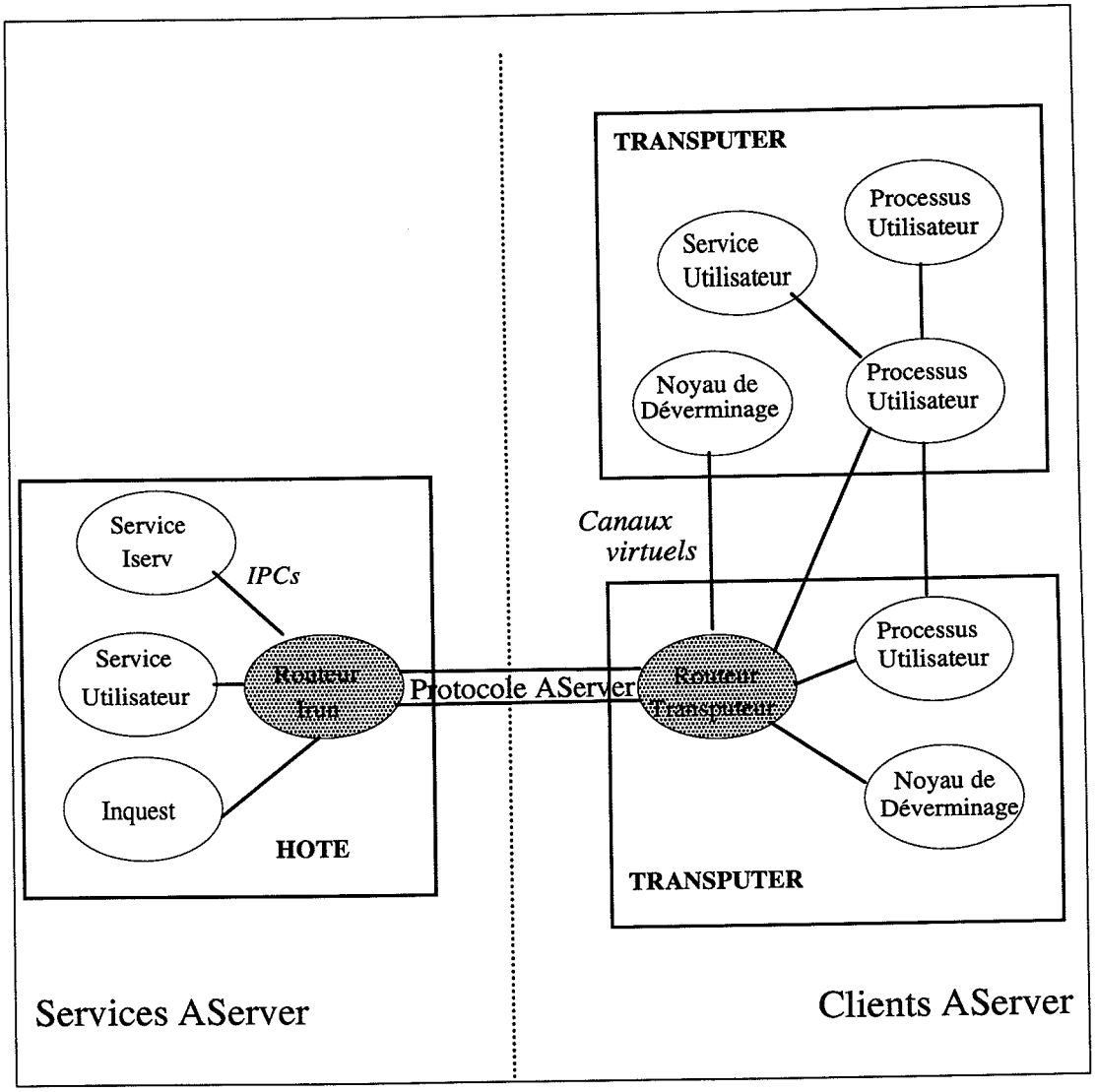


FIG. V.2 - *Utilisation du logiciel AServer*

En mode local ou global, l'architecture prévoit deux processus clients *t_t2run* et *t_t2surv* sur

chaque *Transputer* chargés l'un du contrôle et du pilotage, l'autre de la surveillance. Côté Unix, les deux services *j_t2run* et *j_t2surv* sont respectivement dédiés au contrôle et à la surveillance du Niveau 2 (cf figure V.1).

V.5 Pilotage du Niveau 2

Les principales étapes d'une prise de données qui conditionnent le pilotage du système sont:

- le lancement des tâches de pilotage initiée par une commande CREATE,
- le démarrage à froid initié par une commande COLD,
- l'initialisation du système initié par une commande INIT,
- le début de la prise de données marqué par une commande START,
- la fin de la prise de données marquée par une commande STOP.

Les succès des différentes commandes définissent les états CREATE-SUCC, COLD-SUCC, INIT--SUCC, START-SUCC et STOP-SUCC du système.

V.5.1 Protocole et Format des commandes

Le protocole d'échange des commandes et des réponses est conforme à celui établi pour le système actuel, à savoir:

- émisssion d'une commande: COMMAND,
- accusé de réception: ACKNOWL,
- confirmation d'exécution: COMMAND-ACTI,
- compte-rendu d'exécution: COMMAND-STATUS.

Le format des messages échangés est le suivant. Il s'agit de chaînes de caractères comprenant plusieurs champs d'information FIELD1/FIELD2/../FIELDn//:

- une commande est une chaîne de caractères du type 'COMMAND//', où COMMAND est le mnémonique de la commande (CREATE, COLD, INIT, START, STOP)
- une réponse est une chaîne de caractères du type 'FIELD1/FIELD2//', où FIELD1 est le mnémonique de la réponse (ACKNOWL, COMMAND-ACTI, COMMAND-SUCC, COMMAND-ERROR), FIELD2 l'identificateur du *Transputer* dans le cas d'une réponse à l'une des commandes INIT, START, STOP.

V.5.2 Etat CREATE-SUCC

La commande CREATE permet de démarrer les serveurs de communication J_RUNL2D, côté VMS, et *j2svr*, côté Unix, ainsi que la tâche de contrôle local *j_runt2A*. Cette étape assure la mise en place des mécanismes de contrôle en ligne (cf figure V.3).

En mode local, on se contentera de démarrer *j_runt2A* depuis la tâche intéractive locale.

Par lancement d'une tâche, on entend la création d'un processus mais aussi la mise en place des canaux de communication entre le processus amont et le processus créé. Ainsi, la création de *j_runt2A* s'accompagne de la création s'ils n'existent pas déjà des tubes de communication *commandes* et *interrupts* pour le mode global, *commlocal* et *interlocal* pour le mode local. Le succès de cette étape est garanti par le renvoi d'une réponse CREATE-SUCC émise par *j_runt2A*

vers la tâche *j2svr* et l'univers DEC ou bien vers la tâche de contrôle locale selon le mode de contrôle.

Sur CREATE-SUCC, le monde *Transputer* est invisible. Les serveurs de communication et la tâche de contrôle du Niveau 2 sont opérationnels. Le niveau 2 est prêt à recevoir les commandes émises par les contrôleurs de prise de données L3RUN_CO ou J_RUNCO.

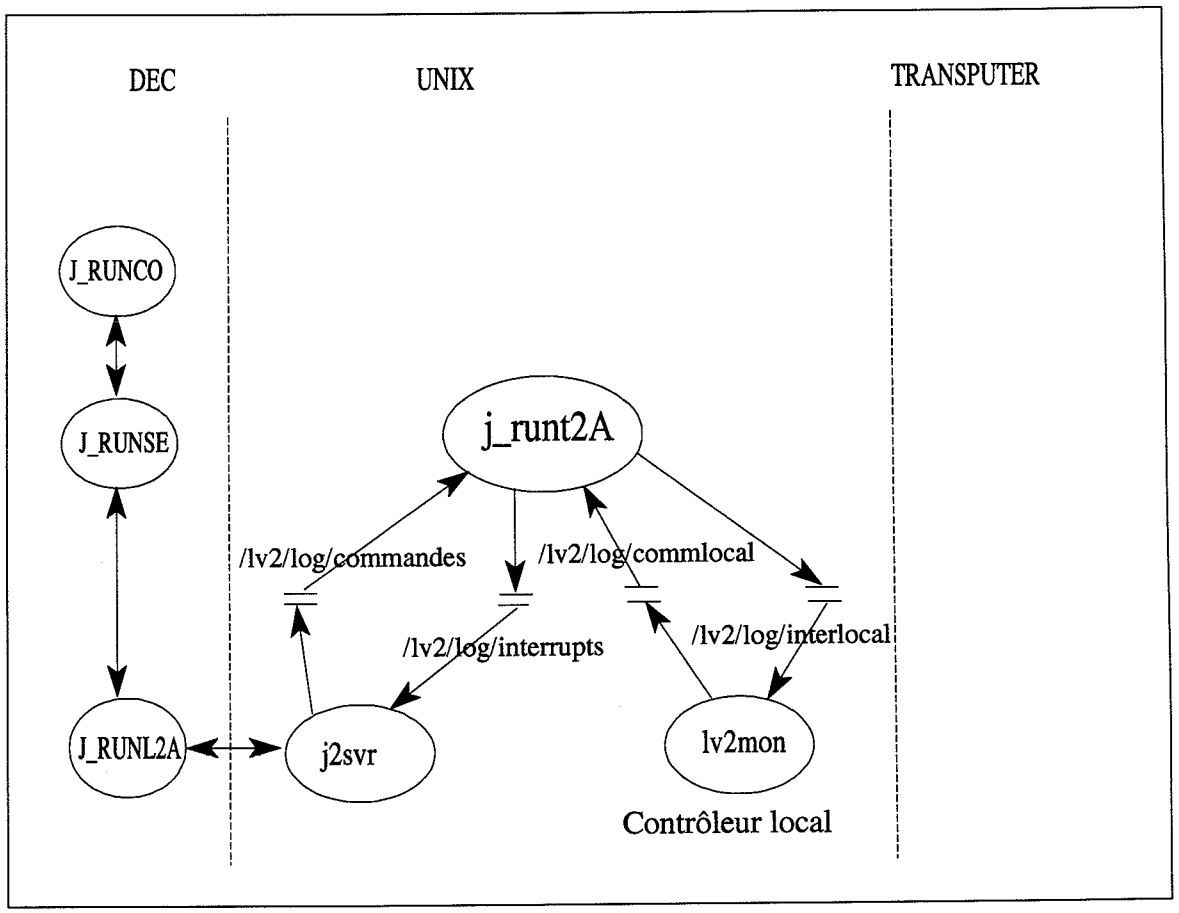


FIG. V.3 - *Etat* CREATE-SUCC

V.5.3 Etat COLD-SUCC

La commande est reçue par la tâche *j_runt2A* en vertu du mécanisme mis en placeau moment de l'étape de création.

Il est effectué:

- une remise à zéro du réseau de *Transputer*,
- un test préliminaire de la configuration *Transputer*,
- le lancement du routeur Aserver (*irun*),
- l'initialisation complète du réseau de *Transputer*s et le chargement du code.

Dès lors les processus de pilotage *t_t2run* et de surveillance *t_t2surv* deviennent actifs et établissent leur connexion respective avec les services *j_t2run* et *j_t2surv*. La création de *j_t2run* s'accompagne de la création (s'ils n'existent pas déjà) des tubes *writetra* et *readtra* (cf figure V.4). Sur chaque *Transputer*, les deux processus *t_t2run* et *t_t2surv* sont opérationnels.

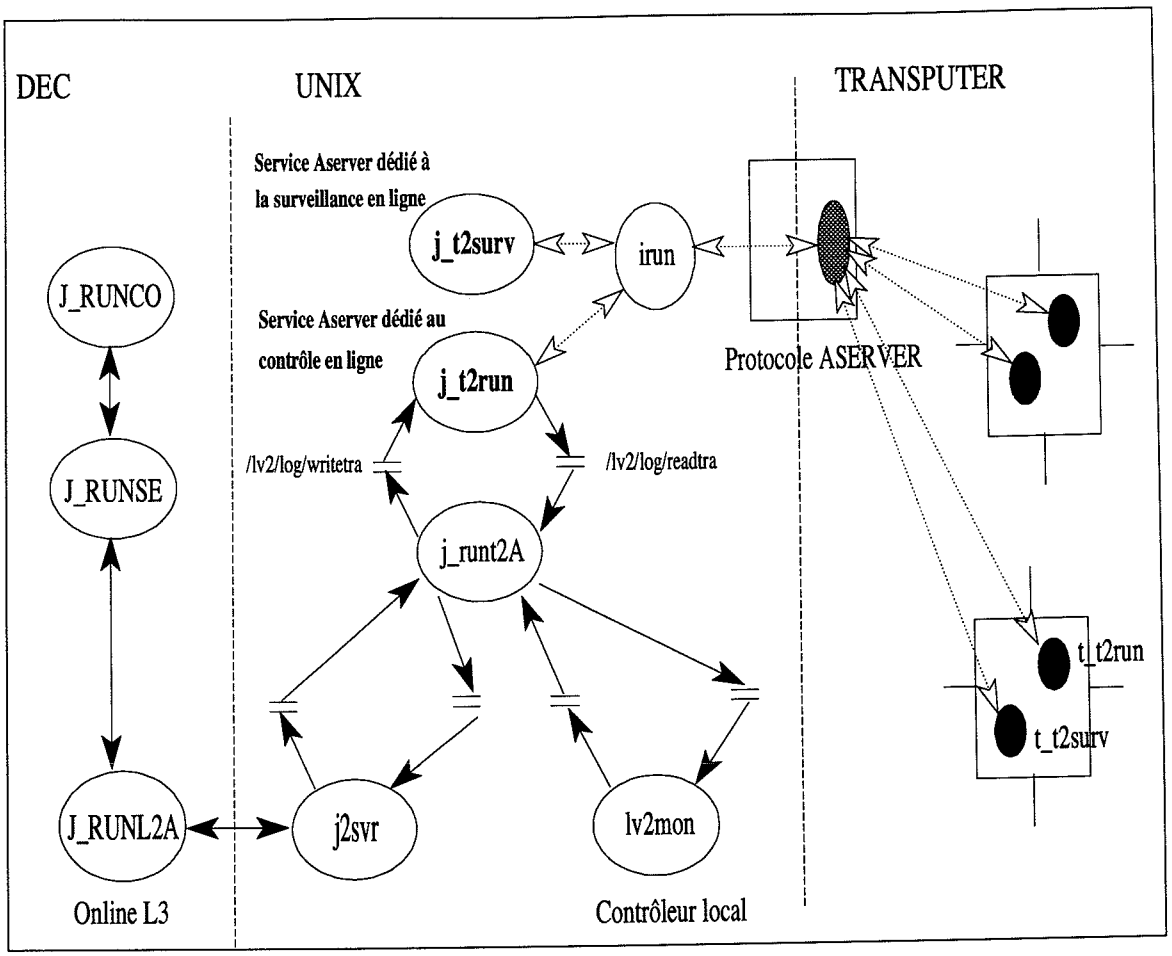


FIG. V.4 - *Etablissement des connexions AServer sur* COLD

Le processus applicatif est également activé. Il peut exécuter une partie de code correspondant à l'initialisation à froid de sa tâche, typiquement: initialisation des ressources mémoire *Mbm*, des tâches et de leurs espaces de travail.

Le service *j_t2run* envoie ensuite deux sous–commandes qui permettent:

- en outre de tester les échanges de commandes avec chacun des sites sous contrôle.
- la construction du chaînage de propagation des signaux de contrôle expérimentaux entre les modules *Tmbs*.
- puis le démarrage de tâches *Opérateur* sur l'ensemble des sites: toutes ces tâches sont ensuite synchronisées sur l'arrivée des données expérimentales.

Au terme de cette étape, toutes les tâches de contrôle en ligne, de surveillance et applicatives sont lancées. Toutes les communications sont établies. Le réseau de *Transputers* est opérationnel, prêt à recevoir les commandes `INIT`, `START`, `STOP` qui pilotent réellement la prise de données.

Le succès de cette étape est garanti par le renvoi d'une réponse `COLD-SUCC` émise par *j_t2run* à l'adresse de *j_runt2A*. Dans l'état `COLD-SUCC`, l'applicatif est lancé et se trouve en attente d'événements (cf figure V.5).

V.5.4 Etat `INIT-SUCC`

La commande est reçue par la tâche *j_t2run* en vertu du mécanisme mis en place au moment du `COLD`. Elle est transmise en diffusion à tous les *Transputers*, réceptionnée et traitée sur chaque nœud par *t_t2run*.

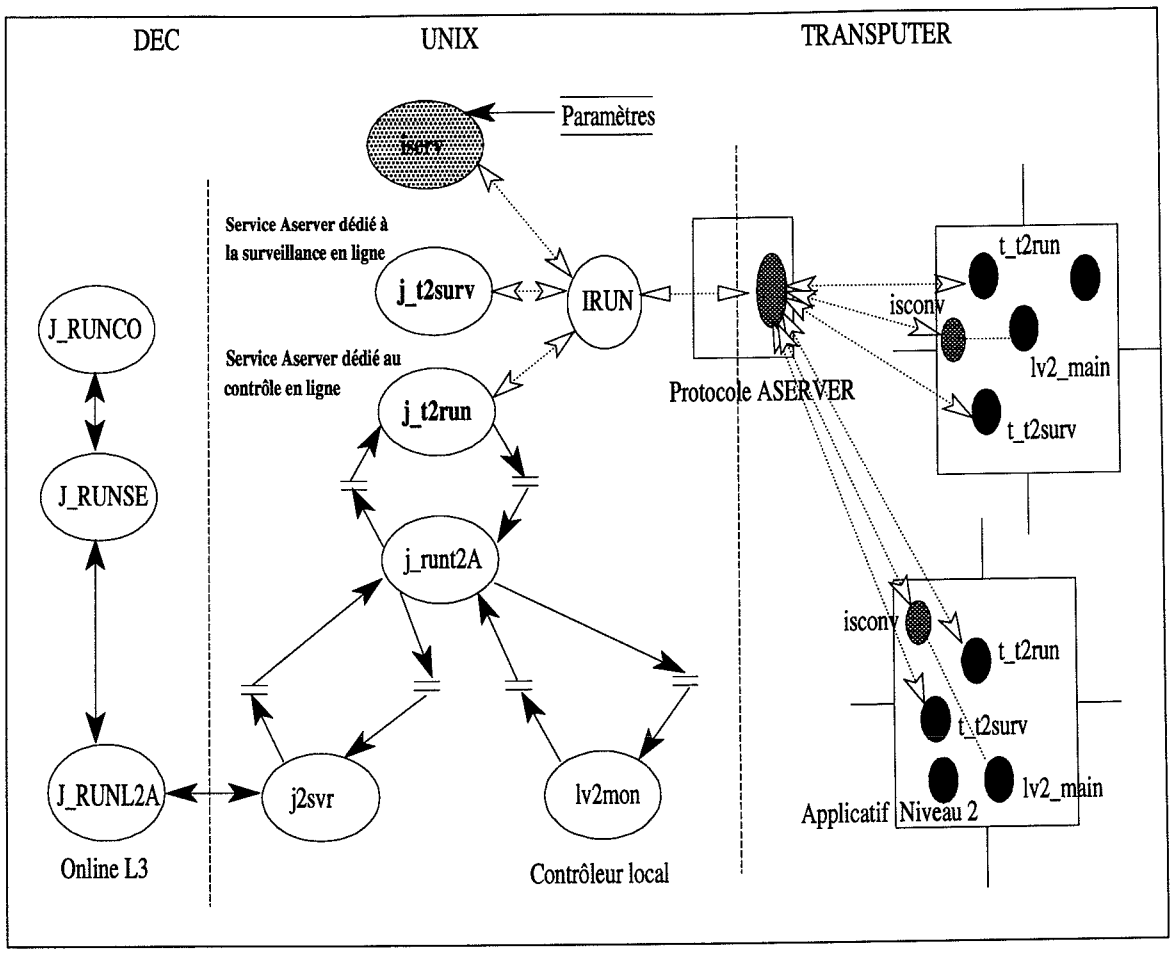


FIG. V.5 - *Etat* COLD-SUCC

Il est effectué:

- la mise à jour des paramètres immédiats (lecture d'un fichier UNiX) et la relance de certaines opérations de démarrage à froid le cas échéant,
- des opérations d'initialisation et de remise à zéro de ressources partagées,
- remise à zéro des compteurs et des échelles statistiques.

Ces opérations sont assurées par le processus *t_t2run* qui réceptionne la commande et exécute une routine dédiée. Le succès de cette étape est garanti par le renvoi d'une réponse `INIT-SUCC` émise par *j_t2run* à l'adresse de *j_runt2A*.

V.5.5 Etat `START-SUCC`

La commande est reçue par la tâche *j_t2run* en vertu du mécanisme mis en place au moment du `COLD`. Elle est transmise d'aval en amont du site *FT9000* jusqu'aux sites *Tmbs*.

A la réception de la commande `START`, il convient de démasquer le signal d'*Event Request* qui permet aux mémoires d'entrée du Niveau 2 de se synchroniser sur les signaux de l'expérience et d'autoriser la synchronisation avec le système d'assemblage central sur le site *ft9000*.

Ces opérations sont assurées par le processus *t_t2run* qui réceptionne la commande et exécute une routine dédiée. Le succès de cette étape est garanti par le renvoi d'une réponse `START-SUCC` émise par *j_t2run* à l'adresse de *j_runt2A*.

V.5.6 Etat STOP-SUCC

La commande est reçue par la tâche *j_t2run* en vertu du mécanisme mis en place au moment du *COLD*. Elle est transmise d'amont en aval des sites *Tmbs* au site *FT9000*.

A la réception de la commande STOP, il convient:

- de masquer le signal d'*Event Request* sur les sites *Tmbs*,
- de masquer la synchronisation avec le système d'assemblage central sur le site *Ft9000* pour permettre la vidange des événements résidant dans le niveau–2 lors de l'exécution de la commande STOP sur un disfonctionnement du système d'acquisition,
- de vidanger les compteurs et échelles statistiques.

Ces opérations sont assurées par *t_t2run* par l'intermédiaire d'une routine dédiée. Le succès de cette étape est garanti par le renvoi d'une réponse STOP-SUCC émise par *j_t2run* à l'adresse de *j_runt2A*.

V.6 Gestion des paramètres immédiats

On appelle paramètres immédiats, les paramètres dont la sélection est intéractive. Le choix de ces paramètres est proposé au niveau du contrôleur de prise de données. Il s'agit:

- des références du code Niveau 2 (code *Transputer*),
- des masques de contrôle des algorithmes (CNTRL_WORD, LV2_REJECT),
- des facteurs de prescaling etc...

La modification des paramètres immédiats est prise en compte sur INIT. Dans certains cas, un redémarrage à froid (COLD) peut se révéler nécessaire notamment si la référence du code Niveau 2 est modifiée.

Chaque *Transputer* a accès au fichier qui tient à jour les paramètres du run. On utilise pour cela le service *iserv* (service AServer dédié aux I/O standard et fourni par Inmos) côté hôte et le convertisseur iserver *isconv* côté *Transputer*. Toutes les opérations d'entrées/sorties standards en C telles que *printf, fopen, fread,...* de la *Run time C* sont servies de la même manière et accessibles par n'importe quel *Transputer*.

V.7 Surveillance en ligne du Niveau 2

V.7.1 Cahier des charges et réalisation

Le système de surveillance en ligne est un logiciel dont les fonctions essentielles sont:

- de contrôler la qualité des données accumulées,
- de veiller au bon fonctionnement du rejet en ligne des événements,
- de détecter un éventuel mauvais fonctionnement du mécanisme d'acquisition de données assuré par le Niveau 2.

