50376
1986
151

N° d'ordre : 75

ANNEE 1986

50376
1986
151

LIFL

U.A. 369 du C.N.R.S.

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

THÈSE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

FLANDRES ARTOIS

pour obtenir

Le titre de DOCTEUR

en Informatique

ARCHITECTURES ET ALGORITHMES PARALLÈLES POUR LES MÉTHODES MONTÈ CARLO EN PHYSIQUE DES PARTICULES

par

Jean – Luc DEKEYSER

BIBLIOTHEQUE INTERUNIVERSITAIRE
SECTION DE SCIENCES
LILLE

UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS
U.F.R. d'I.E.E.A. Bât. M3 - 59655 VILLENEUVE D'ASCQ CEDEX
Tél. 20 43 44 93

BIBLIOTHÈQUE INTERUNIVERSITAIRE
SECTION DE SCIENCES
LILLE

Thèse soutenue le 20 octobre 1986, devant la commision d'examen

Membres du jury

M.DAUCHET	Président
V.CORDONNIER	Rapporteur – Examinateur
G.PLATEAU	Rapporteur – Examinateur
D.LEVINTHAL	Rapporteur
R.BRUN	Examinateur
C.CARREZ	Examinateur

Remerciements

Je tiens à remercier

Monsieur Max Dauchet, professeur à l'université de Lille 1, qui m'a fait l'honneur de présider le jury de ma thèse, et à qui j'exprime toute ma sympathie.

Monsieur René Brun, responsable du projet GEANT3 au CERN, qui est à l'origine de ces travaux et n'a cessé d'y participer, et qui a aussi accepté d'examiner ce travail.

Monsieur Christian Carrez, professeur à l'université de Lille 1, qui a accepté de participer au jury de ma thèse, avec lequel auparavant j'ai pu apprendre l'essentiel des connaissances nécessaires pour la réalisation de ce travail.

Monsieur Vincent Cordonnier, professeur à l'université de Lille 1, qui participa à ce jury après avoir produit son rapport et qui lors de mes études a occupé une place importante.

Monsieur David Levinthal, professeur à l'université de Floride, qui a bien voulu écrire un rapport sur cette thèse, mais qui aussi m'a cordialement reçu au sein de son laboratoire pour la réalisation des travaux présentés dans le troisième chapitre.

Monsieur Gérard Plateau, professeur à l'université de Paris Nord, qui a accepté d'examiner ce travail après en avoir fait un rapport.

Monsieur Paolo Zanella, directeur du laboratoire informatique du CERN, qui m'a acueilli au sein de son équipe à fin de réaliser ce travail.

Messieurs A.Fucci et M.Storr ainsi que les membres de l'équipe chargée du développement des émulateurs au CERN, pour leur disponibilité et leurs conseils.

Messieurs J.Lannutti et C.Georgiopoulos ainsi que les chercheurs de l'université de Floride pour leurs conseils avisés en matière de vectorisation.

Messieurs V.Frigo et M.Goossens pour leurs nombreuses discussions autour d'Ada.

Mademoiselle S.Biziaux qui a accepté de sacrifier plusieurs heures de loisirs à la réalisation des figures de cette thèse sur Mac−Intosh.

Enfin, je tiens à exprimer toute ma sympathie aux amis que j'ai pu rencontrer durant mes études à Lille.

TABLE DES MATIÈRES

Remerciements iii

Part 1: Introduction 1

Part 2: Développement 3

1. ASPECT SEQUENTIEL DU SIMULATEUR GEANT 3 3
1.1 Introduction à GEANT3 3
1.1.1 Applications 3
1.1.2 Décomposition séquentielle de GEANT3 6
1.2 Structure de données de GEANT3 8
1.2.1 Présentation de ZEBRA 8
1.2.2 Structures de données statiques 9
1.2.2.1 Les matériaux 9
1.2.2.2 Les paramètres caractérisant le milieu 10
1.2.2.3 Les particules 11
1.2.2.4 La géométrie 12
1.2.2.5 Matrices de rotation 16
1.2.2.6 Les détecteurs sensitifs 16
1.2.3 Structures de données dynamiques 17
1.2.3.1 Les particules générées 17
1.2.3.2 Les trajectoires de particules 18
1.2.3.3 Réponses des détecteurs 18
1.3 Algorithme de simulation 19
1.3.1 Simulation d'un évènement 21
1.3.2 Progression d'une particule 22
1.3.3 Algorithme du suivi d'une particule dans un volume 24
1.3.4 Interface avec la géométrie 25
1.4 Machines séquentielles à grande mémoire 27
2. DEVELOPPEMENT SUR UNE MACHINE MULTIPLE 31
2.1 Particularités de GEANT3 liées aux Multi−Calculateurs 31
2.1.1 Initialisation des structures de données statiques 31
2.1.2 Restructuration des résultats 32
2.1.3 Accès à un fichier commun 32
2.1.4 Conséquence pour l'architecture 32
2.1.5 Génération de nombres aléatoires 32
2.1.5.1 Définition 33
2.1.5.2 Exemple d'application 35
2.2 Multi−Calculateurs de type maître−esclaves 35
2.2.1 Présentation de l'architecture 35
2.2.2 Parallélisation de type multi−évènements 37
2.3 Développement sur une batterie d'émulateurs 3081/E 38