Les deux premières fonctions font appel à une technique identique à savoir le traitement statistique de l'information. La troisième fonction est plus délicate à mettre en place car elle interfère directement avec le fonctionnement du système. Il convient à la fois de détecter et de signaler toutes erreurs fatales qui peut être a priori identifiées comme par exemple

- une erreur d'assemblage d'un événement,

- ou une incohérence des compteurs de signaux LV1 au niveau des modules *Tmb*s.

mais également d'accumuler des valeurs susceptibles d'être corrélées a posteriori pour permettre une évaluation globale de l'état du système. Ces observations permettront d'améliorer le système de surveillance lui même. Certaines valeurs pourront également être couplées à des alarmes en cas de dépassement de tolérance.

Le logiciel de surveillance en ligne du Niveau 2 inclut un schéma de report d'erreur et de surveillance statistique auquels sont adjoints un format de codage de l'information utile et des fonctionnalités d'archivage. L'ensemble est intégré au sein du système de contrôle en ligne.

Le report d'erreur et de la surveillance statistique du Niveau 2 sont assurés par un seul et même mécanisme qui inclut la génération et la détection d'événements survenant sur un site *Transputer*, le transport de l'information relative aux événements des différents sites à travers le réseau enfin l'analyse et l'archivage de l'ensemble des informations sur la station de travail UNIX.

Le terme d'événement doit être ici pris au sens large.

Un événement concerne indifféremment ce qui survient sur un site à un instant donné dont l'occurence est repérée et transmise vers l'extérieur. Il peut s'agir du report d'un compteur d'événements, de la détection d'une erreur de lecture par une carte *Tmb*, de la visualisation de l'état de remplissage d'une zone de mémoire tampon, d'une erreur d'éxécution, d'une erreur de protocole Aserver, de l'abandon d'un processus etc...

Les événements sont prédéfinis, aussi il est important que le système de surveillance ne se contente pas de détecter et transmettre les erreurs prévisibles. Il est nécessaire de prévoir le report d'un certain nombre d'informations d'état qui permet de cerner les erreurs imprévues. L'exécution de l'application parallèle est perçue à travers des événements qui donnent une image de l'état global du système. On aura aussi le souci de limiter la quantité d'information véhiculée.

Le mécanisme d'erreur et de la surveillance statistique du Niveau 2 est intégré au système de contrôle et prévoit l'envoi d'avis et d'alarmes vers le controleur de run. Côté *Transputer*, les tâches de surveillance sont totalement dissociées des tâches applicatives et fonctionnent de manière asynchrone pour permettre la poursuite de l'applicatif en cas d'échec du système de surveillance. Autrement dit, la génération d'un événement par une tâche applicative n'est jamais bloquante vis à vis de sa détection par une tâche de surveillance sauf en cas d'erreur fatale. Si besoin est, la possibilité de désactiver la surveillance de certains sites est envisageable.

On peut donc résumer le schéma de base retenu par quelques points–clés:

- Génération d'événement au sens large associés à une erreur ou une information de surveillance,
- Récolte à travers AServer par l'intermédiaire de processus clients asynchrones (mécanisme non bloquant vis à vis de l'applicatif excepté pour les erreurs fatales),
- Côté UNIX, traitement et analyse centralisés,
- Mécanisme d'archivage.

V.7.2 Evénements de surveillance

Il est prévu deux types d'événements. Les événements de type *USER* sont des événements spécifiés par l'utilisateur dans le cadre de son application. Les événements intrinsèques sont ceux générés par le système de surveillance lui même. Un dépassement de capacité sur les *Tmb*s sera de type `USER` car généré par une ou plusieurs tâches applicatives. Les erreurs occasionées par la *Run Time* Librairie C (division par zéro ou autre) sont intrinsèques car directement détectées par le système de surveillance. Dans ce schéma, les événements qui seront générés par des logiciels utilitaires (comme le logiciel MBM) sont également de type `INTRINSEQUE`.

Par ailleurs, on définit la nature d'un événement (EXE/DATA/PAR) selon qu'il concerne l'exécution d'un processus, le flux de données, ou l'interaction d'un processus avec un autre. Une condition d'exécution non remplie, un temps de préremption sont typiquement liées à l'éxécution d'une tâche alors que les dépassements de zone–tampon, les statistiques de physique concernent les flux de données. Enfin des informations liées aux communications (par ex: création d'un processus fils) sont liées au contexte applicatif distribué.

Deux fonctions spécifiques permettront de générer les événements USER: bloquante pour les erreurs fatales, non bloquante dans tous les autres cas.

Il est prévu de récupérer les signaux C générés par le système de recouvrement (*Trap Handler*) à la suite des exceptions ou erreurs d'exécution. Les informations issues de ce mécanisme seront étiquettées INTRINSEQUE-EXE.

N.B : Dans le cadre du Toolset INMOS, un système d'exécution (Run Time system) comprenant un système de recouvrement (Trap Handler) est installé par tâche définie au niveau de la configuration (c.a.d par main). A contrario, les processus créés dynamiquement (via ProcAlloc) partagent un seul et même système d'exécution (celui du main).

Il est également prévu de générer des événements de type INTRINSEQUE-PAR (pour signaler par exemple la naissance de processus fils au sein d'une tâche .b0x) de manière à surveiller l'ensemble de l'applicatif distribué sur le réseau.

Chaque événement est étiquetté en fonction:

- de son origine géographique: No de *Transputer*, qualité (Tmb, Unité de traitement, Ft9000 etc)...
- de son type USER/INTRINSEQUE,
- de sa nature EXE/DATA/PAR,
- de son origine (à préciser par un mnémonique adapté du type MBM-ALGO-ASERVER-SYS--RTL-HARD....),
- de son niveau de sévérité INFO/WARNING/ERROR/FATAL,
- du Message (ou id. équivalent) qui lui est associé.

V.7.3 Mise en œuvre

Le mécanisme de report asynchrone (warning, info ...) entre n'importe quel processus applicatif et le processus de surveillance *t_t2surv* utilise une boîte aux lettres. Il s'agit d'une zone–tampon en mémoire utilisée de façon non bloquante du côté de la génération des événements de surveillance. Ainsi on accepte de perdre certains événements si ces derniers sont surécrits dans la zone–tampon au profit d'événements plus récents et réactualisés. Le logiciel utilitaire MBM a été étendu en conséquence.

Le mécanisme de report synchrone (erreur fatale) ne peut tolérer de perte d'information et utilise quant à lui un canal de transmission d'erreur définie comme une ressource partagée par l'ensemble des processus présent sur un même *Transputer*. Le même canal interne est accessible à tous les processus par l'intermédiaire d'un sémaphore binaire.

Si besoin était, il pourrait être implanté un mécanisme de contrôle de flux en testant régulièrement la présence des chemins de communications entre *j_t2surv* et *t_t2surv*, comme entre tout couple client/serveur Aserver.

V.8 Structure du code *Transputer*

Tout code *Transputer* s'exécutant en ligne est bâti selon une structure bien précise qui autorise le pilotage et la surveillance de l'applicatif.

Le code en ligne inclut sur chaque nœud du réseau de *Transputers* un agent de pilotage et un agent de surveillance qui s'exécutent en parallèle (parallélisme concurrant) avec le reste de l'application.

t_t2run est connecté au service *j_t2run*. Il interpréte les commandes émises par le système de prise de données, effectue les actions correspondantes et retourne les acquitements, réponses, messages d'erreurs attendus par le système de contrôle amont. Il réceptionne également les avis d'erreurs considérés comme fatals pour le cours de la prise de données et qui en conséquence doivent être transmis au système de contrôle amont.

t_t2surv est quant à lui connecté au service *j_t2surv*. Il est chargé de recolter tous les messages destinés à la surveillance.

La partie principale (*main*) du code est commune sur tous les sites; c'est le squelette de l'application. Ce code prend en charge l'initialisation des communications AServer, l'établissement des connexions et le lancement des agents nécessaires pour le mode d'exécution considéré. En dernier lieu, la partie *main* lance le processus dédié à l'application proprement dite. Les deux connexions AServer nécessaires au contrôle en ligne sont ouvertes par le tt main vers *j_t2run* et *j_t2surv*. Trois processus sont alloués et initiés: *t_t2run*, *t_t2surv*, *proc_applic*.

Le mécanisme de lancement utilisé fait que les processus s'exécutent en parallèle, indépendamment et sans autres intéractions que celles prévues par la partie *main*. Il est bien entendu que le processus *application* pourra une fois lancé travailler sans aucune contrainte selon ses besoins ; c'est à dire récupérer des pointeurs sur des canaux externes définis dans le fichier de configuration, définir des canaux de communication internes, lancer des processus etc...

Il est à noter que les communications avec chacun des deux services Unix se font par l'intermédiaire d'un point d'accès dédié. Le logiciel AServer donne ensuite accès au descripteur de l'objet sur lequel il s'appuit pour réaliser le canal de communication. Comme nous l'avons indiqué, il s'agit d'un tube nommé, côté *host*, et d'un canal virtuel, côté *Transputer*. Le logiciel permet donc de récupérer le descripteur du fichier Unix ou l'identificateur de canal associé à un point d'accès. De part et d'autre, on dispose donc d'un mécanisme de scrutation cohérent qui inclut totalement les communications AServer et permet l'utilisation de fonctions telles que *select* et *ProcAlt*.

L'ensemble du code source (partie *main* et différents processus) une fois compilé et après édition de lien constitue une unité de code notée *lv2_unit*.

La figure V.6 précise comment le code Niveau 2 doit être configuré pour permettre les échanges AServer et le contrôle en ligne. L'unité *isconv* est placée entre le routeur et l'application. Elle autorise les entrées/sorties standards en cas de besoin. On notera que les points d'accès AServer sont initialisés dans le processus *main* pour être ensuite passés aux processus qui en ont besoin. Cette opération affecte en effet certaines variables statiques internes à la librairie d'exécution qui est partagée par tous les processus concurrents issus de la tâche initiale (*main*). Les points d'accès AServer utilisent des canaux externes définis entre le routeur et l'unité de code Niveau 2 (ici *lv2_unit*). Les canaux dits externes sont déclarés dans le fichier de configuration par opposition aux canaux internes alloués dans le code lui-même. Ils sont externes à l'unité de code Niveau 2 (sans être nécessairement externes au processeur).

V.9 Test et contrôle "*in situ*"

V.9.1 Les différents tests de la configuration

Différents tests ont été mis au point afin de s'assurer du bon fonctionnement du système avant chaque période de prise de données. Ces tests permettent d'exercer la configuration, matériel et logiciel confondus, et de s'assurer de l'état de fonctionnement du Niveau 2 de façon parfaitement autonome. Ces tests peuvent être exécutés independamment de l'état des autres parties du système d'acquisition, directement depuis la station de travail UNIX.

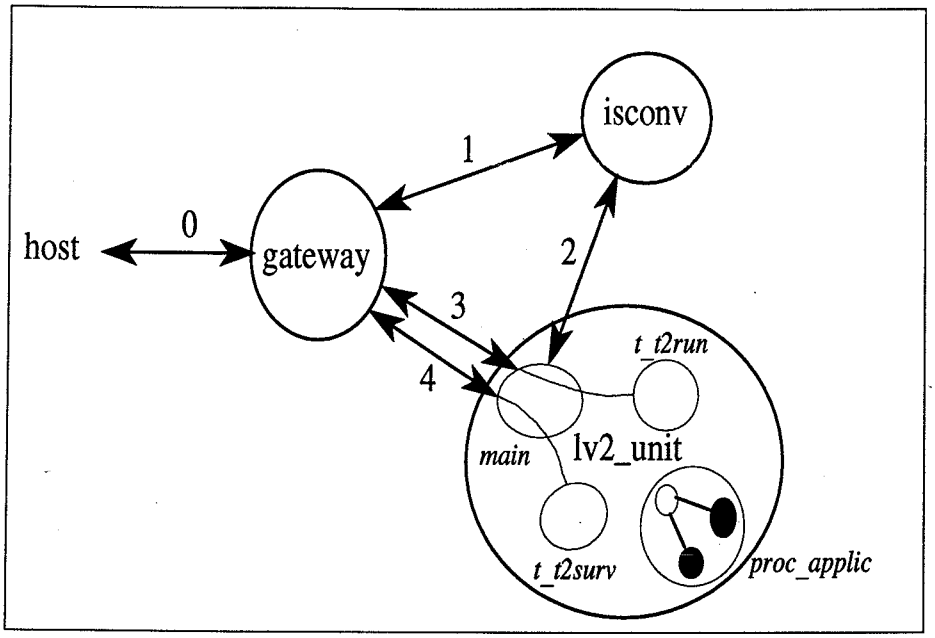


FIG. V.6 - *Configuration du code* Transputer *du Niveau 2*

Pour ce faire, l'information à l'entrée du Niveau 2 est verrouillée et les mémoires d'éntrée initialisées en mode test. Dans ce cas, les multiplexeurs d'entrée des modules TMBs (cf figure II.5) permettent la mémorisation des données provenant du transputer d'injection. La séquence d'acquisition est simulée par programme (génération des données et signaux de synchronisation) depuis la partie injection des TMBs. Selon les tests, les données ainsi que les nombres de mots par port sont simulés (configuration de bits ou données aléatoires) ou bien extraites d'événements expérimentaux issus de prises de données antérieures et rejoués à des fins de vérification. Dans le premier cas, on s'intéresse à des tests efficaces et contraignants du matériel. Dans le second cas, on procède à des tests complets vis à vis du logiciel puisque, s'agissant d'événements de physique, il est alors possible d'activer les algorithmes de sélection. Les différents tests élaborés constituent un ensemble cohérent qui permet d'exercer de proche en proche l'ensemble du système en poursuivant des tests de plus en plus complets.

Par ailleurs, le logiciel étant distribué, il convient de noter que les tests adjoignent des unités de code sur les sites d'injection (cf figure II.6) mais ne modifient en rien les unités de code placées sur les sites d'acquisition, de traitement. Le principe consiste donc essentiellement à simuler par programme les parties amont et aval manquantes. Avant l'injection des données en amont du système, il convient de générer l'information au format attendu par chaque module TMB et chacun des ports. En particulier pour rejouer des événements de physique, il convient d'effectuer un changement de format et d'éclater l'événement vers les différents sites d'injection. Ces fonctions sont assurées par une unité de code spécialisée dite de génération des données adjointe au logiciel. Cette unité de code est placée sur le Transputer relié au système hôte afin de faciliter l'accès aux fichiers d'événements expérimentaux stockés sur la station de travail UNIX. Ce même code est chargé a des fins de diagnostics de comparer l'information restituée après son passage à travers le réseau.

Les différents tests mis au point sont les suivants:

- un test de vérification du bon fonstionnement des mémoires d'entrée de la partie *Acquisition* des modules *Tmbs* avec "assemblage" jusqu'à l'unité Ft9000 sur des données simulées dit "Test à Grande Vitesse" ou TGV,

- un test sans les mémoires d'entrée jusqu'à l'unité Ft9000 de l'assemblage et du traitement

par les algorithmes sur des événements expérimentaux dit "Test sur des Evénements Vrais" ou TEV),

- • un test jusqu'à l'unité Ft9000 de l'assemblage et du traitement par les algorithmes sur des événements expérimentaux dit test EXPRESS,
- enfin un test complet identique au précédent avec l'écriture dans la mémoire partagée *Transputer–FASTBUS* dit test FASTBUS.

Pour utiliser ce dernier test, il convient bien entendu d'activer un programme VAX de lecture FASTBUS de la mémoire partagée *Transputer–FASTBUS* sur le module *FT9000*.

V.9.2 La tâche de surveillance de la prise de données et de contrôle en mode local

Mode local Le Niveau 2 est piloté depuis la station13sunlv2 par le contrôleur local. Ce programme dispose d'une interface Motif et propose à l'utilisateur toutes les commandes utiles au pilotage du système, la possibilité d'exécuter tous les utilitaires de test: remise à zéro et vérification du réseau de transputers jusqu'au test complet de la configuration du Niveau 2. Cette tâche intéractive notée lv2mon a une double fonction car elle assure également aussi l'analyse et la présentation des informations de surveillance.

Mode global Les fonctions de pilotage du système, notamment l'envoi de commandes est impossible.

En prise de données, la tâche lv2mon reste cependant utile pour consulter les différentes traces d'exécution stockées sur fichier ainsi que les compte-rendus de run archivés sur disque. La surveillance en ligne de la prise de données présente les statistiques des algorithmes de sélection et différents histogrammes relatifs aux erreurs de lecture par port TMB.

Les logiciels de contrôle, de surveillance et de test du système offrent l'ensemble des fonctionalités nécessaires à l'intégration du système au sein de l'expérience et sont aujourd'hui opérationnels "in situ".

Conclusion

Ce travail décrit le nouveau système de déclenchement de niveau-2, développé autour des composants *T9000* et *C104* de la toute récente technologie *Transputer* proposée par INMOS/SGS Thomson.
Il met l'accent sur l'implémentation matérielle et logicielle du système ainsi que sur les interfaces avec le déclenchement de niveau-1, le système d'assemblage central et le système d'acquisition des données.
Un système de test intégré permet d'injecter des événements stockés sur mémoire de masse en reproduisant les conditions expérimentales de prise de données et ainsi de valider le bon fonctionnement du système en mode *local*.

Le système est entièrement piloté par des langages de haut niveau apportant la souplesse de fonctionnement nécessaire au programme *LEP* phase–2:

- les algorithmes sont aisément modifiables par du code FORTRAN standard,
- l'assemblage des données est codé en langage C. La modification et l'extension sont gérés par un tableau.

Les principales performances mesurées dans la configuration L3 sont les suivantes:

- Les bandes passantes de transfert mesurées sur les liens sont compatibles avec les valeurs attendues. Nous n'avons observé aucun disfonctionnement du circuit de routage *C104*.
- L'assemblage de 76 blocs de données (4400 octets au total) est effectué en 1.8 millisecondes répartis comme suit:
  - Temps de transfert du nombre de mots par bloc: $0.3ms$
  - Temps logiciel d'initialisation: $1ms$, paramétrisable par $T(\mu s) = 16.7 + (12.92 * nbblocs)$
  - Temps de transfert sur les 2 liens connectés: $0.5ms$
- Le comportement du système en fonction de la fréquence de déclenchement montre des performances stables (constantes par déclenchement) jusqu'à la fréquence de saturation. Le temps mort introduit dans l'acquisition est de $356\mu s$ par déclenchement. Etant inférieur au temps de numérisation des détecteurs, il n'introduit aucun temps mort additionnel.
- Les valeurs mesurées à la saturation sont:
  - $420Hz$ pour l'assemblage avec une seule unité dans la ferme.
  - $450/530Hz$ pour l'assemblage avec deux unités dans la ferme.
  - $75Hz$ pour l'assemblage suivi d'un traitement de $10ms$ par une unité.
  - $150Hz$ pour l'assemblage suivi d'un traitement de $10ms$ par deux unités.

- $275Hz$ pour la transmission complète des données vers la mémoire de sortie.

- Les algorithmes de traitement des données sont codés en Fortran, puis transcrits en C par l'utilitaire F2C. Le temps de traitement des algorithmes (non parallélisés) varie entre 2 et $10ms$, selon l'algorithme et les données. Le temps moyen en prise de données est de $7ms$ pour un *Transputer T9000* à $20MHz$. Pour un *Transputer T9000* à $50MHz$, les performances serait voisines de celles obtenues par *XOP*.

Le calendrier de livraison des composants (matériel et logiciel) et l'instabilité des logiciels (prototypes) ont considérablement perturbé le développement et la mise au point du système (pas d'outil de déverminage, serveur prototype, interface B103 prototype ...). Une utilisation plus large de ces composants passe par une amélioration sérieuse de l'environnement logiciel proposé par le constructeur.

Depuis son intégration dans l'expérience en juillet 95, le système composé de trente *Transputers T9000* et de deux routeurs dynamiques *C104*, assure la prise de données, l'assemblage des événements et le rejet en ligne du bruit de fond.
Bien que développé essentiellement autour de composants matériels et logiciels prototypes, dont certains souffrent de restrictions sévères, la fréquence moyenne de pannes (inhérentes au système) reste inférieure à une par 24 heures, ce qui est acceptable par L3. Les prototypes seront remplacés par des composants validés pendant l'arrêt hivernal du LEP (décembre 95 à avril 96).

Ce système est le premier déclenchement de niveau-2 développé dans cette technologie et installé au CERN.
Il satisfait pleinement le cahier des charges de L3 et confirme l'interet de la technologie *Transputer* dans cette application. La fiabilité du système contraste avec les difficultés de mise en œuvre.
Cette technologie apporte la souplesse attendue pour l'extensibilité du réseau. L'utilisation optimale des propriétés des circuits de routage pour l'assemblage d'événements nécessite une parfaite compréhesion du réseau. L'initialisation, le contrôle et la surveillance de tels réseaux demeurent le problème majeur pour des réseaux de grandes dimensions.

Annexe A

Présentation du *Transputer T9000* et du circuit de routage *C104*

A.1 Introduction

Le modèle de Processus Séquentiels Communicants définit par C.A.R Hoare dans [16] est un modèle de programmation parallèle à base de processus séquentiels asynchrones ayant la possibilité de communiquer entre eux.

On ne considère plus un programme comme une suite de commandes que l'on exécute séquentiellement de la première jusqu'à la derniére, mais comme un ensemble de tâches qui s'exécutent en parallèle et qui sont capables de communiquer entre–elles, chacunes des tâches exécutant un code séquentiel.

Les caractéristiques d'un language de programmation basé sur ce modèle sont (cf [38]):

- les événements non–déterministes sont gérés avec des commandes *gardées* (dont l'exécution est conditionnée par la vérification d'une condition logique).
- il existe une commande capable de démarrer plusieurs tâches en parallèle. Cette commande se termine lorsque toutes les tâches parallèles ont terminées leur exécution.
- la synchronisation entre deux tâches s'effectue par *rendez–vous*: la communication a lieu uniquement lorsque la tâche émettrice et la tâche réceptrice sont présentes au *rendez–vous*, la première présente au *rendez–vous* attendant que la seconde s'y présente.
- les *gardes* peuvent comporter plusieurs commandes d'entrées (accès à un moyen de communication, horloge ...). Dans ce cas, la commande *gardée* n'est exécutée que si l'une des sources nommées dans la commande d'entrée est prête à émettre. Si plusieurs gardes d'entrée sont prêtes à communiquer, l'une d'elles est choisie arbitrairement, les autres non pas d'effet.

Dans ce modèle, chaque tâche possède son propre espace de travail et s'éxécute de manière *asynchrone* par rapport aux autres tâches, sauf lors d'une communication où il y a *synchronisation par rendez–vous*.

Il constitue les bases de l'architecture matériel et logiciel du *Transputer*.

A.2 Le Transputer *T9000*

INMOS intègre sur le même circuit le *Transputer T9000*:

- une unité arithmétique et logique pour des entiers 32 bits,
- une unité de calcul en virgule flottante pour des flottants de 64 bits,
- 16 Koctets de mémoire interne ou de mémoire cache,
- un gestionnaire de tâches matériel pour la gestion du parallélisme concurrent,
- 4 canaux *événement* pour les synchronisations matérielles,
- 4 liens de communications séries offrant une bande passante totale de 80 Moctets/s,
- un processeur de canaux virtuels permettant de répartir les canaux virtuels sur les liens de communications,
- deux liens de contrôle permettant l'initialisation et la surveillance d'un réseau par chaînage,
- une interface mémoire programmable (*PMI*) pour la gestion de l'espace mémoire adressable de 4 Goctets.
- ainsi que plusieurs unités système et des horloges.

L'ensemble des échanges entre ces différents éléments s'effectue par l'intermédiaire d'une matrice CROSSBAR constituée de 4 bus de données et d'adresses de 32 bits. La figure A.1 présente l'architecture du *Transputer T9000*.

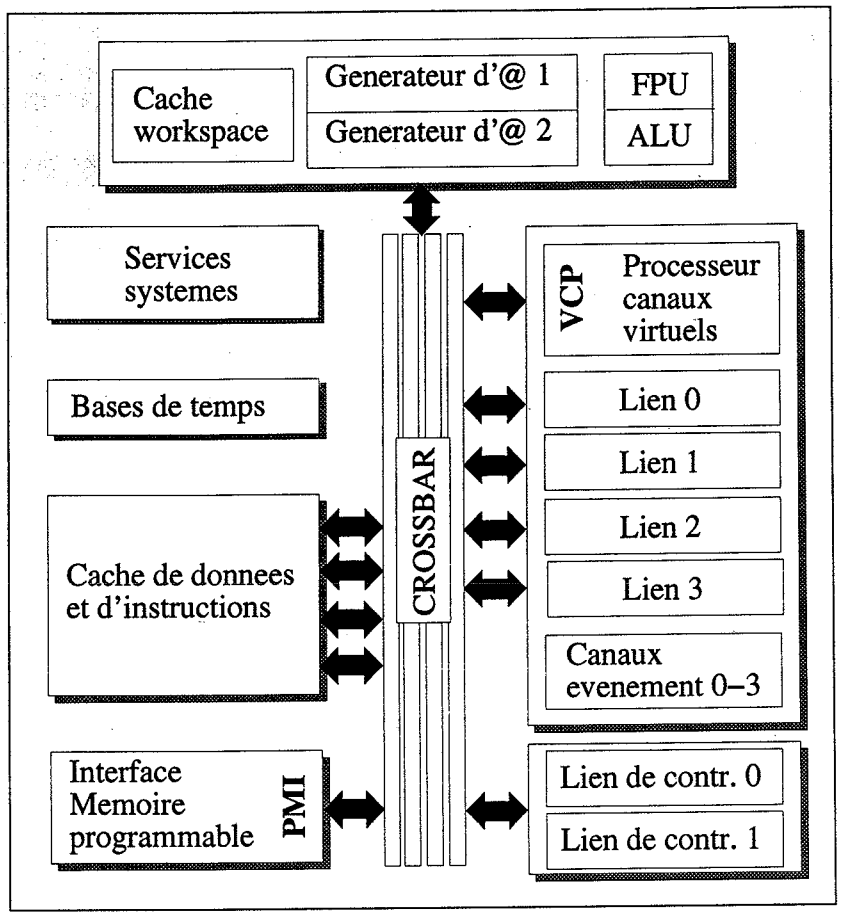


FIG. A.1 - *Architecture du* Transputer T9000

A.2.1 Le système de mémoire hiérarchique

Le *Transputer T9000* possède un système de mémoire hierarchique permettant des accès rapides et efficaces aux données et aux instructions. Il est constitué de deux caches indépendants, la mémoire cache principale par laquelle transitent les données et les instructions ainsi que d'une mémoire cache plus petite permettant un accès rapide aux variables locales.

La figure A.2 donne le schéma du système de mémoire hierarchique.

Le cache principal

Les 16 Koctets du cache principal sont découpés en 4 banques mémoire indépendantes de 256 lignes de 4 mots consécutifs de 32 bits. Un accès peut être effectué sur chaque banque à chaque cycle offrant ainsi une bande passsante de 200Mmots/s aux différentes unités pour un *Transputer* travaillant à une fréquence de 50 MHz. Un jeu de 4 bus d'adresses et de données permet l'accès simultané en écriture et lecture sur chaque banque mémoire. Un système d'arbitration gère l'accès des différentes unités aux banques mémoire.

Chaque banque mémoire gère un quart de l'espace mémoire. La répartition dans les quatres banques est effectuée par les bits d'adresse 4 et 5. Lorsqu'un accès mémoire est effectué, on vérifie si l'adresse est présente dans la banque correspondante. Si elle est présente et la ligne

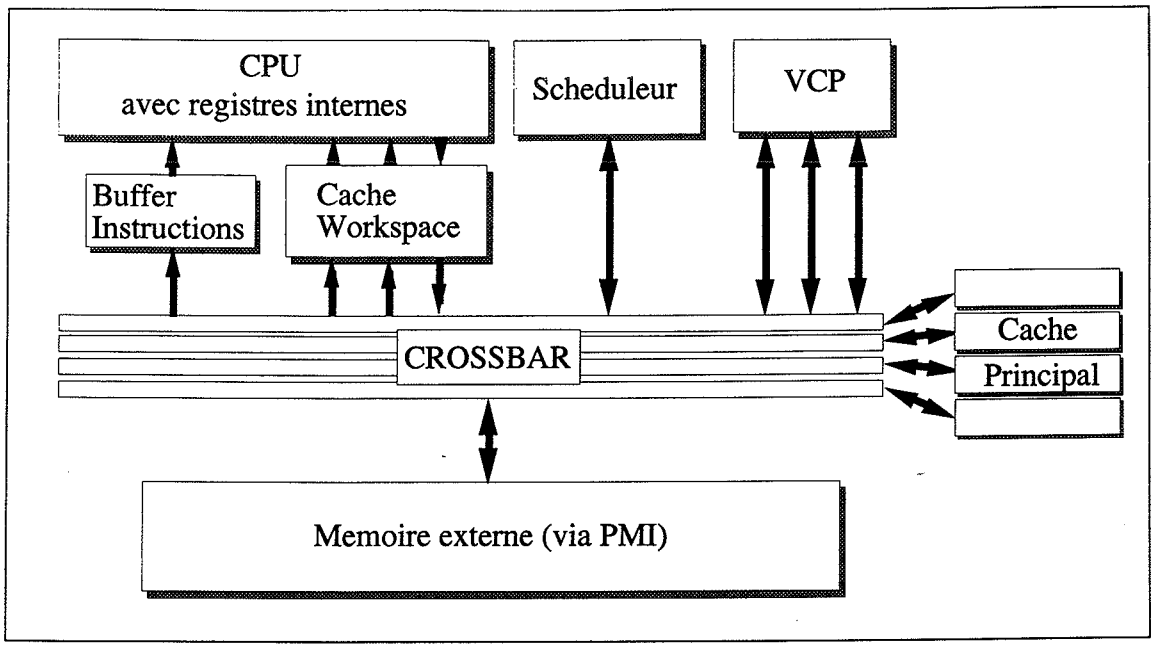


FIG. A.2 - *Système de mémoire hiérarchique du* Transputer T9000

valide, elle est accédée en un cycle. Si elle n'est pas présente dans la banque, l'adresse est passée au système de remplissage de cache (*refill engine*). Si cette adresse est *cachable*, le système de remplissage de cache calcule les adresses nécessaires au remplissage de la ligne. L'interface mémoire récupère alors la ligne dans la mémoire externe. Si l'adresse est marquée *non-cachable*, l'interface mémoire récupère uniquement cette adresse.

Le système remplissage de cache assure qu'il y a toujours une ligne de disponible dans chaque banque. La sélection de la ligne est aléatoire. Si un mot de la ligne a été modifié depuis la dernière lecture, il est écrit dans la mémoire externe. Ce mode de fonctionnement est appelé *early write-back*.

A l'initialisation le cache principal se comporte comme une mémoire statique interne de 16 Koctets. Lors de l'intialisation du *Transputer*, cette mémoire est configurée pour être utilisée comme une mémoire cache de 16 Koctets, 8 Koctets de cache et 8 Koctets de mémoire interne ou bien 16 Koctets de mémoire interne.

Le fonctionnement précis de la mémoire cache principale est décrit dans le manuel [18, Chapitre 9] et [19, Chapitre 15].

Le cache espace de travail

Sur le *Transputer T9000*, les variables locales sont stockées dans un cache inclu dans le pipeline appelé *cache espace de travail*(*Workspace cache*).

Ce cache de 32 mots possède trois ports d'accès, deux en lecture et un en écriture. Il peut contenir les 32 premiers mots de la pile de processus et de l'espace de travail. Il permet l'accès aux variables locales sans sortir de l'unité centrale. Le compilateur *C INMOS* définit le type *register* qui assure que la variable utilisera le *cache espace de travail*.

Comme les variables locales peuvent être accédées rapidement, elles peuvent être lues dans le premier étage du pipeline et utilisées ensuite pour le calcul d'adresses non-locales dans le prochain étage. Le cache espace de travail est *write-through*, c'est à dire que lorsqu'une variable est mise à jour dans le *cache espace de travail*, elle est également mise à jour dans le cache principal.

Les trois ports d'accès peuvent être utilisés à chaque cycle. Pour un *Transputer* à 50 MHz,

la bande passante du cache espace de travail est donc de 150Mmots/s. Pour de plus amples informations sur le fonctionnement du cache espace de travail, se reporter à [18] et [19].

A.2.2 Le pipeline et son groupeur d'instructions

Pour augmenter sa puissance de calcul, le *Transputer T9000* intégre une architecture pipeline superscalaire qui lui permet d'exécuter plusieurs instructions par cycle. Pour utiliser de façon optimale la structure pipeline, il faut que les instructions soient présentes à l'entrée du pipeline selon une séquence précise. Le *Transputer T9000* possède un groupeur d'instructions qui analyse le flot d'instructions et répartit les instructions par groupes pour permettre le meilleur taux d'occupation du pipeline.

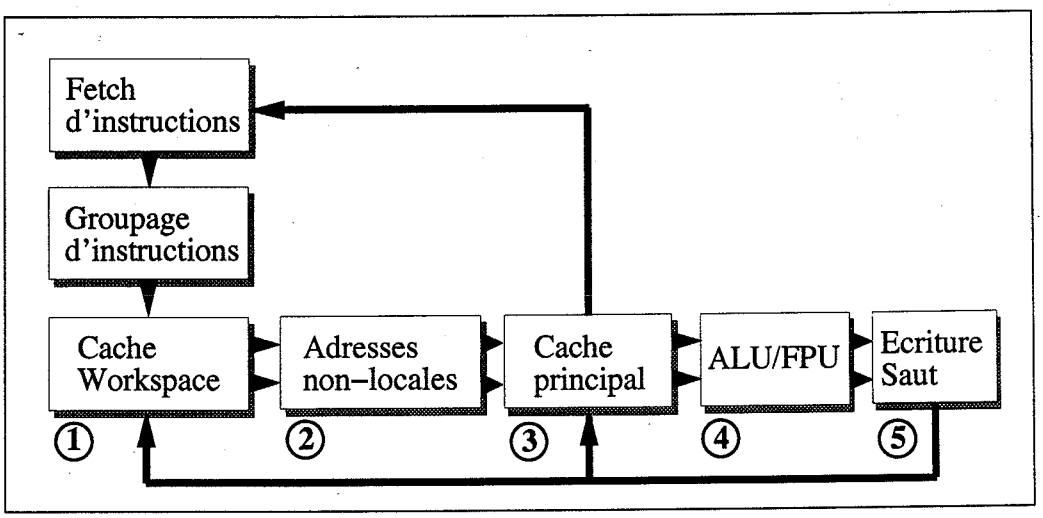


FIG. A.3 - *Pipeline du* Transputer T9000

Le pipeline est constitué de 7 étages (2 + 5) (voir figure A.3):

- Les étages de recherche d'instructions (*fetch*) et de décodage/groupage: lecture du flot d'instructions puis construction des groupes d'instructions exécutées dans les 5 étages suivants.
- Le *cache espace de travail*: récupération de deux variables locales. Grâces aux trois ports d'accès, deux lectures et une écriture peuvent être exécutées pendant le même cycle dans le cache espace de travail.
- La génération des adresses non–locales: deux calculs d'adresse peuvent être exécutés pendant le même cycle.
- Le cache principal: récupération de deux variables non–locales. Si les données sont contenues dans des banques différentes, les deux lectures sont effectuées pendant le même cycle.
- Les unités de calcul (*ALU* et *FPU*): exécution d'une opération sur l'une ou l'autre unité de calcul par cycle.
- L'écriture et la gestion des sauts. Dans le cas de l'écriture si l'adresse d'écriture est présente dans le cache espace de travail, celui–ci est mis à jour ainsi que la mémoire principale.

L'optimisation du pipeline est transparente pour l'utilisateur grâce au groupeur d'instructions.

A.2.3 L'interface mémoire programmable

L'interface mémoire programmable permet de répartir l'espace mémoire adressable de 4 Goctets en quatre banques mémoire indépendantes. Ce partage permet la construction de systèmes mixtes avec des mémoires statiques et dynamiques. Le *Transputer T9000* possède un bus de données d'une largeur de 64 bits. Chaque banque mémoire est définie par:

- son adresse de base et sa taille,
- la largeur de son bus de données: 8, 16, 32 ou 64 bits pour de la mémoire statique, 32 ou 64 bits pour de la mémoire dynamique,
- ses modes d'accés: *cachable, non-cachable* et/ou *device only*(le mode *device* est décrit ci-dessous). Lorque l'on travaille avec un bus de données de 64 bits, l'ensemble des accès doivent être effectués à travers la mémoire cache.
- 4 signaux de contrôle,
- les registres de programmation des cycles de lecture et/ou écriture des signaux de contrôle.

Pour des informations plus précises sur l'interface mémoire programmable se référer à [18, Chapitre 10].

A.2.4 Les accès en mode *device*

Le fonctionnement du pipeline et de la mémoire cache assure que la lecture à une adresse donnée retourne la dernière valeur écrite à cette adresse. Mais il ne permet pas de connaître le moment où l'écriture a effectivement lieu dans la mémoire principale, et la relation d'ordre temporel entre les opérations de lecture et d'écriture à des adresses distinctes.

Ce type de fonctionnement pose quelques problèmes pour des périphériques externes, tels que des registres de commande ou d'état, des mémoire *First In First Out*. Le fonctionnement de ces périphériques est directement défini par l'ordre temporel entre les opérations d'écriture et de lecture à leurs adresses, et par le fait que les données écrites à ces adresses soient effectivement copiées dans ces périphériques.

Pour ces raisons, le *Transputer T9000* possède un jeu d'instructions, appelé *instructions device* permettant l'accès à ces périphériques. Ce jeu d'instructions est utilisé par le compilateur *C INMOS* en définissant la variable avec le type *volatile* [21]. Ces instructions assurent que:

1. pour les zones mémoire définies par l'interface mémoire programmable comme *device only*, les opérations d'écriture et de lecture ont effectivement lieu dans la mémoire principale.
2. les opérations de lecture et d'écriture sont exécutées selon leur ordre dans la séquence de code.
3. l'instruction de transfert de bloc *devmove* effectue chaque lecture successive à une adresse supérieure à la précédente, de même pour l'écriture.

Pour plus d'informations sur les accès en mode *device* se reporter à [19].

A.2.5 L'unité centrale de traitement

Opérations séquentielles

Le *Transputer* dispose d'un nombre réduit de registres, compensé par le *cache espace de travail*, pour l'exéxution d'une tâche. Lors de l'exécution d'une tâche, cinq registres sont utilisés:

- un registre **Wptr** contenant le pointeur sur l'espace de travail courant permettant l'accès aux variables locales.

- un registre **IptrReg** contenant le pointeur sur la prochaine instruction à exécuter.
- et trois registres **Areg, Breg** et **Creg** formant une pile d'évaluation.

L'unité de calcul flottant contient aussi trois registres **FPAreg, FPBreg** et **FpCreg** également organisés en pile d'évaluation.

Priorité

Le *Transputer* supporte deux niveaux de priorité: basse priorité et haute priorité.

Une tâche haute priorité est exécutée dès qu'elle est disponible, interrompant le cas échéant une tâche basse priorité. Une tâche haute priorité ne s'arrête que si elle accède un point de débranchement ou si elle se termine. Lors de l'interruption d'une tâche basse priorité par une tâche haute priorité, l'état de la tâche basse priorité est sauvegardé dans des registres "fenêtre" avant que la tâche haute priorité ne s'exécute. Lorsqu'il n'y a plus de tâches haute priorité, l'état de la tâche interrompue est rechargée, puis elle reprend son exécution à son point d'interruption.

Pour les tâches basse priorité, le temps CPU est partagé en tranches temporelles de $256\mu s$. Une tâche basse priorité sera active au maximun pendant deux tranches temporelles si elle ne possède que des points de débranchement par temps partagé. Autrement elle peut suspendre son exécution sur des points de débranchement, ou bien lorsqu'elle a terminé. Si elle ne possède aucun point de débranchement, son exécution ne peut être interrompue que par une tâche haute priorité et suspendue lorsqu'elle s'achève.

Gestion des tâches concurrentes

Le *Transputer* possède un gestionnaire de tâches microcodé qui permet la mise en place de parallélisme concurrent. Le changement de contexte s'effectue en moins d'une microseconde.

A tout instant, une tâche peut se trouver dans l'un des états suivants:

Active :

- en cours d'exécution.
- interrompue par une tâche haute priorité.
- dans la liste d'attente pour être exécutée.

Inactive :

- prêt à émettre sur un canal.
- prêt à recevoir sur un canal.
- en attente jusqu'à une date donnée.
- en attente sur un sémaphore (cf Chapitre C.1).

Le gestionnaire de tâches est tel que les tâches inactives ne consomment pas de temps CPU. Il gère deux listes d'attente relatives aux deux priorités. Une liste d'attente est une liste chaînée de tâches actives gérée à partir des registres **FptrReg**, pointeur de tête, et **BptrReg**, pointeur de queue. La figure A.4 illustre le mécanisme de gestion de la liste de basse priorité. La tâche S constitue la tâche courante, en cours d'exécution, tandis que les tâches P, Q et R sont dans la liste d'attente.

Les points de débranchement rendent les tâches inactives quelque soit leur priorité. Pour les tâches basse priorité, il existe en plus des points de débranchement par temps partagé qui suspende l'exécution d'une tâche courante, et l'insère en fin de la liste d'attente.

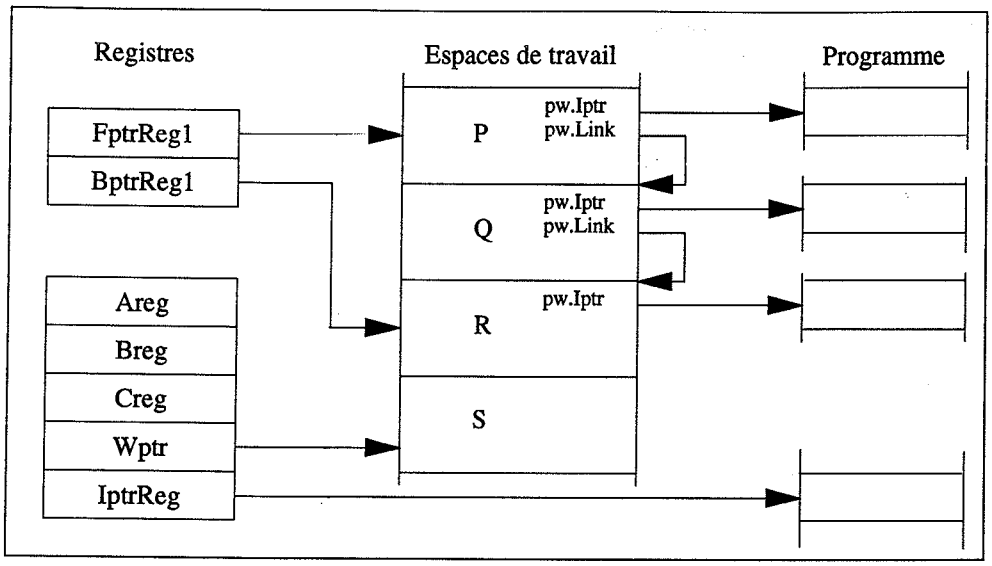


FIG. A.4 - *Liste chaînée de tâches basse priorité*

A.2.6 Les bases de temps (*timers*)

Le *Transputer* possède deux registres de 32 bits incrémentés périodiquement appelés *timer*, un pour chaque priorité. Ces registres sont utilisés comme base de temps. Accessibles en lecture, ils permettent de suspendre une tâche jusqu'à une date donnée.

Le premier *timer* est accessible par les tâches haute priorité, il est incrémenté toutes les microsecondes. Un cycle complet correspond à environ 4295 secondes.

Le deuxième *timer* est accessible par les tâches basse priorité, il est incrémenté toutes les 64μs. Un cycle complet correspond à environ 76 heures.

Pour plus d'informations sur les *timers*, se référer à [19].

A.2.7 Les communications

Les tâches communiquent par des canaux. Un canal est un moyen de communication point–à–point unidirectionnel entre deux tâches pouvant se situer sur un même *Transputer*: *canal interne*, ou sur des *Transputers* distincts: *canal externe*. La communication par canal comprend la synchronisation et le transfert de l'information. En effet la synchronisation assure que le transfert de l'information est activé uniquement lorsque les tâches sont prêtes selon un modèle de communication dit par *rendez–vous*.

Canal de communication interne

Un canal interne est mis en œuvre par un mot de l'espace mémoire dans lequel est stocké la référence de la tâche présente la première au *rendez–vous*. Ce mot est initialisé à la valeur *NotProcess*. Considérons deux tâches S et R effectuant une communication interne. Supposons que la tâche S se présente la première au rendez–vous, elle consulte le canal interne pour savoir si la tâche R est présente. Si la tâche R n'y est pas, la tâche S dépose sa référence dans le canal et devient inactive. Lorsque la tâche R vient au rendez–vous, elle consulte le canal interne et voit que la tâche S est présente. L'échange de l'information a alors lieu, il est effectué par un transfert mémoire géré par l'unité centrale. Une fois le transfert effectué la tâche S est activée, le canal réinitialisé à *NotProcess* et la tâche R continue son exécution.

Canal de communication externe

Le *Transputer T9000* possède quatres liens de communications bidirectionnels. Chaque lien est *full–duplex*, succeptible de supporter des communications simultannées dans les deux directions [12]. Il est constitué de deux ports de communication: un en entrée et un en sortie. Un contrôleur *Direct Memory Access* est dédicacé à chaque port, les données sont donc échangées indépendament de l'unité centrale. L'ensemble des quatres liens de communication offre une bande passante bidirectionnelle de 80 Moctets/s.

Chaque lien de communication permet la mise en place d'un nombre arbitraire de communications point–à–point via des *liens virtuels*. Chaque lien virtuel donne accès à deux canaux de communication, un dans chaque direction. On associe sur les deux processeurs communicant à travers un lien virtuel un bloc de contrôle de lien virtuel. Le processeur de canaux virtuels *Virtual Channel Processor* multiplexe ces canaux virtuels sur les liens physiques.

Le protocole de communication externe est constitué de quatre niveaux comme le montre la figure A.5 (extraite de [9]).

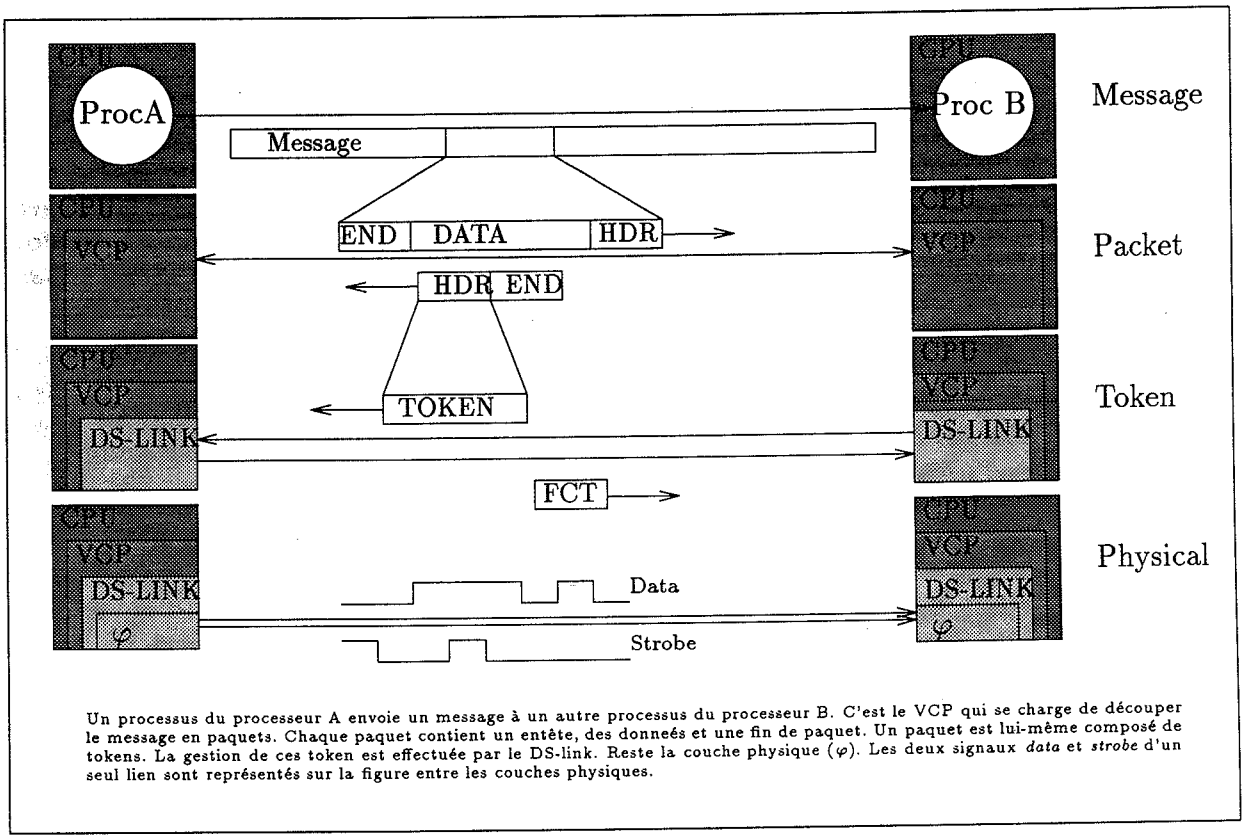


FIG. A.5 - *Les différentes couches du protocole de communication*

Le protocole niveau *message*: Les messages sont échangés entre les tâches et peuvent être de longueur variable. Lorsqu'une tâche effectue une communication externe, elle est désactivée. Le gestion du transfert est laissée aux processeurs de canaux virtuels. Lorsque le transfert est achevé, les tâches sont réactivées. Chaque message est découpé en paquets par le processeur de canaux virtuels *VCP*. Les paquets relatifs à un même message sont transmis à travers le même lien physique.

Le protocole niveau *paquet*: Il assure la synchronisation entre les tâches, il est géré de chaque côté par les processeurs de canaux virtuels.

Les paquets échangés entre les processeurs de canaux virtuels sont constitués:

- d'un en–tête (*header*) pour identifier le bloc de contrôle du lien virtuel *VLCB* utilisé par la tâche destinataire. L'en–tête peut être constitué de plusieurs octets.
- d'une terminaison indiquant la fin du paquet (*EOP*) ou la fin du message pour le dernier paquet (*EOM*).

On distingue deux types de paquets: les paquets de données et les paquets d'acquittement. Les paquets de données sont constitués d'un en–tête, d'un champ de données de 32 octets maximun et d'une terminaison. Les paquets d'acquittement sont constitués d'un en–tête et de la terminaison *EOP*. La figure A.6 présente la structure d'un paquet.

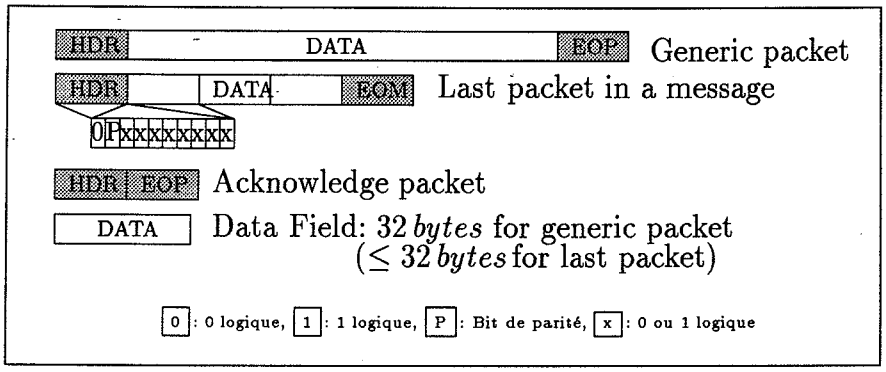


FIG. A.6 - *Structure d'un paquet*

Chaque paquet de données transmis à travers un lien virtuel doit être acquitté avant que le prochain ne soit émis pour s'assurer qu'il n'y a pas de données perdues. Un paquet d'acquittement est automatiquement émis par le *VCP* du processeur récepteur dès la réception de l'en–tête du paquet de données. Sur le processeur émetteur, la réception de l'acquittement autorise la construction puis l'émission du prochain paquet de données.

La synchronisation est effectuée de la façon suivante:

- Lors d'un transfert, si la tâche réceptrice n'est pas présente au *rendez–vous*, le premier paquet transmis par le *VCP* du processeur émetteur est stocké dans une zone tampon profonde de 32 octets. Le paquet d'acquittement n'est transmis que lorsque la tâche réceptrice se présente au *rendez–vous*.
- sur le processeur récepteur, la réception du dernier paquet réactive la tâche réceptrice.
- le processeur émetteur attend la réception de l'acquittement du dernier paquet de données pour réactiver la tâche émettrice.

Le multiplexage des canaux virtuels est obtenu en constituant pour chaque lien physique une liste chaînée d'attente des paquets prêts à émettre. Dès que l'émission d'un paquet est achevée sur un lien, le *VCP* prend le prochain disponible dans la liste d'attente. La réception d'un paquet d'acquittement pour un canal insère le prochain paquet de données à la fin de la liste d'attente du lien physique considéré jusqu'à transmission complète du message.

On distingue deux listes d'attente: une pour les paquets correspondant à des tâches haute priorité, l'autre pour les paquets correspondant à des tâches basse priorité. Les paquets de la liste d'attente basse priorité ne sont transmis que lorsque la liste d'attente haute priorité est vide. Pour chaque liste d'attente, les paquets de données et d'acquittement sont transmis selon leur position dans cette liste jusqu'à ce que celle–ci soit vide.

La figure A.7 donne un exemple de répartition des différents paquets sur un lien physique lors de transfert unidirectionnel et bidirectionnel.

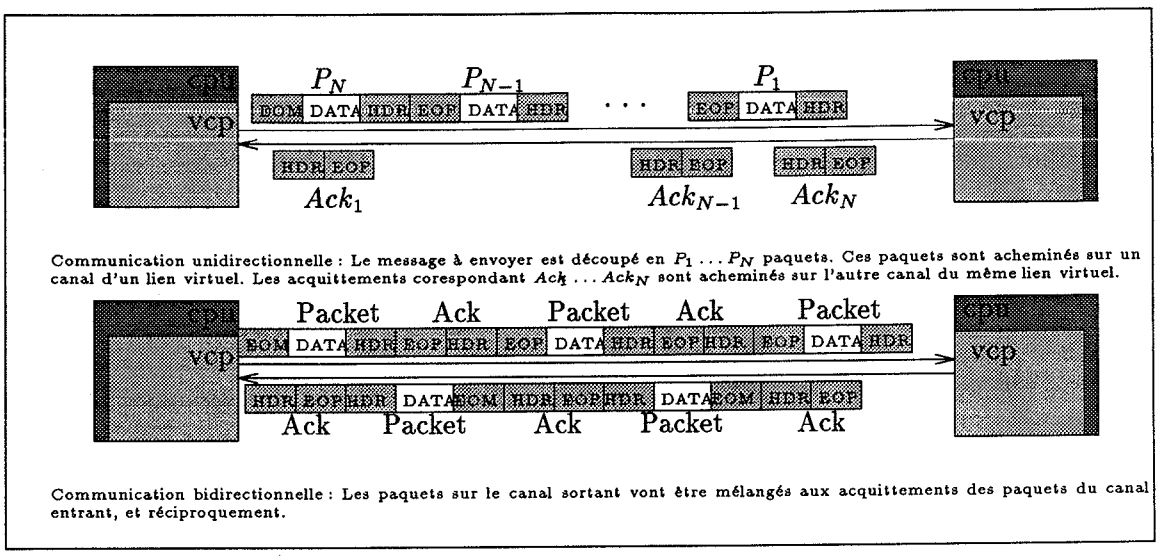


FIG. A.7 - *Exemple de communications uni et bidirectionnelles*

Le protocole niveau *token*: il permet le contrôle du flux de données. Il est géré par les ports de communication.

Les paquets sont constitués de *tokens* echangés entre les ports de communication *DS-Link*. On distingue deux types de *tokens*:

- les *tokens* de données comprenant l'en–tête et les données.
- les *tokens* de contrôle. Ils permettent le contrôle du flux d'informations et participent à la synchronisation des tâches.

Le tableau A.1 énumère les différents *tokens* et précise leur structure.

Type de Token	Abbréviation	Structure
Data token	–	P0DDDDDDDD
Flow control token	FCT	P100
End of paket	EOP	P101
End of message	EOM	P110
Escape token	ESC	P111
Null token	NULL	ESC FCT

P = bit de parité
D = bit de donnée: 0 ou 1

TAB. A.1 - *Structure des différents tokens*

Le contrôle du flux de données est effectué entre chaque port de communication. Ce contrôle empêche le port émetteur de surécrire le tampon d'entrée sur le port récepteur. Chaque port récepteur posséde un tampon profond d'au moins 8 *tokens*, en réalité de 20 *tokens* [33]. A chaque fois que le port d'entrée est succeptible de recevoir les prochains 8 *tokens*, un *token* de contrôle de flux *FCT* est transmis sur le port de sortie associé. Ce *FCT* autorise le port émetteur à transmettre les 8 prochains *tokens*. Une fois transmis ces 8 *tokens*, le port émetteur attend la réception du prochain *FCT* pour transmettre les futurs *tokens*. Le fait que le tampon soit profond de plus de 8 *tokens* assure que le prochain *FCT* est reçu avant que les prochains 8 tokens soient totalement transmis. Ainsi ce contrôle de flux ne doit pas diminuer la bande

passante uni-directionnelle d'un lien. Cependant lorsque les ports de communications *DS-Link* sont distants de plus de $10m$, le contrôle de flux détériore la bande passante [15].

En l'absence de *tokens de données*, des *Null tokens* sont échangés pour permettre la détection d'erreurs de parité et de déconnexion.

Le protocole niveau *Bit*: Il assure une réception asynchrone des bits par encodage d'une horloge lors de la transmission.

Chaque port de communication *DS-Link* est constitué de deux lignes, une pour les données, l'autre pour le signal *Strobe*. La ligne de données transmet les bits de données, la ligne *Strobe* change à chaque fois que le bit de donnée à le même état que le bit précédent. La figure A.8 présente un chronogramme des signaux relatifs à ces deux lignes. Par ce moyen, ces deux lignes

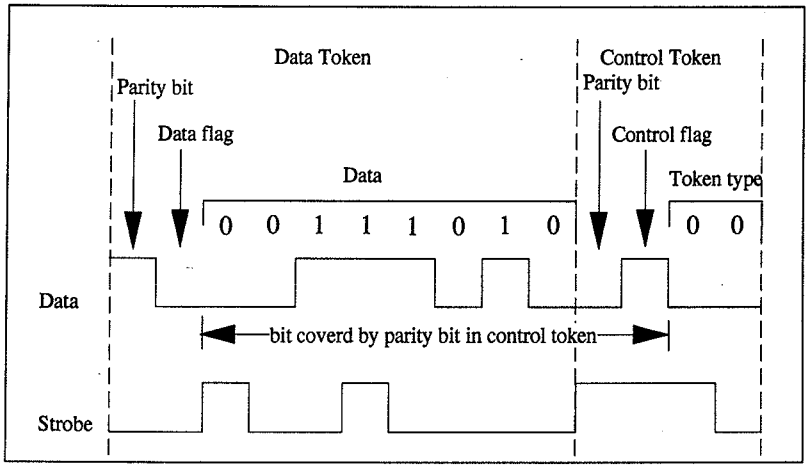


FIG. A.8 - *Format des signaux* Data *et* Strobe

encode une horloge, utilisée par le port récépteur pour le décodage des données. Ceci simplifie la distribution d'horloge à l'interieur d'un réseau. Un bit de parité permet de détecter des erreurs de transmission.

A.2.8 Les canaux *événement (Event)*

Le *Transputer* possède quatres canaux *événement* bidirectionnels. Ces canaux *événement* interfacent des synchronisations externes avec des tâches. La synchronisation s'effectuant sans transfert de données, les canaux *événement* permettent une synchronisation temporelle entre un signal externe et des tâches. Cette synchronisation est mise en œuvre par les broches *EventIn* et *EventOut*.

On peut utiliser ces canaux deux façons différentes:

- pour recevoir une synchronisation externe: canal *événement* en entrée.
- pour émettre une synchronisation externe: canal *événement* en sortie.

Canal *événement* en entrée

La synchronisation est effective lorsque la broche *EventIn* a un niveau logique 1, le canal événement est alors prêt. Si la tâche est prête à recevoir la synchronisation, elle est activée et le processeur passe la broche *EventOut* à l'état haut. Si elle n'est pas prête, le processeur attend qu'elle le soit pour acquitter la synchronisation.

Canal *événement* en sortie

Lorsqu'une tâche doit émettre une synchronisation matérielle, elle accède le canal *événement*, elle devient alors inactive. Le processeur positionne alors au niveau logique 1 la broche *EventOut*. Tant que l'acquittement n'est pas effectué, la tâche reste inactive. L'acquittement de la synchronisation est effectif lorsque la broche *EventIn* passe a un niveau logique 1, la tâche redevient alors une tâche active.

Les interruptions

Pour les applications temps–réel, il est important que le processeur réponde le plus rapidement possible à une synchronisation extérieure. En associant un canal *événement* en entrée à une tâche haute priorité, on réalise l'équivalent d'une interruption. En effet la tâche reste inactive tant qu'il n'y a pas de synchronisation extérieure. Lorsque la synchronisation est effective, la tâche est activée et devient la tâche courante pourvu qu'elle soit l'unique tâche haute priorité.

Pour de plus amples informations sur les canaux événement, se référer à [20, page 21] et [19, page 155].

A.3 Le circuit de routage *C104*

Ce circuit de routage permet l'interconnexion entre des *Transputers T9000* non voisins, et la construction de diverses topologies de réseaux.

Il est constitué de 32 liens de communication *DsLink* et d'une matrice d'interconnexion 32*32 non–bloquante capable de router un paquet de n'importe quel lien vers n'importe quel autre. Les liens travaillent en parallèle et le transfert d'un paquet entre une paire de liens n'affecte pas le temps de latence ou la bande passante d'un autre paquet transmis sur une seconde paire de liens. Le *C104* possède une unité de commande qui lui permet de sélectionner le lien de sortie en fonction de l'en–tête. Tout ce qui suit l'en–tête est considéré comme le corps du paquet jusqu'au *token* de fin de paquet, ceci permet au *C104* de transmettre des messages de taille variable.

Il possède en plus deux liens de contrôle qui permettent son initialisation et sa programmation. La figure A.9 présente l'architecture interne du circuit de routage *C104*.

A.3.1 Mode et Algorithme de routage

Mode de routage du *C104*

Le *C104* utilise le mode de routage *wormhole*. La figure A.10 présente le cheminement d'un message à travers un réseau de *C104*. L'avantage de ce modèle est notoire puisqu'un message peut commencer à être reçu avant que l'émission ne soit terminée. De la même façon, si le message est suffisament court pour pouvoir être stocké sur le circuit virtuel, la source est libérée avant la réception du premier *token*.

Algorithme de routage du *C104*

Le mode de routage *wormhole* nécessite un algorithme de routage afin de déterminer sur quel lien doit être envoyé un paquet qui vient d'arriver. L'algorithme de routage du *C104* est un algorithme de routage par intervalle: un routeur trouve la direction pour communiquer un message en déterminant l'intervalle contenant l'adresse de destination du message, chaque intervalle étant associé à une direction particulière [13]. La figure A.11 donne un exemple de routage par intervalle pour un réseau comprenant 2 *C104* et 6 *T9000* possédant chacun un lien virtuel étiqueté de 0 à 5. Chaque intervalle contient les numéros des liens virtuels accessibles par le lien physique. La notation $[x, y)$ signifie que les messages contenant des en–têtes supérieurs ou égals à x et strictement inférieurs à y utiliseront le lien associé à cet intervalle.

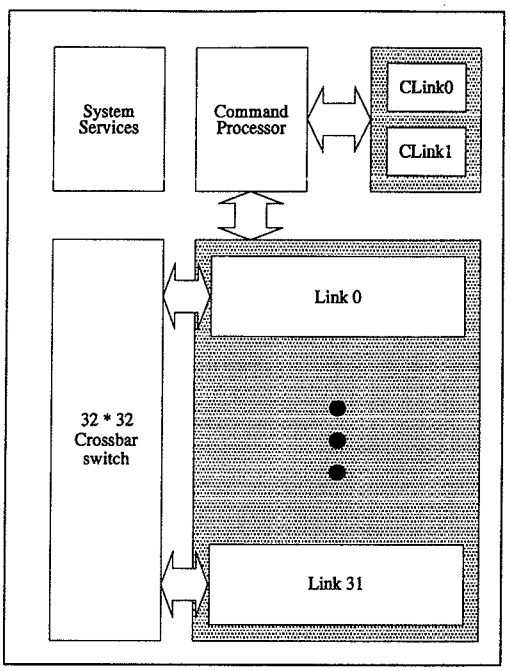


FIG. A.9 - *Architecture interne du circuit de routage* C104

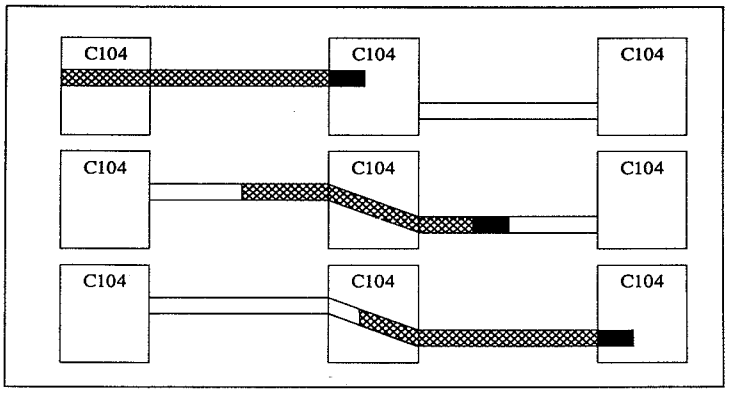


FIG. A.10 - *Mode de routage* wormhole

L'interblocage dans les réseaux utilisant le mode de routage *Wormhole*

L'interblocage se caractérise par la présence dans le réseau de paquets bloqués et qui le restent. Dans le routage *wormhole*, si l'en–tête d'un message référence un lien déja utilisé, le message est bloqué dans les tampons jusqu'a ce que le lien se libére. Par conséquent ces paquets peuvent utiliser des ressources nécessaires à d'autres paquets et les bloquer eux aussi. Voilá pourquoi le routage *wormhole* associé à un contrôle de flux *Flow Control Token* est succeptible d'introduire de l'interblocage dans un réseau [11]. Néanmoins il est possible d'étiqueter la plupart des topologies de réseaux de telle sorte que le paquet utilise le meilleur chemin à travers le réseau sans introduire interblocage.

En l'abscence de congestion sur un lien de sortie, le temps de latence pour un paquet à travers un *C104* est inférieur à 1μs. Le circuit de routage offre un taux de traitement des paquets supérieur à 200Mpaquets/s. Les problèmes d'interblocage sont résolus pour des réseaux

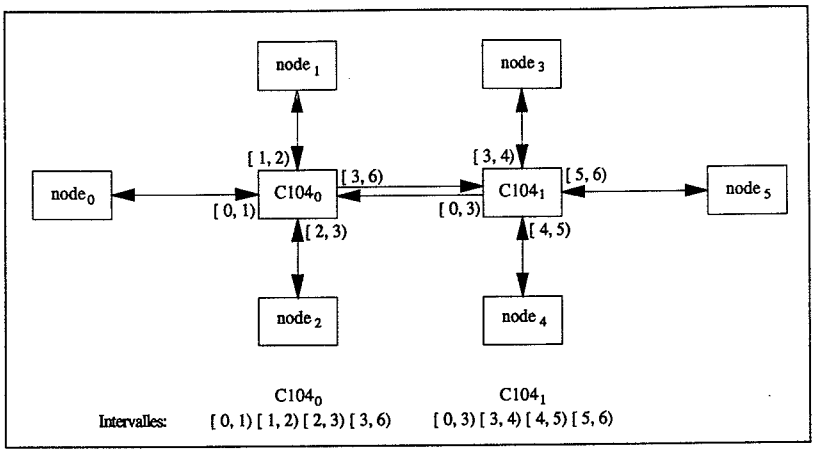


FIG. A.11 - *Exemple de routage par numérotation d'intervalle*

simples, arbre, pipeline, par le compilateur de réseau *indl* en utilisant l'option *autolabel*. Ce compilateur génère l'étiquetage des différents *C104* présents dans le réseau [22].

A.3.2 La suppression d'en–tête

Une approche pour simplifier la construction de réseaux est de réaliser des réseaux hiérarchiques. L'en–tête d'un paquet pouvant contenir plusieurs octets, chaque octet permet l'accès à un sous–réseau. La suppression d'en–tête permet de supprimer l'octet utilisé pour accéder à un sous–réseau à la sortie de ce sous–réseau. La figure A.12 donne un exemple de réseaux construits avec suppression d'en–tête. Les avantages apportés par l'utilisation de la suppression d'en–tête

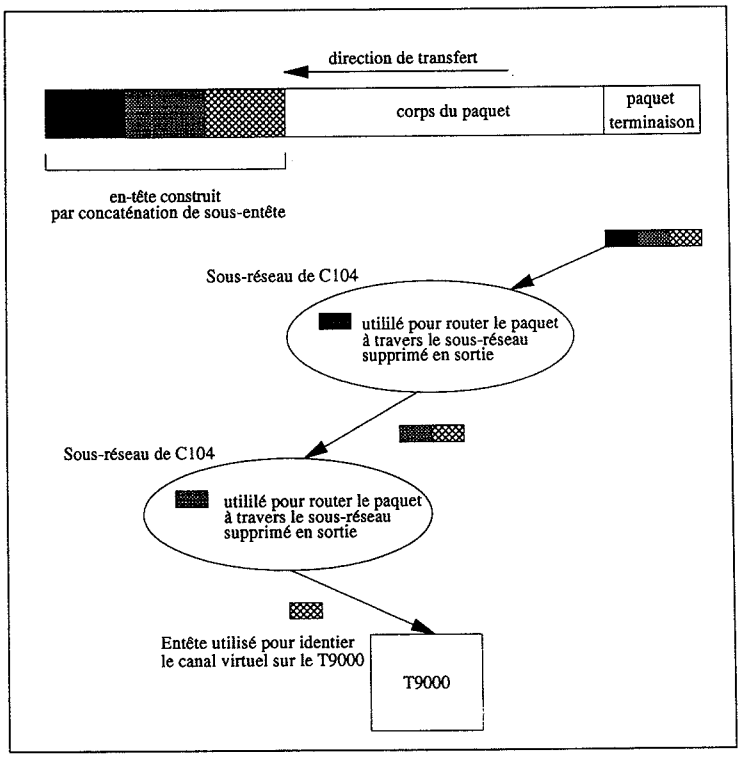


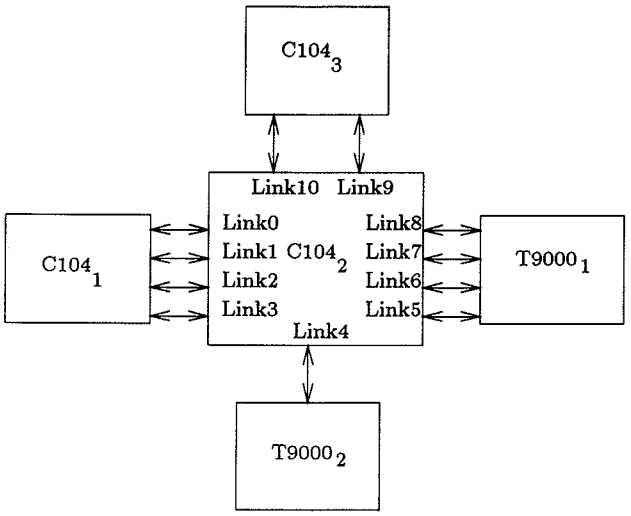
FIG. A.12 - *Réseau hiérarchique utilisant la suppression d'en–tête*

dans un réseau sont [33] les suivants:

- Les en–têtes pour les liens virtuels sont distincts des en–têtes pour le réseau de routage.
- L'étiquettage d'un réseau peut être fait indépendamment de l'application utilisant ce réseau.
- l'accès à un nombre infini de canaux virtuels est théoriquement possible.
- Enfin cette technique minimise le temps de latence.

A.3.3 Le routage adaptatif groupé

Le circuit de routage *C104* met en œuvre un routage adaptatif groupé. Plusieurs liens consécutifs d'un *C104* sont regroupés, de sorte que lorsqu'un paquet arrive à destination d'un de ces liens, il est dirigé vers le premier disponible. La figure A.13 donne un exemple de routage utilisant cette méthode. Ce mode de routage permet d'augmenter la bande passante et de dimin-



Soit un paquet routé du $C104_1$ via le $C104_2$ à destination du $T9000_1$. Supposons qu'arrivant sur le $C104_2$, l'en-tête du paquet spécifie que ce dernier doit être dirigé en sortie sur le lien 6. Si ce lien est utilisé, le paquet sera automatiquement routé vers les liens 5, 7 ou 8 selon l'ordre de leur disponibilité.

Les liens sont configurés par le registre spécial **Group**. Chaque bit correspond à un lien et peut être initialisé à "Start" pour commencer un groupe ou "Continue" pour être inclus dans un groupe.

FIG. A.13 - *Routage adaptatif groupé*

uer les probabilités de congestion. Cependant lorque le nombre de paquets destinés à un groupe de liens est supérieur au nombre d'éléments dans le groupe, le problème de congestion réapparait. Dans ce cas les entrées en excès sont *suspendues*(*Flow Control Token*), elles ont alors accès aux liens de sortie au fur et à mesure que ces liens se libèrent. Un système d' arbitrage assure une rotation entre les entrées *suspendues* demandant l'accès à un même groupe de sortie. Chaque entrée aura transmis un paquet avant que le deuxiéme ne soit transmis pour l'ensemble des entrées accédant à ce groupe.

Annexe B

Le Module *Tmb*

B.1 Le cahier des charges

les fonctionnalités d'un module *Tmb* sont:

1. stocker à chaque croisement les données au standard ECLine à 60 ns par mot puis les mémoriser en moins de 500 μs lorsque l'événement associé est validé par le système de déclenchement de niveau–1.

2. intégrer un injecteur de données pour reproduire *in situ* les conditions expérimentales de prise de données.

Chaque module est constitué de deux parties indépendantes appelées respectivement *Acquisition* et *Injection*. Les données expérimentales et les données injectées sont multiplexées à l'entrée de la partie *Acquisition*.

L'ensemble des données est réparti sur 48 ports. Pour des raisons d'encombrement physique sur une carte au format *Fastbus*, chaque module *Tmb* comporte 4 ports.

B.2 La séquence de réception des données

La séquence de réception [8] des données est effectuée à partir:

- du signal d'échantillonage des données *Str*[1] propre au standard ECLine,
- des signaux de contrôle propres à l'expérience
  - le signal matérialisant l'instant de croisement des faisceaux *Bcr*[2],
  - le signal *Lv1*[3] indiquant que l'événement est validé par le système de déclenchement de niveau–1,
  - le signal *Overflow* chargé d'inhiber la réception de nouveaux croissements jusqu'à ce que le système soit à nouveau prêt.

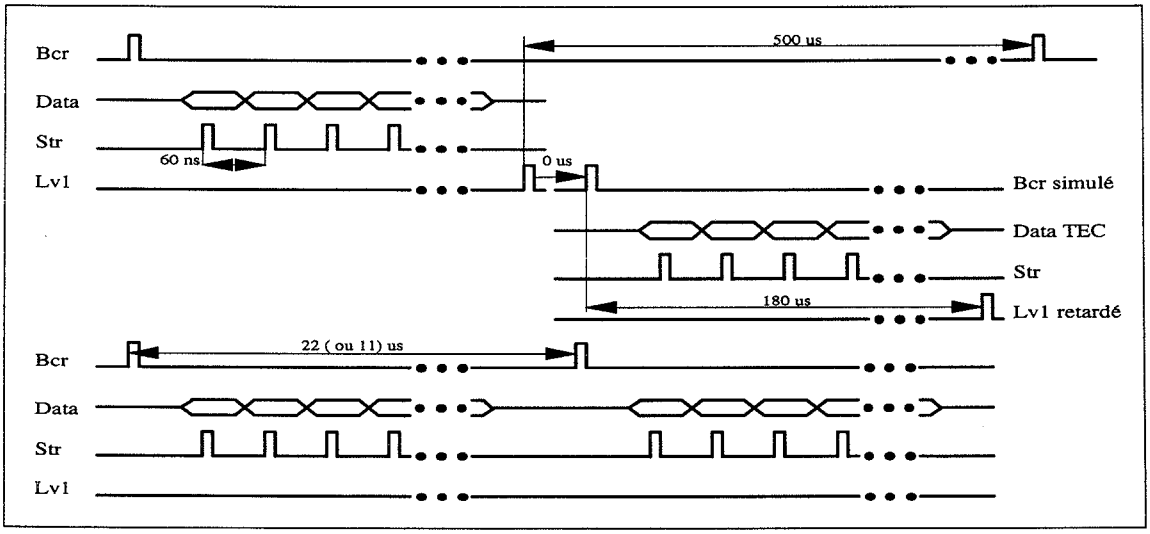


FIG. B.1 - *Séquence de réception des données sur les modules* Tmb

1. *Strobe*
2. *Beam Crossing*
3. *Level 1*

Comme le montre la figure B.1, les signaux de contrôle ne sont pas émis en même temps selon la nature du détecteur. On distingue les détecteurs *Non–Tec* du détecteur *Tec* qui génére sa propre séquence et ses signaux de contrôle. Pour chaque détecteur, les signaux de contrôle sont reçus sur le premier port puis distribués vers les autres modules par l'intermédiaire d'un chaînage.

B.3 La partie *Acquisition*

Chaque port est constitué:

- d'une mémoire *Fisrt In First Out*($Fifo$) profonde de 4096 mots de 2 octets,
- d'un compteur de mots($WordCount$) permettant de connaître le nombre de mots présents dans la mémoire *Fisrt In First Out*.

B.3.1 Utilisation de signaux de contrôle

Sur chaque module, la réception du signal *Bcr* remet à zéro les mémoires *Fisrt In First Out* et les compteurs de mots des quatres ports. Sur chaque port, un mot est écrit dans la mémoire *Fisrt In First Out* lors de la réception de l'impulsion *Str* correspondante. Le signal de contrôle *overflow* est activé si:

1. il y a des données présentes dans les mémoires *First In First Out* et si le signal *Lv1* est émis par le système de déclenchement de niveau–1,
2. lorsque une ou plusieurs mémoires *First In First Out* sont pleines.

B.3.2 Enumération des éléments fonctionnels

Les éléments fonctionnels de la partie *Acquisition* sont:

- un *Transputer T9000*,
- une mémoire dymanique profonde de 4Moctets extensible à 8 Moctets. Le code relatif à ce *Transputer* est résidant dans cette mémoire.
- une mémoire statique d'accès rapide (20ns) profonde de 128 Koctets extensible à 256Kotets. Elle est utilisée comme mémoire tampon pour les données expérimentales.
- un registre de commande contenant les signaux pour la commande des différents commutateurs et d'un registre d'état,
- un compteur de mots de 2 octets modulo 4096 et une mémoire *FIFO* profonde de 8 Koctets par port.
- un compteur($BcrCount$) dont le premier octet correspond au nombre d'impulsions *Bcr* et le second au nombre d'impulsions *Lv1* reçues par le module.

B.3.3 Organisation de l'espace adressable du *Transputer*

Les éléments fonctionnels sont répartis sur les quatres banques *PMI* simplifiant ainsi la logique de décodage de l'espace mémoire. En effet, chaque banque permet l'accès à une zone de l'espace mémoire, définie par son adresse de base et sa taille, elle dispose de son propre jeu de 3 signaux de contrôle permettant l'élaboration de la séquence d'accès en lecture et écriture pour cette zone [18, chapitre 10].

Les mémoires *FIFO* sont accessibles sur une zone mémoire profonde de la taille de ces mémoires pour permettre l'utilisation de l'instruction de transfert de blocs mis en œuvre par le

Transputer. Pour que chaque accès successif s'effectue à une adresse supérieure à la précédente, il faut que les mémoires *FIFO* soient accessibles en mode *device*.

Pour garantir que chaque accès aux registres de commande et d'état, aux différents compteurs ait effectivement lieu, il faut qu'ils soient accessibles en mode *device*. Pour de plus amples informations concernant le mode *device* confère [19, page 59].

Le compteur *BcrCount* est accessible à partir de plusieurs adresses réparties dans l'espace adressable.

Le tableau B.1 résume les différentes caractéristiques des quatres banques *PMI* et donne la répartition de l'espace mémoire adressable.

Banque	Elément		Adresse	Cycle	Taille (octets)	Bus (bits)	Mode d'accès
0	Mémoire Dynamique		#80000000	Rd/Wr	4 M	32	Non-Cache/Cache
					8 M	64	Cache
1	Registre d'Etat		#00000000	Rd	2	32	Device
	Registre de Commande		#00000004	Rd	2		
				Wr	4		
2	Port 0	BcrCount	#00010000	Rd	2	16	Device
		WordCount	#00010002		2		
		Fifo	#00012000		8 K		
	Port 1	BcrCount	#00014000		2		
		WordCount	#00014002		2		
		Fifo	#00016000		8 K		
	Port 2	BcrCount	#00018000		2		
		WordCount	#00018002		2		
		Fifo	#0001A000		8 K		
	Port 3	BcrCount	#0001C000		2		
		WordCount	#0001C002		2		
		Fifo	#0001E000		8 K		
3	Mémoire Statique		#10000000	Rd/Wr	128 K	32	Non-Cache/Cache
					256 K	64	Cache

TAB. B.1 - *Répartition de l'espace mémoire du* Transputer Acquisition

B.3.4 Description du Registre de Commande

Ce registre contient l'ensemble des signaux permettant la configuration de la partie *Acquisition*. On distingue deux types de signaux:

- les signaux fonctionnant en mode *Set-Reset*. Le tableau B.9 précise ce mode de fonctionnement.
- les signaux impulsionnels.

Le tableau B.2 énumère les différents bits utilisés par ce registre, précise le signal correspondant ainsi que la fonctionnalité associée.

B.3.5 Description du Registre d'Etat

Le tableau B.3 énumère les différents bits utilisés par ce registre et l'état correspondant.

B.4 La partie *Injection*

La partie *Injection* doit permettre de réaliser la séquence des données transmises à la partie *Acquisition*. Elle doit donc reproduire les signaux de contrôle de l'expérience *Bcr* et *Lv1* ainsi

Bit	Signal	Cycle		Fonctionnalité
0 16	*enlv1evt*	Wr/Rd Wr	active désactive	la réception du signal *Lv1* sur le canal *Event0* du *Transputer*
2 18	*enretsync*	Wr/Rd Wr	active désactive	le signal de synchronisation inter–module
3 19	*disdcntsyncin*	Wr/Rd Wr	désactive active	la réception par le module du signal de synchronisation inter–module généré par les modules précédents
5 21	*disdcovfli*	Wr/Rd Wr	désactive active	la réception par le module du signal de *overflow* généré par les modules précédents
6 22	*disexpovfl*	Wr/Rd Wr	désactive active	l'émission par le module du signal expérimental *overflow*
7 23	*seldcbcr*	Wr/Rd Wr	active désactive	la réception par le module des signaux *Bcr* et *Lv1* provenant du chaînage
8 24	*ndistr0*	Wr/Rd Wr	désactive active	la réception des données sur le port 0
9 25	*ndistr1*	Wr/Rd Wr	désactive active	la réception des données sur le port 1
10 26	*ndistr2*	Wr/Rd Wr	désactive active	la réception des données sur le port 2
11 27	*ndistr3*	Wr/Rd Wr	désactive active	la réception des données sur le port 3
12 28	*seltest*	Wr/Rd Wr	active désactive	la réception des données provenant de la partie *Injection*
13 29	*faulty*	Wr/Rd Wr	active désactive	le drapeau *faulty* et la diode luminescente associée
14 30	*Resetf&c*	Wr/Rd Wr	active désactive	le signal de mise à zéro des mémoires *FIFO* et des compteurs sur l'ensemble des ports
31	*ResetRegister*	Wr	*génére l'impulsion* de mise à zéro du registre de commande	

TAB. B.2 - *Description du Registre de Commande du* Transputer Acquisition

que le signal d'échantillonnage *Str*. Chaque port est constitué:

- d'une mémoire *First In First Out*($Fifo$) profonde de 4096 mots de 2 octets dans laquelle sont stokées les données d'un événement correspondant au port,
- d'un séquenceur permettant la lecture de la mémoire *First In First Out* à la vitesse de 60 ns par mot et générant le signal d'échantillonnage *Str* associé.

Les signaux de contrôle sont gérés à partir du registre de commande.

Bit	Signal	Etat	Fonctionnalité
0	*ovflp0t*	1	Overflow
1	*nef1p0t*	0	port 0 : mémoire *First In First Out* vide
2	*nff1p0t*	0	port 0 : mémoire *First In First Out* pleine
3	*nef1p1t*	0	port 1 : mémoire *First In First Out* vide
4	*nff1p1t*	0	port 1 : mémoire *First In First Out* pleine
5	*nef1p2t*	0	port 2 : mémoire *First In First Out* vide
6	*nff1p2t*	0	port 2 : mémoire *First In First Out* pleine
7	*nef1p3t*	0	port 3 : mémoire *First In First Out* vide
8	*nff1p3t*	0	port 3 : mémoire *First In First Out* pleine

TAB. B.3 - *Description du Registre d'Etat du* Transputer Acquisition

B.4.1 Enumération des éléments fonctionnels

Les éléments fonctionnels de la partie *Injection* sont:

- un *Transputer T9000*,
- une mémoire dymanique profonde de 4Moctets extensible à 8Moctets. Le code relatif à ce *Transputer* est résidant dans cette mémoire.
- une mémoire statique d'accès rapide (20ns) profonde de 128Koctets extensible à 256Kotets. Elle est utilisée comme mémoire tampon pour les données expérimentales à rejouer.
- un registre de commande contenant les signaux pour la commande des différents commutateurs et d'un registre d'état,
- un séquenceur et une mémoire *FIFO* profonde de 8Koctets par port.

B.4.2 Le séquenceur

Le séquenceur est piloté par une horloge à 16 Mhz. Cette horloge définit la vitesse de transfert des données, assure la synchronisation du signal d'échantillonage *Str* avec la lecture de la mémoire *FIFO* et l'envoi des données. Le fonctionnement du séquenceur est le suivant:

1. lorsque des données sont présentes dans la mémoire *FIFO*, une impulsion du signal de contrôle *Bcr* démarre la séquence d'émission des données à la vitesse de 60ns par mots. Le séquenceur ne peut démarrer que s'il y a des données dans la mémoire *FIFO*.
2. lorsque la mémoire *FIFO* est vide, le séquenceur s'arrête.

B.4.3 Organisation de l'espace adressable du *Transputer*

Les éléments fonctionnels sont répartis dans les quatres banques *PMI*. Le tableau B.4 résume les différentes caractéristiques de ces banques et donne la répartition de l'espace mémoire adressable.

Banque	Elément	Adresse	Cycle	Taille (octets)	Bus (bits)	Mode d'accès
0	Mémoire Dynamique	#80000000	Rd/Wr	4 M	32	Non-Cache/Cache
				8 M	64	Cache
1	Registre d'Etat	#00000000	Rd	2	32	Device
	Registre de Commande	#00000004	Rd	2		
			Wr	4		
2	Fifo 0	#00010000	Wr	8 K	16	Device
	Fifo 1	#00012000				
	Fifo 2	#00014000				
	Fifo 3	#00016000				
3	Mémoire Statique	#10000000	Rd/Wr	128 K	32	Non-Cache/Cache
				256 K	64	Cache

TAB. B.4 - *Répartition de l'espace mémoire du* Transputer Injection

B.4.4 Description du Registre de Commande

Le tableau B.5 énumère les différents bits utilisés par ce registre, précise le signal correspondant ainsi que la fonctionnalité associée.

Bit	Signal	Cycle	Fonctionnalité
0 16	*enevt1*	Wr/Rd Wr	active désactive	la réception du signal *notsyncl* sur le canal *Event1* du *Transputer*
1 17	*enevt2*	Wr/Rd Wr	active désactive	la réception du signal *ovflp0t* sur le canal *Event2* du *Transputer*
2 18	*enevt3*	Wr/Rd Wr	active désactive	la réception du signal *nefp0–3t* sur le canal *Event3* du *Transputer*
3 19	*ntestsyncl*	Wr/Rd Wr	active désactive	le signal de synchronisation local
8	*lv1p0t*	Wr	*génére l'impulsion* de contrôle *lv1p0t*
9	*bcrp0t*	Wr	*génére l'impulsion* de contrôle *bcrp0t*
10 11	0 0 *str0* 0 1 *str1* 1 0 *str2* 1 1 *str3*	Wr	*génére l'impulsion* de contrôle *Str* sur le port sélectionné lorsque l'*impulsion Strobe* est présente
12	*Strobe*	Wr	*génére l'impulsion* de contrôle *Strobe* sur le port sélectionné par les bits 10–11
13 29	*faulty*	Wr/Rd Wr	active désactive	le drapeau *faulty* et la diode luminescente associée
14 30	*Resetf&c*	Wr/Rd Wr	active désactive	le signal de mise à zéro des mémoires *FIFO* sur l'ensemble des ports
31	*ResetRegister*	Wr	*génére l'impulsion* de mise à zéro du registre de commande

TAB. B.5 - *Description du Registre de Commande du* Transputer Injection

B.4.5 Description du Registre d'Etat

Le tableau B.6 énumère les différents bits utilisés par ce registre et l'état correspondant.

Bit	Signal	Etat	Fonctionnalité
0	*ovflp0t*	1	Overflow (identique au signal de la partie *Acquisition*)
1	*nef1p0t*	0	port 0 : mémoire *First In First Out* vide
2	*nff1p0t*	0	port 0 : mémoire *First In First Out* pleine
3	*nef1p1t*	0	port 1 : mémoire *First In First Out* vide
4	*nff1p1t*	0	port 1 : mémoire *First In First Out* pleine
5	*nef1p2t*	0	port 2 : mémoire *First In First Out* vide
6	*nff1p2t*	0	port 2 : mémoire *First In First Out* pleine
7	*nef1p3t*	0	port 3 : mémoire *First In First Out* vide
8	*nff1p3t*	0	port 3 : mémoire *First In First Out* pleine
9	*mnotsyncl*	1	signal de synchronisation inter–module mémorisé

TAB. B.6 - *Description du Registre d'Etat du* Transputer Injection

B.5 Les synchronisations matérielles

Le *Transputer* met en œuvre des canaux de synchronisation *Event* qui permettent des synchronisations entre des signaux externes et des tâches internes [18]. Le principal avantage de l'utilisation des canaux *Event* est que leur accès constitue un point de débranchement par rapport à une boucle d'attente sur un bit d'un registre d'état. De plus si la tâche en attente est une tâche *haute priorité*, la synchronisation est alors équivalente à une interruption puisque les tâches *basse priorité* sont interrompues. Dans notre application, les canaux *Event* sont utilisés comme des canaux d'entrée.

B.5.1 Fonctionnement des canaux *Event*

Lorque le signal connecté à la pin d'entrée du canal *Event* passe à l'état haut, le canal d'entrée est prêt. Si une tâche est en attente sur ce canal, elle est activée et une impulsion de largeur *ProcClockOut* est générée sur la pin de sortie du canal *Event*. Tant qu'il n'y a pas de tâches disponibles pour recevoir la synchronisation, le processeur ne génére pas l'impulsion sur la pin de sortie du canal *Event*. Pour plus de détails, confère [20, page 21] et [19, page 155–156].

B.5.2 Description des synchronisations

Dans la partie *Acquisition*, on utilise une seule synchronisation matérielle correspondant au signal *Lv1* indiquant que l'événement est validé par le système de déclenchement de niveau–1 et que l'information contenue dans les mémoires *FIFO* doit être transférée dans la mémoire–tampon.

Dans la partie *Injection*, on utilise trois synchronisations matérielles correspondant:

1. au signal de synchronisation inter–module *notsyncl* indiquant que l'ensemble des modules sont dans le même état,

2. au signal *ovflp0t* indiquant que les mémoires *FIFO* de la partie *Acquisition* sont disponibles pour recevoir le prochain événement,

3. aux *"Et logique"* des signaux *nef1p0–3t* indiquant que toutes les mémoires *FIFO* de la partie *Injection* sont disponibles pour recevoir le prochain événement.

B.5.3 Mise en oeuvre

Pour chaque signal de synchronisation, on définit un signal de contrôle qui autorise l'accès du signal de synchronisation au canal *Event* correspondant. Le signal de contrôle est positionné par le registre de commande. Lorsque le signal de contrôle autorise l'accès au canal *Event*, l'arrivée du signal de synchronisation est mémorisée dans une bascule, et induit un niveau haut sur la pin d'entrée du canal *Event*. L'impulsion générée par le processeur sur la pin de sortie du canal *Event* est utilisée pour mettre à zéro la bascule et inhiber la mémorisation.

Le tableau B.7 résume les différentes caractéristiques des synchronisations matérielles misent en œuvre sur le module *Tmb*

Partie	Signal	Contrôle	Canal	Fonctionnalité
Acqusition	*Lv1*	*enlv1evt*	Event 0	autorise le transfert mémoire *FIFO* vers mémoire–tampon
Injection	*notsyncl*	*enevt0*	Event 0	synchronisation inter–module
	ovflp0t	*enevt1*	Event 1	indique que les mémoires *FIFO* de la partie *Acquisition* sont vides
	nef1p0–3t	*enevt2*	Event 2	indique que les mémoires *FIFO* de la partie *Injection* sont vides

TAB. B.7 - *Caractéristiques des synchronisations matérielles d'un module* Tmb

B.6 Le chaînage

Le chaînage permet de distribuer les signaux de contrôle sur les modules et de les synchroniser entre–eux.

B.6.1 Description du chaînage

Les signaux de contrôle *Bcr* et *Lv1*

Ces signaux sont disponibles sur le premier port relatif à chaque type de détecteurs. Il sont ensuite propagés sur les différents modules associés au même type de détecteur.

Le signal *overflow*

Le signal *overflow* est un signal actif au niveau logique 1. Le signal *overflow* se propage du dernier module vers le premier module. La construction du signal de contrôle *overflow* s'effectue de la façon suivante:

- chaque module génére son propre signal *overflow local*. Ce signal *overflow local* active une diode luminescente.
- sur chaque module, on effectue le *"OU logique"* du signal *overflow* des modules précédents avec le signal *overflow local*.

Ainsi sur chaque module, le signal *ovflp0t* représente le *"OU logique"* des signaux *overflow* des modules précédents et du module.

Le signal de synchronisation inter–module *notsync*

Le signal *notsync* est un signal actif au niveau logique 0. Ce signal se propage du dernier module vers le premier module, il est identique pour tous les modules. Il est construit à partir des signaux *ntestsyncl* correspondant à la demande d'une synchronisation inter–module par les modules. Sur chaque module , on effectue le *"OU logique"* du signal *ntestsyncl* local avec celui du module provenant des modules précédents. Sur le dernier module, le signal résultant du *"OU logique"* est envoyé sur le signal *notsync*. Lorsque le signal *notsync* est actif son état est mémorisé dans le registre d'état de la partie *Injection*. La réception du signal *notsync* sur le canal *Event* doit activer une tâche *haute priotité* dont la première opération est de désactiver la demande de synchronisation inter–module, ceci pour assurer la cohérence temporelle des demandes sur les différents modules. La désactivation de la demande de synchronisation inter–module inhibe la mémorisation de l'état du signal dans le registre d'état.

B.6.2 Configuration du chaînage suivant la position du module *Tmb* dans le système d'acquisition de l'expérience L3

Les signaux de contrôle *Bcr* et *Lv1*

Sur le premier module relatif à chaque type de détecteur, on doit sélectionner suivant le mode de fonctionnement d'utiliser les signaux de contrôle provenant soit de l'expérience, soit de la partie *Injection*. Ce multiplexage est obtenu à partir du signal *seltest*. Quel que soit le type du détecteur, pour les autres modules on choisit les signaux de contrôle provenant du chaînage. Cette sélection est obtenue à partir du signal *seldcbcr*.

Les signaux *seltest* et *seldcbcr* sont accessibles à partir du regsitre de commande de la partie *Acquisition*.

Le signal *overflow*

Le signal *overflow* se propage du dernier module vers le premier. Donc sur le dernier module, on désactive la réception du signal d'entrée par contre sur les autres modules on active la réception du signal d'entrée. Cette sélection est obtenue à partir du signal *disdcovfli*. Sur le premier module, lorsque que l'on travaille avec des données expérimentales il faut que le signal

overflow soit émis vers le sytème d'acquisition. Tandis que lorsque l'on travaille avec des données de test, il ne faut pas que le système d'acquisition soit perturbé par le test, on masque alors le signal *overflow* au système d'acquisition. Cette sélection est obtenue à partir du signal *disexpovfl*.

Les signaux *disdcovfl* et *disexpovfl* sont accessibles à partir du registre de commande de la partie *Acquisition*.

Le signal de synchronisation inter–module *notsync*

Ce signal de synchronisation est uniquement utilisé lorsque l'on travaille avec des données de test. L'ensemble des demandes de synchronisation inter–module se propage jusqu'au dernier module qui lui renvoie le signal *notsync* vers tous les modules. La propagation est obtenue en désactivant le signal *disdcntsyncin* sur le premier module et en l'activant sur les autres. Le retour du signal de synchronisation global est obtenu en activant le signal *enretsync* uniquement sur le dernier module.

Les signaux *ntestsyncl*, *disdcntsyncin* et *enretsync* sont accessibles à partir du registre de commande de la partie *Injection*.

Tableau récapitulatif

Le tableau B.8 résume l'initialisation du chaînage d'un module *Tmb* suivant sa position dans le système d'acquisition.

Mode	Détecteurs / Tmb / Signal	Non–TEC 0	Non–TEC ...	Non–TEC 8	TEC 9	TEC 10	TEC 11
Acquisition *seltest 'off'*	*seldcbcr*	*'off'*	*'on'*	*'on'*	*'off'*	*'off'*	*'on'*
	disdcovfl	*'on'*	*'on'*	*'on'*	*'on'*	*'on'*	*'off'*
	disexpovfl	*'off'*	*'on'*	*'on'*	*'on'*	*'on'*	*'on'*
	disdcsyncin	*'on'*	*'on'*	*'on'*	*'on'*	*'on'*	*'on'*
	enretsync	*'off'*	*'off'*	*'off*	*'off'*	*'off'*	*'off'*
Injection *seltest 'on'*	*seldcbcr*	*'off'*	*'on'*	*'on'*	*'off'*	*'off'*	*'on'*
	disdcovfl	*'off'*	*'off'*	*'off'*	*'off'*	*'off'*	*'on'*
	disexpovfl	*'on'*	*'on'*	*'on'*	*'on'*	*'on'*	*'on'*
	disdcsyncin	*'on'*	*'off'*	*'off'*	*'off'*	*'off'*	*'off'*
	enretsync	*'off'*	*'off'*	*'off'*	*'off'*	*'off'*	*'on'*

TAB. B.8 - *Configuration du chaînage d'un module* Tmb *suivant sa position dans le système d'acquisition*

B.7 Caractéristiques techniques

B.7.1 Extension des mémoires à 64 bits

L'extension des mémoires à 64 bits est obtenue en permuttant le bit adresse 2 avec le bit d'adresse 23 pour la mémoire dynamique et le bit d'adresse 18 pour la mémoire statique. Cette permutation est effectuée par un commutateur physique. Lorsque l'on utilise la mémoire dynamique avec un bus de données de 32 bit, on peut travailler en mode *page*. Tandis que si on utilise la mémoire dynamique avec un bus de données de 64 bits, le fonctionnement en mode *page* n'est plus autorisé.

Bit_{i+16}	Bit_i	Signal Bit_i
0	0	inchangé
0	1	1
1	0	0
1	1	inchangé

TAB. B.9 - *Table de vérité des signaux fonctionnant en mode* Set–Reset

B.7.2 Table de vérité des signaux *Set–Reset* des registres de commande

B.7.3 Temps d'accès des différents périphériques

Les temps d'accès sont donnés pour un *Transputer T9000* travaillant à 20 Mhz. La base de temps est un cycle correspondant à 50 ns. Le tableau B.10 donne les différents temps d'accès des éléments adressables.

périphérique	Accès	Cycle.Time	Bus.Release
Mémoire Dynamique	Wr/Rd	3	1
Mémoire Statique	Wr/Rd	2	1
Mémoire *FIFO*	Wr	2	1
	Rd	2	1
Registres	Wr/Rd	2	1

TAB. B.10 - *Temps d'accès des différents périphériques*

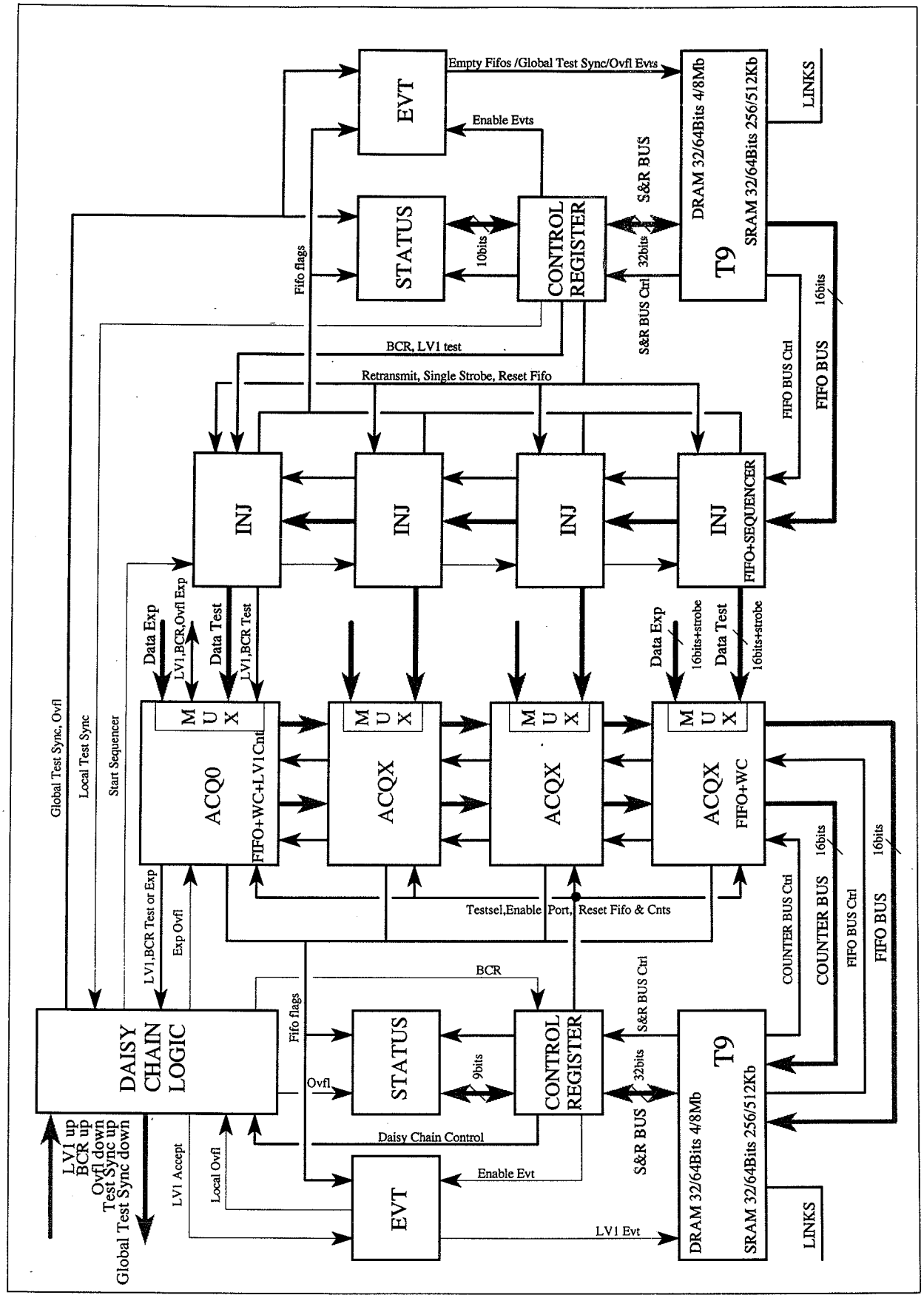


FIG. B.2 - *Bloc diagramme du module* TMB

Annexe C

Le logiciel Mbm *Memory Buffer Management*

C.1 Définition de quelques concepts propres à la programmation paralléle

Zone critique et exclusion mutuelle

Une *zone critique* est une partie de programme qui ne peut être exécutée simultannément par des processus indépendants[14]. Lorsqu'un processus entre dans la *zone critique*, il en doit interdire l'accès aux autres, d'ou le terme d'exclusion mutuelle.

Les sémaphores

Le sémaphore est un objet logiciel permettant d'opérer à la synchronisation de processus asynchrones, et par là même de résoudre les problèmes d'exclusions mutuelles. C'est un objet fourni par une bibliothèque de programmation parallèle. Un sémaphore peut être binaire ou n–aire. Un sémaphore binaire ne peut prendre que les valeurs 0 ou 1. Un sémaphore n–aire peut prendre les valeurs entières de 0 à n. Un sémaphore est manipulable par l'intermédiaire de deux opérations:

- *SemWait*: lorsque le processus accède le sémaphore, si la valeur du sémaphore est:
  - supérieure à zéro, elle est décrémentée de 1 et le processus continue son exécution,
  - est nulle, le processus est mis en attente.
- *SemSignal*: incrémente de 1 la valeur du sémaphore si celle–ci est non nulle. Si la valeur du sémaphore vaut zéro:
  - s'il y a des processus en attente, un processus est reveillé et la valeur du sémaphore reste à zéro,
  - s'il n'y a plus de processus en attente, la valeur du sémaphore est mise à 1.

Les Tâches *synchrones*

Lorsqu'une tâche active une liste de tâches *synchrones*, elle devient inactive et reste inactive tant que l'ensemble des tâches *synchrones* n'ont pas terminé leur exécution.

C.2 Schéma de base (*"Buffer Management model"*)

La gestion du flux de données est identique sur l'ensemble des sites opérant sur les données.

Sur chaque site, il s'agit de gérer la présence de processus producteurs et consommateurs de données, autour d'une zone de mémoire commune partitionnée organisée en tampon. On définit pour cela un certain nombre d'opérateurs (typiquement réception, traitement et émission) correspondant aux différentes opérations à effectuer sur les données transitant par ce site. Ces données sont stockées dans une zone mémoire–tampon de profondeur fixe définie à l'initialisation et partagée en emplacements de la taille d'un événement. L'ordre d'exécution des opérateurs définit le cycle opératoire. Chaque événement subit l'ensemble des opérations du cycle opératoire. A chaque opérateur, on associe une ou plusieurs tâches. Ainsi:

- au niveau de chaque site, le modèle définit une *structure pipeline* de profondeur la taille du cycle opératoire,
- au niveau de chaque opérateur, le modèle permet un *parallélisme réplicatif*. A chaque opérateur est associé une ou plusieurs tâches concurrentes autorisant le cas échéant le traitement en parallèle de plusieurs événements.

Un des avantages de ce schéma est son extensibilité vis à vis du nombre d'opérateurs constituant un cycle opératoire et du nombre de tâches qui lui sont associées. Différentes mises en œuvre de ce modèle sont possibles selon le mode de gestion mémoire utilisé. Le logiciel **MBM** décrit ici se distingue quelque peu d'une approche classique où la zone mémoire est gérée en simple **FIFO** (*First In First Out*).

C.3 Gestion de l'espace mémoire

Une gestion de l'espace mémoire construite à partir d'un pointeur incrémental modulo le nombre d'emplacements associé à chaque opérateur permet en effet la construction d'une *structure pipeline* et autorise le *parallélisme réplicatif* au niveau d'un opérateur. Cependant ce type de gestion présente certaines restrictions. Il ne permet pas d'exploiter au mieux l'espace mémoire dès que plusieurs tâches sont associées à un même opérateur. En effet, les emplacements sont utilisés de façon consécutive ce qui implique que chaque événement est directement corrélé avec un emplacement. Les temps de traitement pouvant varier d'un événement à l'autre, il peut se trouver que l'emplacement $p+1$ soit disponible avant l'emplacement p. Il est alors impossible d'utiliser cet emplacement libre: d'où l'idée de décorréler totalement les emplacements des événements qui y transitent et de privilégier la disponibilité des emplacements. On associe pour cela à chaque opérateur une *liste de disponibilité* contenant à chaque instant la référence des emplacements dont celui–ci peut disposer. Cette liste de disponibilité est gérée en *FIFO*, ainsi le premier emplacement rendu disponible par l'opérateur précédent subira en premier le traitement relatif à l'opérateur correspondant à la liste de disponibilité. Avec ce type de gestion, le *parallélisme réplicatif* est optimum.

C.4 Gestion des opérateurs

Etant donné un cycle opératoire comprenant n opérateurs notés Op_i où $i \in \{0, \ldots, n-1\}$, les opérateurs Op_0 et Op_{n-1} gèrent respectivement les flux entrant et sortant du cycle opératoire. A chaque opérateur est associé une liste de disponibilité ou une queue d'accès à la mémoire–tampon qui contient à chaque instant la liste des emplacements disponibles pour cet opérateur. L'ordre des emplacements dans la liste de disponibilité de l'opérateur Op_i reflète:

- soit l'ordre de traitement des emplacements par l'opérateur Op_{i-1}, on dit alors que les opérateurs sont *synchrones*,
- soit l'ordre de libération des emplacements par l'opérateur Op_{i-1}, on dit alors que les opérateurs sont *asynchrones*

selon le mode de synchronisation entre les opérateurs Op_{i-1} et Op_i.

Le cycle opératoire s'exécute de la façon suivante:

- A l'initialisation, la liste de disponibilité de l'opérateur Op_0 contient l'ensemble des emplacements de la zone mémoire–tampon. Les listes de disponibilité des autres opérateurs sont vides.
- Considérons l'opérateur Op_i du cycle opératoire. Une des tâches associée à l'opérateur Op_i consulte la liste de disponibilité de cet opérateur et y retire la référence de l'emplacement dont elle peut disposer. A la fin de son traitement, l'emplacement est rendu disponible pour l'ensemble des tâches de l'opérateur suivant en insérant la référence de cet emplacement dans la liste de disponibilité de l'opérateur Op_{i+1}.
- En fin de cycle l'opérateur Op_{n-1} insère l'emplacement rendu disponible à la fin de son traitement dans la liste de disponibilité de l'operateur Op_0. Ainsi le cycle opératoire est bouclé.

La liste de disponibilité de l'opérateur Op_i est donc une ressource partagée entre l'opérateur Op_{i-1} et l'opérateur Op_i pour l'ensemble des tâches associées à ces 2 opérateurs. Un sémaphore binaire est associé à chaque liste de disponibilité autorisant l'accès à une seule tâche.

C.5 Synchronisation des opérateurs

Pour assurer la synchronisation entre les opérateurs, on associe à chaque liste de disponibilité (en plus du sémaphore d'accès) un sémaphore de synchronisation dont la valeur à chaque instant représente le nombre d'éléments dans la liste de disponibilité. Dans la plupart des cas de production ou consommation, le passage à zéro de ce sémaphore bloque le mécanisme de suppression d'éléments dans la liste de disponibilité puisque celle–ci est alors vide. On verra plus loin que l'on peut concevoir d'autres synchronisations selon les besoins.

Chaque sémaphore de synchronisation est un sémaphore de valeur p, oú p est le nombre d'emplacements de la zone mémoire–tampon. A l'initialisation le sémaphore de l'opérateur Op_0 est positionné à p et les autres à zéro.

C.6 Gestion des flux de données

On peut gérer le flux de données de différentes manières. La plus naturelle est de considérer que l'opérateur Op_i traite le premier emplacement rendu disponible par l'opérateur Op_{i-1}. Dans ce cas, on considère que l'opérateur Op_i est *asynchrone*. En effet, dès que plusieurs tâches associées à Op_{i-1} travaillent en parallèle, l'ordre de disponibilité des emplacements pour Opi n'est pas nécessairement identique à l'ordre de disponibilité des emplacements pour Op_{i-1}. Cependant il peut être intéressant de respecter l'ordre des données au cours du cycle opératoire, en imposant par exemple que l'opérateur Op_i traite l'ensemble des emplacements dans le même ordre que l'opérateur Op_{i-1}. Dans ce cas, on considère que l'opérateur Op_i est *synchrone*. Comme l'événement subit l'ensemble des opérations du cycle opératoire, il n'y a aucune relation d'ordre entre Op_{n-1} et Op_0. Par conséquent, l'opérateur Op_0 est toujours de type *asynchrone*.

En plus des gestions de flux *synchrone* ou *asynchrone* on distingue:

1. la gestion de flux de données bloquante, qui stoppe l'exécution d'un opérateur si sa liste de disponibilité est vide,

2. la gestion de flux de données non–bloquante qui autorise un opérateur à prélever un emplacement dans la liste de disponibilité des autres opérateurs du cycle si sa liste de disponibilité est vide. Ce mode est utilisé dans des cas très particuliers comme celui d'un opérateur producteur de données que l'on souhaite rendre non–bloquant vis à vis de la production de données. Un tel mécanisme autorise la production de données alors que celle–ci n'ont pas été consommées par les opérateurs avals.

La figure C.1 présente la structure d'un gestionnaire de flux de données.
On définit les variables suivantes:

- p: nombre d'emplacements de la zone mémoire–tampon,
- n: nombre d'opérateurs du cycle opératoire,
- Op_i: le ième opérateur du cycle opératoire,
- $PrOp_i$: nombre de tâches associées au ième opérateur,
- $SemOp_i$: le sémaphore de synchronisation du ième opérateur,
- $LdOp_i$: la liste de disponibilité du ième opérateur,

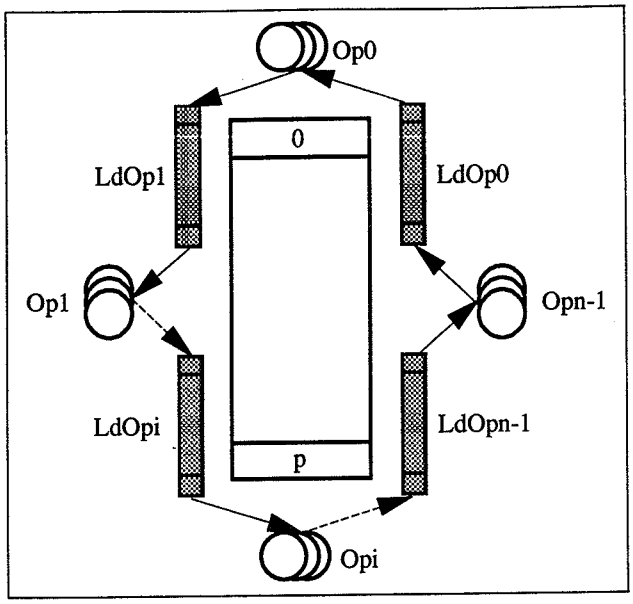


FIG. C.1 - *Structure d'un gestionnaire de flux de données*

- $SemLdOp_i$: le sémaphore d'accès de la liste de disponibilité du ième opérateur.

et la notation:

- *type(Op)*: pour le type de l'opérateur *Async* ou *Sync*,
- *value(Sm)*: pour la valeur du sémaphore Sm,
- *size(Ld)*: pour la profondeur de la liste de disponobilité Ld.

On a tout instant et $\forall i \in \{0, \ldots, n-1\}$

- value($SemOp_i$) = size($LdOp_i$),
- $0 \leq$ value($SemOp_i$) $\leq p$,
- $0 \leq$ value($SemLdOp_i$) ≤ 1.

C.6.1 Gestion de flux de données bloquante contenant des opérateurs asynchrones

Ce mode est typiquement celui utilisé pour la gestion des flux de données expérimentales sur les sites *TMB* et sur les sites de *traitement*. L'ensemble des données suivent le cycle opératoire. Si pour une raison quelconque la liste de disponibilité d'un opérateur est vide l'ensemble de tâches qui lui sont associées sont alors suspendues, en ce sens la gestion de flux est considérée comme *bloquante*.

Les figures C.2 et C.3 montrent deux exemples de gestion de flux de données bloquante impliquant respectivement 2 et 3 opérateurs asynchrones.

Description du cycle opératoire relatif à cette gestion:

Initialisation :

- $\forall i \in \{0, \ldots, n-1\}$ type(Op_i) = Async;
- size($LdOp_0$) = p, value($SemOp_0$) = p;
- pour $\forall i \in \{1, \ldots, n-1\}$ size($LdOp_i$) = 0,value($SemOp_i$) = 0.

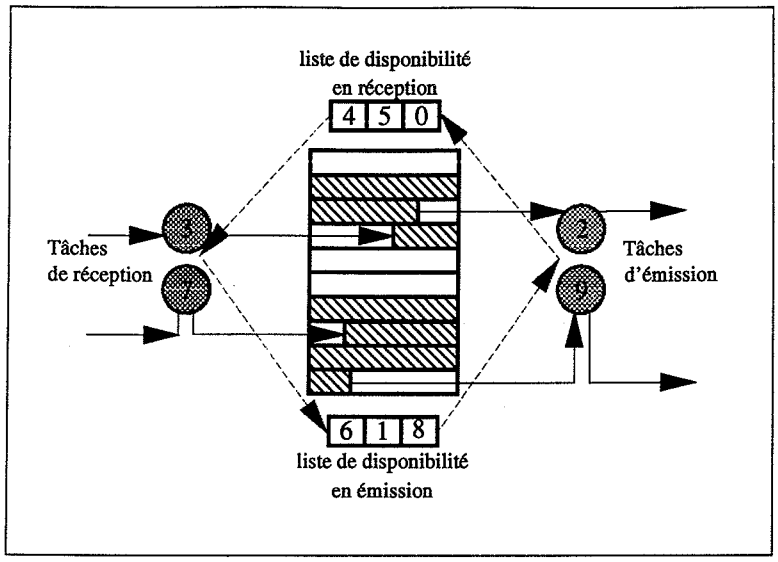


FIG. C.2 - *Exemple de gestion de flux bloquante : Schéma de type réception/émission.*

Déroulement : pour chaque opérateur Op_i, $i \in \{0,\ldots,n-1\}$, chaque tâche j, $j \in \{0,\ldots,PrOp_i-1\}$, doit exécuter la séquence suivante:

1. Demande d'emplacement *Event*: SemWait($SemOp_i$),
2. Accès et récupération d'un emplacement *Event* dans $LdOp_i$:
   (a) Demande d'accès à $LdOp_i$: SemWait($SemLdOp_i$),
   (b) Récupération d'un emplacement *Event* dans $LdOp_i$,
   (c) Libére l'accès de $LdOp_i$: SemSignal($SemLdOp_i$),
3. Exécution de l'opération relative à l'opérateur op_i,
4. Accès et insertion de l'emplacement *Event* dans $LdOp_{i+1}$,
   (a) Demande d'accès à $LdOp_{i+1}$: SemWait($SemLdOp_{i+1}$),
   (b) Insertion de l'emplacement *Event* dans $LdOp_{i+1}$,
   (c) Libére l'accès de $LdOp_{i+1}$: SemSignal($SemLdOp_{i+1}$),
5. Signale l'insertion de l'emplacement *Event* à Op_{i+1} : SemSignal($SemOp_{i+1}$).

C.6.2 Gestion de flux de données non–bloquante contenant des opérateurs asynchrones

Il est parfois intéressant d'être toujours disponible à recevoir des données à l'entrée du cycle opératoire et en contrepartie d'accepter que l'ensemble des données ne suivent pas tout le cycle opératoire. Dans notre cas, ce mode concerne typiquement la production des messages d'avertissement destinés à la surveillance du système.

Le principe de fonctionnement est le suivant: si pour une raison quelconque la liste de disponibilité d'un opérateur est vide celui–ci va alors consulter une à une les listes de disponibilité des opérateurs suivants jusqu'à disposer d'un emplacement, en ce sens la gestion du flux est considérée comme totalement *non–bloquante*. Pour la production des messages d'avertissement, on définit un cycle opératoire de taille 2. On trouve ci–après la description relative à ce cas.

Description du cycle opératoire relatif à cette gestion:

Initialisation :

- $\forall i \in \{0,\ldots,1\}$ type(Op_i) = Async;

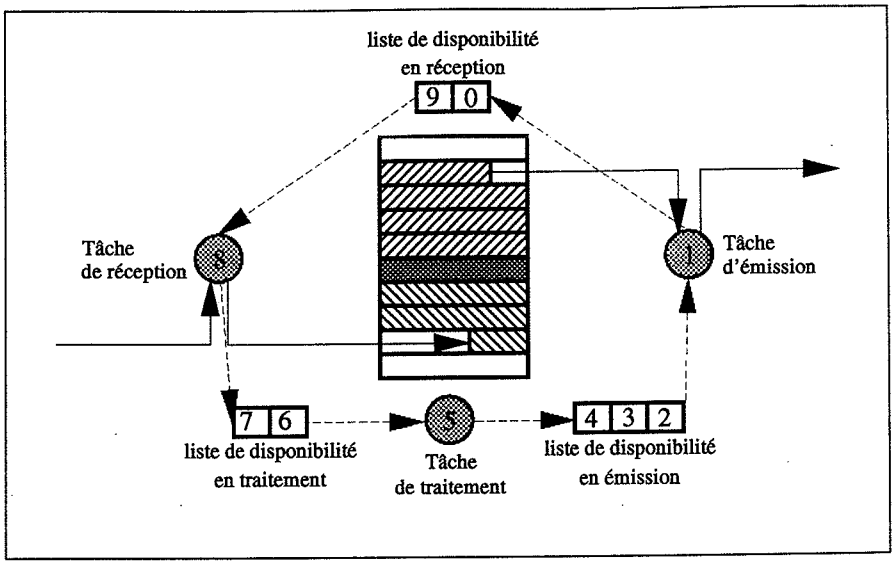


FIG. C.3 - *Exemple de gestion de flux bloquante: Schéma de type réception/traitement/émission.*

- size($LdOp_0$) = p, value($SemOp_0$) = p;
- size($LdOp_1$) = 0,value($SemOp_1$) = 0.

Déroulement : pour l'opérateur Op_0, chaque tâche j, $j \in \{0, \ldots, PrOp_0 - 1\}$, doit exécuter la séquence suivante:

- Cas où value($SemOp_0$) ≥ 0:
  1. Demande d'emplacement *Event*: SemWait($SemOp_0$),
  2. Accès et récupération d'un emplacement *Event* dans $LdOp_0$,
  3. Exécution de l'opération relative à l'opérateur op_0,
  4. Accès et insertion de l'emplacement *Event* dans $LdOp_1$,
  5. Signale l'insertion de l'emplacement *Event* à Op_1 : SemSignal($SemOp_1$).
- Cas où value($SemOp_0$) = 0:
  1. Demande d'emplacement *Event*: SemWait($SemOp_1$),
  2. Accès et récupération d'un emplacement *Event* dans $LdOp_1$,
  3. Exécution de l'opération relative à l'opérateur op_0,
  4. Accès et insertion de l'emplacement *Event* dans $LdOp_1$,
  5. Signale l'insertion de l'emplacement *Event* à Op_1 : SemSignal($SemOp_1$).

C.6.3 Gestion de flux de données bloquante contenant des opérateurs asynchrones et synchrones

Ce mode sera typiquement utilisé pour la gestion du flux de données expérimentales sur l'interface Transputer–Fastbus *Ft9000*. Pour que l'opérateur Op_i soit *synchrone* avec l'opérateur Op_{i-1}, il est nécessaire de sauvegarder l'ordre de traitement des emplacements *Event* établi par ce dernier. L'ordre est repéré par l'accès des différentes tâches associées à Op_{i-1} à la liste de disponibilité $LdOp_{i-1}$. Donc les informations qui permettent de synchroniser deux opérateurs sont:

- la référence j des tâches associées à Op_{i-1}, $j \in \{0, \ldots, PrOp_{i-1}\}$,
- les références des emplacements *Event* traités par chaque tâche j.

La référence des tâches constitue un flux de contrôle distinct du flux de données mais géré de la même façon. On définit alors un cycle opératoire noté *Ordre* de taille 2 incluant les opérateurs Op_{i-1} et Op_i. Ce cycle opératoire est associé à l'opérateur Op_i. Les 2 listes de disponibilité associées à ce cycle sont respectivement notée $Ordre.LdOp_{i,0}$ et $Ordre.LdOp_{i,1}$. Dans ces listes de disponibilité transite la référence des tâches de l'opérateur Op_{i-1}. Le contenu de la liste de disponibilité $Ordre.LdOp_{i,1}$ reflète directement l'ordre d'accès des tâches j à $LdOp_{i-1}$. De plus, on associe à chaque tâche j, $j \in \{0, \ldots, PrOp_{i-1} - 1\}$, une liste de disponibilité $LdOp_{i,j}$ et un sémaphore de synchronisation $SemOp_{i,j}$. Dans les listes $LdOp_{i,j}$ transitent les références des emplacements *Event* traités par chacune des tâches j. Chaque tâche associée à l'opérateur Op_i consulte la liste de disponibilité $Ordre.LdOp_{i,1}$ pour y retirer la référence j de la liste de disponibilité $LdOp_{i,j}$ où est stocké la référence du prochain emplacement *Event* à traiter.

La figure C.4 montre un exemple de gestion de flux bloquante contenant deux opérateurs *asynchrones* et un opérateur *synchrone*.

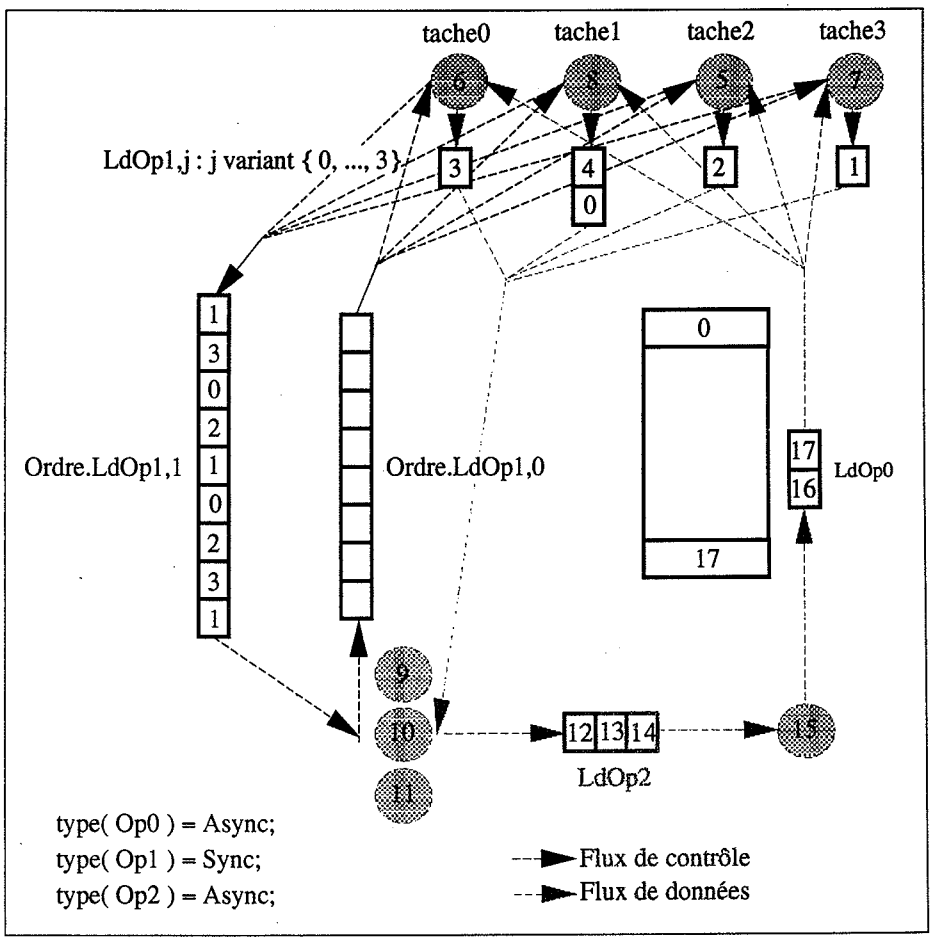


FIG. C.4 - *Exemple de fonctionnement d'un gestionnaire de flux de données comprenant 3 opérateurs dont un synchrone.*

Description du cycle opératoire relatif à cette gestion:

Initialisation :

Op_{i-1} :

- type(Op_{i-1}) = Async;
- si $i - 1 = 0$, size($LdOp_{i-1}$) = p, value($SemOp_{i-1}$) = p;
- si $i - 1 \neq 0$, size($LdOp_{i-1}$) = 0, value($SemOp_{i-1}$) = 0;

Op_i :

- $\forall i \neq 0$ type(Op_i) = Sync;
- $\forall i \neq 0, \forall j \in \{0, \ldots, PrOp_i - 1\}$ size($LdOp_{i,j}$) = 0, value($SemOp_{i,j}$) = 0;
- size($Ordre.LdOp_{i,0}$) = p, value($Ordre.SemOp_{i,0}$) = p;
- size($Ordre.LdOp_{i,1}$) = 0, value($Ordre.SemOp_{i,1}$) = 0;

Op_{i+1} :

- type(Op_{i+1}) = Sync ou Async;
- $\forall j \in \{0, \ldots, PrOp_{i+1} - 1\}$ size($LdOp_{i+1,j}$) = 0, value($SemOp_{i+1,j}$) = 0;
- size($Ordre.LdOp_{i+1,0}$) = p, value($Ordre.SemOp_{i+1,0}$) = p;
- size($Ordre.LdOp_{i+1,1}$) = 0, value($Ordre.SemOp_{i+1,1}$) = 0;

Déroulement :

Op_{i-1} : séquence exécutée par une tâche j, $j \in \{0, \ldots, PrOp_{i-1} - 1\}$

1. Demande d'emplacement *Event*: SemWait($SemOp_{i-1}$),
2. Accès et récupération de l'emplacement *Event* dans $LdOp_{i-1}$,
3. Mémorisation de l'ordre de traitement des emplacements *Event* par Op_{i-1} :
   (a) Demande d'un élément de liste: SemWait($Ordre.SemOp_{i,0}$),
   (b) Accès et récupération de l'élément dans $Ordre.LdOp_{i,0}$,
   (c) Sauvegarde de la référence de $LdOp_{i,j}$ dans l'élément,
   (d) Accès et insertion de l'élément dans $Ordre.LdOp_{i,1}$,
   (e) Signale l'insertion de l'élément à Op_i: SemWait($Ordre.SemOp_{i,1}$),
4. Exécution de l'opération relative à Op_{i-1},
5. Accès et insertion de l'emplacement *Event* dans $LdOp_{i,j}$,
6. Signale l'insertion de l'emplacement *Event* à Op_i: SemWait($SemOp_{i,j}$).

Op_i : séquence exécutée par une tâche j, $j \in \{0, \ldots, PrOp_i - 1\}$

1. Récupération de l'ordre de traitement des emplacements par Op_{i-1} : récupération de la référence de la liste de disponibilité $LdOp_{i,k}$ où $k \in \{0, \ldots, PrOp_{i-1}\}$
   (a) Demande d'un élément de liste: SemWait($Ordre.SemOp_{i,1}$),
   (b) Accès et récupération de l'élément dans $Ordre.LdOp_{i,1}$,
   (c) Récupération de la référence de la liste de disponibilité à consulter $LdOp_{i,k}$,
   (d) Accès et insertion de l'élément dans $Ordre.LdOp_{i,0}$,
   (e) Signale l'insertion de l'élément à Op_{i-1}: SemWait($Ordre.SemOp_{i,0}$),
2. Demande d'emplacement *Event*: SemWait($SemOp_{i,k}$),
3. Accès et récupération de l'emplacement *Event* dans $LdOp_{i,k}$,
4. si type (Op_i) = Sync alors mémorisation de l'ordre de traitement des emplacements *Event* par Op_i,
5. Exécution de l'opération relative à Op_i,
6. Accès et insertion de l'emplacement *Event* dans $LdOp_{i+1,j}$,
7. Signale l'insertion de l'emplacement *Event* à Op_{i+1} : SemWait($SemOp_{i+1,j}$).

C.7 Mise en œuvre

Il convient de noter que le logiciel comprend 2 niveaux de définition utilisables selon le contexte et les besoins. On peut en effet définir un cycle opératoire:

- à partir des opérateurs, plus précisément leurs gestionnaires, par l'intermédiaire des structures de type **Mbm_TypeOp** et **Mbm_CycleOp**,

- à partir des tâches associées aux opérateurs par l'intermédiaire des structures de type **Mbm_ProcOp** et **Mbm_ProcCycle.**

Ceci permet d'utiliser ce logiciel de manière relativement souple, notamment lorsque les tâches applicatives sont déjà allouées et activées à l'extérieur ou bien lorsque la même tâche est incluse dans deux opérateurs relatifs à deux cycles opératoires distincts.

C.7.1 Définition des différentes structures nécessaires à la mise en œuvre du gestionnaire de flux

Le fichier **T_mbm.h** contient la déclaration des différentes structures de données nécessaires à la construction d'un gestionnaire de flux de données. Un cycle opératoire est défini par la structure **Mbm_CycleOp** comprenant les champs:

- *Space*: une zone mémoire–tampon où transistent les données,
- *SizeCycleO*: le nombre d'opérateurs travaillant autour de la zone mémoire–tampon,
- *LtTypeOp*: une liste d'opérateurs de type *Mbm_TypeOp* telle que size(*LtTypeOp*) = SizeCycleOp.

Un opérateur est défini par la structure **Mbm_TypeOp** comprenant les champs:

- *NbProc*: nombre de tâches qui lui sont associées,
- *Type*: son type *Sync* ou *Async*,
- *Op*: le gestionnaire associé à l'opérateur selon son type:
  - *Op.Async* de type *Mbm_Op* si l'opérateur est asynchrone,
  - *Op.Sync* de type *Mbm_OpS* si l'opérateur est synchrone.

Le gestionnaire relatif à un opérateur asynchrone est défini par la structure **Mbm_Op** comprenant les champs:

- *Ld*: une liste de disponibilité,
- *SemOp*: un sémaphore de synchronisation indiquant le nombre d'éléments présents dans la liste de disponibilité.

Le gestionnaire relatif à un opérateur synchrone est défini par la structure **Mbm_OpS** comprenant les champs:

- *Op*: une liste de structures de type *Mbm_Op* telle que size(*Op*) = nombre de tâches de l'opérateur précédent,
- *Ordre*: un cycle opératoire asynchrone de taille 2 permettant de respecter l'ordre d'entrée des événements de l'opérateur précédent.

Une liste de disponibilité est une liste chaînée "*Fisrt In First Out*" gérée par deux pointeurs, un pointeur de début et un pointeur de fin de liste. L'insertion des éléments s'effectue en fin de liste, la suppression en tête. Une liste de disponibilité est définie par la structure **Mbm_Ld** comprenant les champs:

- *InCount*: un compteur du nombre absolu d'éléments insérés dans la liste,
- *OutCount*: un compteur du nombre absolu d'éléments supprimés dans la liste,
- *SemLd*: un sémaphore autorisant l'accès à la liste à une seule tâche,

- *Fptr*: un pointeur d'accès au premier élément de la liste, pointeur de tête,
- *Bptr*: un pointeur d'accès au dernier élément de la liste, pointeur de fin.

Un élément d'une liste chaînée est défini par la structure **Mbm_list** comprenant les champs:

- *Argument*: correspondant à l'information à mémoriser dans la liste,
- *Next*: pointeur sur le prochain élément de la liste chaînée.

C.7.2 Initialisation d'un cycle opératoire

Deux fonctions permettent l'initialisation d'un cycle opératoire:

1. **Mbm_InitCycleOp** ayant pour paramètres:
   - *int nb_op*: nombre d'opérateurs dans le cycle,
   - *int élément*: nombre d'emplacements dans la zone mémoire–tampon,
   - *size_t size*: taille d'un emplacement,
   - ...: une énumération de la forme:
     - $PrOp_0$,
     - *type(* Op_0 *)*,
     - ⋮
     - $PrOp_{n-1}$,
     - *type(* Op_{n-1} *)*.

   alloue une zone mémoire de taille *element * size*, la partage en *element* emplacements de taille *size* puis alloue et initialise l'ensemble des gestionnaires nécessaires aux différents opérateurs,
2. **_Mbm_InitCycleOp** ayant pour paramètres:
   - *void *space*: pointeur sur la zone mémoire–tampon,
   - *int nb_op*: nombre d'opérateurs dans le cycle,
   - *int element*: nombre d'emplacements dans la zone mémoire–tampon,
   - *size_t size*: taille d'un emplacement,
   - ...: une énumération de la forme:
     - $PrOp_0$,
     - *type(* Op_0 *)*,
     - ⋮
     - $PrOp_{n-1}$,
     - *type(* Op_{n-1} *)*.

   partage la zone mémoire accesible par le pointeur *space* en *element* emplacement de taille *size* sans l'initialiser puis alloue et initialise l'ensemble des gestionnaires nécessaires aux différents opérateurs.

Elles retournent chacune un pointeur sur une structure *Mbm_CycleOp* si l'allocation est correcte, sinon un pointeur *NULL*. La figure C.5 donne l'état du gestionnaire de flux de données après l'exécution d'une des fonctions d'initialisation. Deux fonctions libèrent l'espace alloué pour un cycle opératoire:

1. **Mbm_FreeCycleOp** ayant pour paramètre un pointeur sur une structure *Mbm_CycleOp*.
2. **_Mbm_FreeCycleOp** ayant pour paramètre un pointeur sur une structure *Mbm_CycleOp*,

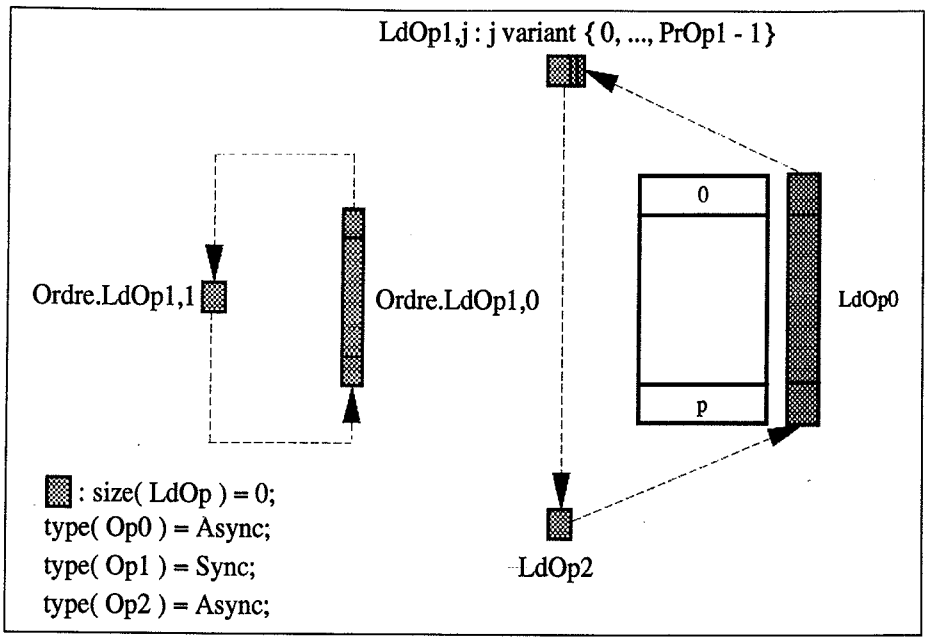


FIG. C.5 - *Etat à l'initialisation d'un gestionnaire de flux de données comprenant 3 opérateurs dont un synchrone.*

C.7.3 Initialisation de l'ensemble des tâches associées à un cycle opératoire

Les tâches associées à un opérateur Op_i sont définies par la structure **Mbm_ProcOp** comprenant les champs:

- *Priority*: leur priorité d'exécution: *PROC_HIGH* ou *PROC_LOW*,
- *Function*: l'opération relative à l'opérateur selon le cas,
  - *Function.A2A*: si type(Op_i) = Async et type(Op_{i+1}) = Async,
  - *Function.A2S*: si type(Op_i) = Async et type(Op_{i+1}) = Sync,
  - *Function.S2S*: si type(Op_i) = Sync et type(Op_{i+1}) = Sync,
  - *Function.S2A*: si type(Op_i) = Sync et type(Op_{i+1}) = Async,
- *NbParam*: le nombre de lots de paramètres associés à l'opérateur,
  - *0* si les tâches opèrent avec le même lot de paramètres d'entrée,
  - $PrOp_i$ si chaque tâche opère avec un lot de paramètres d'entrée distinct
- *Param*: les paramétres relatifs à la fonction:
  - structure contenant les paramètres si $NbParam = 0$,
  - tableau de structures contenant les différents paramètres si $NbParam = PrOp_i$.

Une fois définies les tâches associées à chaque opérateur, il convient de les associer au sein du cycle opératoire. Les tâches associées à un même opérateur sont activées par une tâche *mère*. L'ensemble des tâches associées à un cycle opératoire sont définies par la structure **Mbm_ProcCycle** comprenant les champs:

- *SizeCycleOp*: la taille du cycle opératoire,

- *ProcCycle*: une liste de tâches *mère* de taille **SizeCycleOp**. Chaque tâche *mère* est associée à un opérateur et permet d'activer l'ensemble des tâches opérant pour ce dernier selon leur priorité,
- *ProcOp*: une liste de structures de type *Mbm_ProOp* de taille **SizeCycleOp** qui définissent les tâches associées aux différents opérateurs.

La fonction **Mbm_ProCyleAlloc** permet l'initialisation de l'ensemble des tâches relatives à un cycle opératoire et la construction du cycle opératoire même. Elle effectue l'interconnection entre les différents opérateurs et l'affectation des paramétres aux fonctions associées aux opérateurs. Ces paramètres sont:

- *Mbm_CycleOp cycleop*: pointeur sur un cycle opératoire,
- ...: l'énumération des tâches associées aux opérateurs de la forme:

  ⋮
  - la priorité de Op_i: *PROC_HIGH* ou *PROC_LOW*,
  - le pointeur de la fonction relative à Op_i,
  - la taille de l'espace de travail exprimée en octets ($0 \equiv 1024 octets$),
  - le nombre de paramètres associés à Op_i:
    - *0* si les tâches ont les mêmes paramètres d'entrée,
    - $PrOp_i$ si chaque tâche a des paramètres d'entrée différents,
  - le pointeur sur les paramètres de la fonction:
    - structure contenant les paramètres si $NbParam = 0$,
    - tableau de structures contenant les différents paramètres si $NbParam = PrOp_i$.

  ⋮

Elle retourne un pointeur sur une structure *Mbm_ProcCycle* si l'initialisation est correcte, sinon un pointeur *NULL*.
La fonction **Mbm_ProcCycleAllocClean** libère l'espace alloué par la fonction **Mbm_ProcCycleAlloc**. Elle a pour paramètre d'entrée un pointeur sur une structure *Mbm_ProcCycle*.
La fonction **Mbm_Run** active les différentes tâches du cycle opératoire selon la priorité de l'opérateur. Elle a pour paramètre d'entrée un pointeur sur une structure *Mbm_ProcCycle*.

C.7.4 Les différents types de fonctions d'un cycle opératoire

On distingue 4 types de fonctions relatives aux 4 possibilités de transitions entre opérateurs *synchrones* et *asynchrones*. On définit j, $j \in \{0, \ldots, PrOp_i - 1\}$, indice de la tâche exécutant la fonction relative à Op_i. Les 4 transitions entre opérateurs sont:

1. la transition *Async-Async* pour laquelle on définit le type de fonction **void (*Mbm_A2AFunction)** ayant pour paramètres:
   - *Process *p*: pointeur de tâche,
   - *Mbm_Op *op_in*: opérateur d'entrée d'emplacement *Event*: Op_i,
   - *Mbm_Op *op_out*: opérateur de sortie d'emplacement *Event*: Op_{i+1},
   - *void *param*: pointeur sur une structure contenant les autres paramètres de la fonction.

2. la transition *Async–Sync* pour laquelle on définit le type de fonction **void (*Mbm_A2SFunction)** ayant pour paramètres:
   - *Process *p*: pointeur de tâche,
   - *Mbm_Op *op_in*: opérateur d'entrée d'emplacement *Event*: Op_i,
   - *Mbm_Op *op_out*: opérateur de sortie d'emplacement *Event*: $Op_{i+1,j}$,
   - *Mbm_Op *op_2sin*: opérateur d'entrée pour la mémorisation de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i+1,0}$,

     *Mbm_Op *op_2sout*: opérateur de sortie pour la mémorisation de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i+1,1}$,
   - *void *param*: pointeur sur une structure contenant les autres paramètres de la fonction.
3. la transition *Sync–Sync* pour laquelle on définit le type de fonction **void (*Mbm_S2SFunction)** ayant pour paramètres:
   - *Process *p*: pointeur de tâche,
   - *Mbm_Op *op_out*: opérateur de sortie d'emplacement *Event*: $Op_{i+1,j}$,
   - *Mbm_Op *op_fsin*: opérateur d'entrée pour la récupération de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i,1}$,
   - *Mbm_Op *op_fsout*: opérateur de sortie pour la récupération de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i,0}$,
   - *Mbm_Op *op_2sin*: opérateur d'entrée pour la mémorisation de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i+1,0}$,
   - *Mbm_Op *op_2sout*: opérateur de sortie pour la mémorisation de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i+1,1}$,
   - *void *param*: pointeur sur une structure contenant les autres paramètres de la fonction.
4. la transition *Sync–Async* pour laquelle on définit le type de fonction **void (*Mbm_S2AFunction)** ayant pour paramètres:
   - *Process *p*: pointeur de tâche,
   - *Mbm_Op *op_out*: opérateur de sortie d'emplacement *Event*: Op_{i+1},
   - *Mbm_Op *op_fsin*: opérateur d'entrée pour la récupération de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i,1}$,
   - *Mbm_Op *op_fsout*: opérateur de sortie pour la récupération de l'ordre dans le cycle opératoire **Ordre**: $Ordre.Op_{i,0}$,
   - *void *param*: pointeur sur une structure contenant les autres paramètres de la fonction.

C.7.5 Mise en œuvre d'une fonction associée à un opérateur

Le déroulement d'une séquence de code relatif à une fonction d'un cycle opératoire est le suivant:

1. Récupération d'un emplacement *Event* par Op_i,
2. Exécution de l'opération relative à Op_i,
3. Libération de l'emplacement pour Op_{i+1},

Pour chaque type de transition, on définit une fonction qui permet de récupérer un emplacement. Ces fonctions sont

- pour une gestion de flux *bloquante* contenant des opérateurs *asynchrones et synchrones*:

  Mbm_GetDataEvent (*Mbm_Op* *op_in) pour une transistion *Async–Async*,

  Mbm_GetDataEventA2S (*Mbm_Op* *op_in, *Mbm_Op* *op_out, *Mbm_Op* *op_2sin, *Mbm_Op* *op_2sout) pour une transition *Async–Sync*,

  Mbm_GetDataEventS2S (*Mbm_Op* *op_out, *Mbm_Op* *op_fsin, *Mbm_Op* *op_fsout, *Mbm_Op* *op_2sin, *Mbm_Op* *op_2sout) pour une transition *Sync–Sync*,

  Mbm_GetDataEventS2A (*Mbm_Op* *op_out, *Mbm_Op* *op_fsin, *Mbm_Op* *op_fsout) pour une transition *Sync–Async*.

- pour une gestion de flux *non–bloquante* contenant des opérateurs *asynchrones*:

  Mbm_GetSurveyEvent (*Mbm_Op* *op_in, *Mbm_Op* *op_out)

La libération d'un emplacement est effectuée avec la fonction:

Mbm_FreeEvent (*Mbm_Op* *op_out) quel que soit le type de gestion de flux de données.

C.7.6 Exemple de mise en œuvre d'un gestionnaire de flux de données

L'exemple ci–dessous donne le code relatif à un gestionnaire de flux comprenant 2 opérateurs *asynchrones* et 2 opérateurs *synchrones*.

```
# include ≺ stdlib.h ≻
# include ≺ misc.h ≻
# include ≺ T_mbm.h ≻

void FuncOp0 2Op1( Process *p, Mbm_Op *op_in, Mbm_Op *op_out, Mbm_Op *op_2sin, Mbm_Op
*op_2sout, void *param )
{
    Mbm_list *tmp;
    void *event;

    p = p;
    while(1)
    {
        tmp = Mbm_GetDataEventA2S( op_in, op_2sin, op_2sout );
        event = tmp→Argument;
        /* code relatif à l'opération de Op0 */
        Mbm_FreeEvent( op_out, tmp );
    }
}

void FuncOp1 2Op2( Process *p, Mbm_Op *op_out, Mbm_Op *op_fsin, Mbm_Op *op_fsout, Mbm_Op
*op_2sin, Mbm_Op *op_2sout, void *param )
{
    Mbm_list *tmp;
    void *event;

    p = p;
    while(1)
    {
        tmp = Mbm_GetDataEventS2S( op_fsin, op_fsout, op_2sin, op_2sout );
        event = tmp→Argument;
        /* code relatif à l'opération de Op1 */
        Mbm_FreeEvent( op_out, tmp );
    }
}

void FuncOp2 2Op3( Process *p, Mbm_Op *op_out, Mbm_Op *op_fsin, Mbm_Op *op_fsout, void
*param )
{
    Mbm_list *tmp;
```

```
        *event;

        p = p;
        while(1)
        {
            tmp = Mbm_GetDataEventS2A( op_fsin, op_fsout );
            event = tmp→Argument;
            /* code relatif à l'opération de Op2 */
            Mbm_FreeEvent( op_out, tmp );
        }
}

void FuncOp3_2Op0( Process *p, Mbm_Op *op_in, Mbm_Op *op_out, void *param )
{
        Mbm_list *tmp;
        void *event;

        p = p;
        while(1)
        {
            tmp = Mbm_GetDataEvent( op_in );
            event = tmp→Argument;
            /* code relatif à l'opération de Op3 */
            Mbm_FreeEvent( op_out, tmp );
        }
}

main( void )
{
        Mbm_CycleOp *cycleop;
        Mbm_ProcCycle *proccycle;
        void **paramOp0;
        void *paramOp1;
        void *paramOp2;
        void **paramOp3;

        cycleop = Mbm_InitCycleOp
        (
            n, p, size_p,
            PrOp0, Async,
            PrOp1, Sync,
            PrOp2, Sync,
            PrOp3, Async
        );
```

```
    proccycle = Mbm_ProcCycleAlloc
    (
        cycleop,
        PROC_LOW, FuncOp0 2Op1, 0, PrOp0, paramOp0,
        PROC_HIGH, FuncOp1 2Op2, 0, 0, paramOp1,
        PROC_LOW, FuncOp2 2Op3, 0, 0, paramOp2
        PROC_LOW, FuncOp3 2Op0, 0, PrOp3, paramOp3
    );
    Mbm_Run( proccycle );
    Mbm_ProcCycleAllocClean( proccycle );
    Mbm_FreeCycleOp( cycleop );
    exit( EXIT_SUCCESS );
}
```

Annexe D

Description de l'assemblage

Le format de l'événement, décrit dans [6], est construit à partir de pointeurs relatifs à chaque sous–détecteurs nonmés $Bloc_{Entite}$. $Partition_{Tmb}$ correspond à l'ordre suivant lequel les partitions doivent être émises sur le site Tmb. $Partition_{Entite}$ correspond à l'ordre suivant lequel les partitions doivent être reçues et rangées sur une unité.

Tmb	$Bloc_{Tmb}$	Taille Port (octets)	$Partition_{Tmb}$	Taille Partition (octets)	$Partition_{Entite}$	$Bloc_{Entite}$
Tmb_0	$Port_0$	8	0	8	0	Header
	$Port_1$	64	1	64	1	Luminosity
	$Port_2$	128	2	64	2	BGO
			5	64	16	BGO
	$Port_3$	192	3	64	3	BGO
			4	64	15	BGO
			6	32	28	HCalB
			7	32	40	HCalB
Tmb_1	$Port_0$	256	0	64	4	BGO
			7	64	14	BGO
			8	32	17	HCalA
			15	32	27	HCalA
			16	32	29	HCalB
			23	32	39	HCalB
	$Port_1$	256	1	64	5	BGO
			6	64	13	BGO
			9	32	18	HCalA
			14	32	26	HCalA
			17	32	30	HCalB
			22	32	38	HCalB
	$Port_2$	256	2	64	6	BGO
			5	64	12	BGO
			10	32	19	HCalA
			13	32	25	HCalA
			18	32	31	HCalB
			21	32	37	HCalB
	$Port_3$	256	3	64	7	BGO
			4	64	11	BGO
			11	32	20	HCalA
			12	32	24	HCalA
			19	32	32	HCalB
			20	32	36	HCalB
Tmb_2	$Port_0$	256	0	64	8	BGO
			2	64	10	BGO
			3	32	21	HCalA
			5	32	23	HCalA
			6	32	33	HCalB
			8	32	35	HCalB
	$Port_1$	192	1	128	9	BGO
			4	32	22	HCalA
			7	32	34	HCalB
	$Port_2$	116	9	116	41	EnergyLvl11
	$Port_3$	320	10	320	42	EnergyLvl12

TAB. D.1 - *Description de l'assemblage d'un événement pour le déclenchement de niveau–2*

Tmb	$Block_{Tmb}$	Taille Port (octets)	$Partition_{Tmb}$	Taille Partition (octets)	$Partition_{Entite}$	$Block_{Entite}$
Tmb_3	$Port_0$	156	0	128	43	EnergyLvl0
			1	28	44	EnergyLvl13
	$Port_1$	128	2	128	45	Scintillator
	$Port_2$	140	3	140	46	MuonFdBd
Tmb_4	$Port_0$	0...512	0	0...512	47	MuonRPHI1
	$Port_1$	0...512	1	0...512	48	MuonRPHI2
	$Port_2$	0...512	2	0...512	49	MuonRPHI3
	$Port_3$	0...512	3	0...512	50	MuonRPHI4
Tmb_5	$Port_0$	0...512	0	0...512	51	MuonRPHI5
	$Port_1$	0...512	1	0...512	52	MuonRPHI6
	$Port_2$	0...512	2	0...512	53	MuonRPHI7
	$Port_3$	0...512	3	0...512	54	MuonRPHI7
Tmb_6	$Port_0$	0...512	0	0...512	55	MuonRPZ1
	$Port_1$	0...512	1	0...512	56	MuonRPZ2
	$Port_2$	0...512	2	0...512	57	MuonRPZ3
	$Port_3$	0...512	3	0...512	58	MuonRPZ4
Tmb_7	$Port_0$	0...512	0	0...512	59	MuonRPZ5
	$Port_1$	0...512	1	0...512	60	MuonRPZ6
	$Port_2$	0...512	2	0...512	61	MuonRPZ7
	$Port_3$	0...512	3	0...512	62	MuonRPZ8
Tmb_8	$Port_0$	4	0	4	63	MuonLvl11
	$Port_1$	4	1	4	64	MuonLvl12
	$Port_2$	4	2	4	65	MuonLvl13
	$Port_3$	4	3	4	66	MuonLvl14
Tmb_9	$Port_0$	384	0	384	67	TecRPHI1
	$Port_1$	384	1	384	68	TecRPHI2
	$Port_2$	400	2	400	69	TecRPHI3
Tmb_{10}	$Port_0$	0...512	0	0...512	70	TecRZH1
	$Port_1$	0...512	1	0...512	71	TecRZH2
	$Port_2$	0...512	2	0...512	72	TecRZH3
Tmb_{11}	$Port_0$	0...512	0	0...512	73	TecRZL1
	$Port_1$	0...512	1	0...512	74	TecRZL2
	$Port_2$	0...512	2	0...512	75	TecRZL3

TAB. D.2 - *Description de l'assemblage d'un événement pour le déclenchement de niveau-2 (suite et fin)*

Bibliographie

[1] Ed Barsotti, Alexander Booth, and Mark Bowden. Effects of various event building techniques on data acquisition system architectures. In *Computing For High Luminosity and High Intensity Facilities, Santa Fe*, 1990.

[2] S.P. Beingessner, J.J. Blaising, F. Chollet–Le Flour, A. Degré, G. Dromby, G. Forconi, C. Goy, J. Lecoq, R. Morand, M. Moynot, G. Perrot, and S. Rosier–Lees. The second level trigger of the l3 experiment, part 2. the event selection. *Nuclear Instruments and Methods in Physics Research, Section A 340(1994) 322–327*, 1994.

[3] J.C. Bermond and M. Syska. Routage *wormhole* et canaux virtuels. In M.Cosnard, M.Nivat, and Y.Robert, editors, *Algorithmique Parallèle*, chapter 10. Masson, 1992.

[4] Y. Bertsch, J.J. Blaising, H. Bonnefon, F. Chollet–Le Flour, A. Degré, G. Dromby, J. Lecoq, R. Morand, M. Moynot, G. Perrot, and X. Riccadonna. The second level trigger of the l3 experiment, part 1. the implemtation. *Nuclear Instruments and Methods in Physics Research, Section A 340(1994) 309–321*, 1994.

[5] B. Bertucci, S. Falciano, G Medici, and D. Linnhöfer. The l3 fastbus data acquisition system. In *Computing For High Luminosity and High Intensity Facilities, Santa Fe*, 1990.

[6] J.J. Blaising and A. Degre. Data format of the second level trigger. Laboratoire d'Annecy–Le–Vieux de Physique des Particules, May 1995.

[7] Xudong Cai. *Contribution à l'élaboration et à la mise en œuvre du système de déclenchement et d'acquisition de l'expériemce L3 au LEP.* PhD thesis, Université de Savoie, Laboratoire d'Annecy–Le–Vieux de Physique des Particules, BP 110, 74941 Annecy–Le–Vieux Cedex France, 1994.

[8] Frédérique Chollet. *Contribution à l'Elaboration et à la mise au Point du Système de Déclenchement Microprogrammé de Deuxième Niveau de l'expérience LEP–3.* PhD thesis, Université de Savoie, Laboratoire d'Annecy–Le–Vieux de Physique des Particules, BP 110, 74941 Annecy–Le–Vieux Cedex France, 1988.

[9] Frédéric Desprez, Eric Fleury, and Michel Loi. T9000 et c104, la nouvelle génération de transputers. Technical Report 93–01, Laboratoire de l'Informatique du Parallélisme, Février 1993.

[10] Stuart Fisher. Low–level benchmarcking of the *t9000 transputer*. Technical report, CERN / University of Liverpool, March 1995.

[11] Eric Fleury and Pierre Fraigniaud. Deadlocks in adaptive wormhole routing. Technical Report 94–09, Laboratoire de l'Informatique du Parallélisme, March 1994.

[12] Pierre Fraigniaud. Communications dans un réseau de processeurs. In M.Cosnard, M.Nivat, and Y.Robert, editors, *Algorithmique Parallèle*, chapter 9. Masson, 1992.

[13] Pierre Fraigniaud and Cyril Gavoille. Interval routing schemes. Technical Report 94–04, Laboratoire de l'Informatique du Parallélisme, January 1994.

[14] M. Griffiths and M. Vayssade. *Architecture des systèmes d'exploitation.* Traité des Nouvelle Technologies, série informatique,HERMES, 1990.

[15] Stefan Haas, Xinjian Liu, and Brian Martin. Long distance differential transmission of *ds links* copper cable. Technical report, CERN/ECP–GPMIMD 93, CERN–European Organization For Nuclear Research, ECP Division, CH-1211 Geneve 23, June 1993.

[16] Charles Antony Richard Hoare. *Communicating sequential processes.* Englewood Cliffs, N.J.: Prentice-Hall, 1985.

[17] Inmos. *The Transputer Databook.*, second edition, 1989.

[18] Inmos. *T9000 Transputer Hardware Reference Manual.*, first edition, 1993.

[19] Inmos. *T9000 Transputer Instruction Set Manual.*, first edition, 1993.

[20] Inmos. *PRQ T9000 Transputer.*, August 1994.

[21] Inmos. *T9000 ANSI C Toolset Language and Libraries Reference Manual.*, November 1994.

[22] Inmos. *T9000 Hardware Configuration Manual.*, November 1994.

[23] L3 Collaboration. Letter of intent. *European Organization for Nuclear Research (*CERN*), CERN/LEPC 82–5*, March 1982.

[24] L3 Collaboration. Trigger and data acquisition sysytem of l3. *European Organization for Nuclear Research (*CERN*), CERN/LEPC 84–5*, January 1984.

[25] L3 Collaboration. Hadron calorimetry in the l3 detector. *Nuclear Instruments and Methods in Physics Research, Section A 302 53–62*, 1991.

[26] L3 Collaboration. The l3 scintillation counter system. *L3 Note 1400*, March 1991.

[27] L3 Collaboration, A.Arefrev and al. . Proportionnal chambers for the hadron calirimeter of the l3 experiment. *Nuclear Instruments and Methods in Physics Research, Section A 275 71–80*, 1989.

[28] L3 Collaboration, B.Adeva. and al. Muon detector in the l3 experiment. *Nuclear Instruments and Methods in Physics Research, Section A 277 187–193*, 1989.

[29] L3 Collaboration, B.Adeva. and al. The construction of the l3 experiment. *Nuclear Instruments and Methods in Physics Research, Section A 289 35–102*, 1990.

[30] L3 Collaboration, F.Ferroni and al. The l3 bgo electromagnetic calorimeter at lep. *Nuclear Instruments and Methods in Physics Research, Section B 23A 100–106*, 1991.

[31] LEP Collaboration. Lep design report. Technical report, CERN–LEP/Th/83-29, 1983.

[32] B. Martin, X.D. Cai, S. Falciano, C. Luci, L. Luminari, G. Marzano, and G. Medici. Ft800: A fastbus to transputer interface. In *Conference Record of the EighthConference on Real–Time Computer Applications in Nuclear, Particle and Plasma Physics, Vancouver*, June 8–11 1993.

[33] M.D. May, P.W. Thompson, and P.H. Welch. *Networks, Routers and Transputers: function, performance, and application.*, 1993.

[34] H. Muller. The cern host interface. *IEEE Trans. Nucl. Sci, NS 37(1990) 361*, 1990.

[35] NEBULAS Collaboration. Rd–31 status report, nebulas: A high performance data–driven event building architecture based on an asynchronous self–routing packet switching network. Rd–31 status report, CERN/DRDC/93–55, December 1993.

[36] NEBULAS Collaboration. Rd–31 status report95, nebulas: A high performance data–driven event building architecture based on an asynchronous self–routing packet switching network. Lcrb status report/rd–31, CERN/LHCC/95–14, March 1995.

[37] D.N. Ren. *The L3 Vertex Chamber Development and Infrastructure.* PhD thesis, Swiss Federal Institute of Technology *ETH*, Zurich, 1990.

[38] Alexandre Ungerer. *Mèthodologie pour la Parallèlisation de Programmes Scientifiques. Conception d'une Carte à Processeur Parallèle.* PhD thesis, Université de Savoie, Laboratoire d'Annecy–Le–Vieux de Physique des Particules, BP 110, 74941 Annecy–Le–Vieux Cedex France, 1991.