2.3.1 Présentation technique de la machine 38
2.3.2 Procédures de communication 40
2.3.3 Entrées/sorties. 41
2.3.3.1 Buffer en entrée. 41
2.3.3.2 Buffer en sortie. 42
2.3.3.3 Partage des fichiers d'entrées/sorties 42
2.3.4 Performances 46
2.3.5 Conclusion sur les machines Maître – Esclaves 48
2.4 GEANT3 en tant qu'algorithme à pile 48
2.4.1 Pile partagée, pile répartie 48
2.4.2 Algorithme à pile et architecture 52
2.4.2.1 Machine séquentielle 52
2.4.2.2 Architecture pour pile partagée ou répartie 53
2.4.3 Extension à plusieurs niveaux 54
2.5 Conclusion 55
3. DEVELOPPEMENT SUR UN CALCULATEUR VECTORIEL 57
3.1 Machine vectorielle 58
3.1.1 Concept de base 58
3.1.2 Machine SIMD pipe – line 58
3.1.3 Machine SIMD de type tableau 60
3.2 Vectorisation des procédures géométriques de GEANT3 61
3.2.1 Détection des algorithmes vectorisables 61
3.2.2 Construction des vecteurs en mémoire 63
3.2.3 Vectorisation de GNEXT 63
3.2.4 Vectorisation de GSEAR 67
3.2.5 Performances de cette vectorisation sur Cyber 205 70
3.3 Vectorisation des algorithmes généraux de simulation 73
3.3.1 Algorithme général vectorisé 74
3.3.2 Prévision des performances 75
3.4 FORTRAN 200 et la programmation vectorielle explicite 77
3.4.1 Présentation de FORTRAN 200 77
3.4.2 Codage d'une décomposition algorithmique conditionnelle. 81
3.5 Conclusion sur la vectorisation et les machines vectorielles 84
4. APPROCHE DE ADA POUR GEANT3 85
4.1 Les structures de données de GEANT3 en Ada 86
4.1.1 Notion de paquettage Ada 86
4.1.2 Un paquettage par structure 87
4.1.3 Un paquettage générique de gestion de structure de données 90
4.1.3.1 Définition de l'information 90
4.1.3.2 Construction structurelle du paquettage 91
4.1.3.3 Construction qualitative du paquettage 92
4.1.3.4 Les entrées/sorties 94
4.1.3.5 Réalisation en Ada 95
4.2 Multi – tâches et GEANT3 105
4.2.1 Génération arborescente de tâches 105

4.2.2 Multi−tâches et pipe−line 109
4.2.3 Synchronisation indirecte 112
4.3 Conclusion 115

Part 3: Conclusion 117

Appendice A: Vecteur de sémaphores 119

Appendice B: Utilisation des émulateurs 3081E 123

Appendice C: Vectorisation de la géométrie 156

Appendice D: Synchronisation indirecte en Ada 167

Références 197

FIGURES

1. Géométrie de OPAL : section en z=0 4
2. Développement d'un évènement dans OPAL 5
3. Algorithme de GEANT3 7
4. Structure arborescente binaire 9
5. Gestion interne des liens 10
6. Structure des matériaux : JMATE 11
7. Structure associée aux milieux : JTMED 12
8. Structure des particules : JPART 13
9. Exemple d'arbre géométrique 14
10. Arborescence de volumes 15
11. Structure de la géométrie : JVOLUM 16

12. Structure des matrices de rotation : JROT 17

13. Structure des détecteurs : JSET 18

14. Structure de file des particules : JKINE/JVERTX 19

15. Structure de pile de particules : JSTAK 20

16. Structure des trajectoires : JXYZ 21

17. Structure des chocs avec détecteurs : JHITS 22

18. Structure de digitisation : JDIGI 22

19. Algorithme de GRUN 23

20. Algorithme de GTRIG 23

21. Algorithme de GTREVE 24

22. Algorithme du suivi d'une particule : GTRACK 25

23. Algorithme de propagation dans un volume : GTVOL 26

24. Exemple où GPCHK est nécessaire 27

25. Gestion des paramètres modifiés 28

26. Machine maître – esclaves 36

27. Emulateurs 3081E 39

28. Architecture interne d'un 3081/E 40

29. Gestion du buffer de sortie 42

30. Pile partagée 49

31. Pile répartie 50

32. Gestion de la pile sur une machine séquentielle 52

33. Architecture à mémoire circulante 53

34. Pile répartie à plusieurs niveaux 55

35. Architecture à mémoire circulante à plusieurs niveaux 56

36. Evolution d'une addition en mode pipe – line 59

37. MFLOPS en fonction de la longueur du vecteur sur CY205 60

38. Répartition du temps CPU pour GOPAL, GEXAM1 (cf 3.2.5) 62

39. Utilisation de GNEXT (2 dimensions) 64

40. Partie de l'algorithme séquentiel de GNEXT 65

41. Vectorisation de la partie de GNEXT 66

42. Partie de l'algorithme séquentiel de GSEAR 68

43. Vectorisation d'une partie de GSEAR 69

44. Arborescence de la géométrie de GEXJLD 71

45. Tableau des gains pour la vectorisation de la géométrie 72

46. Critères de vectorisation de boucle sur Cyber 205 73

47. Comparaison du temps d'exécution pour 3 solutions de sélection 82

48. Structure des matériaux en Ada 87

49. Extraction des sous – arbres d'un graphe 92

50. Exemple d'utilisation de l'information structurelle 93

51. Exemple de graphe 100

52. Construction structurelle des matériaux de GEANT3 103

53. Génération arborescente de tâches 106

54. Partage du macro pipe – line de tâches 110

55. Réalisation matérielle pour MAS 120

56. Réalisation matérielle pour RES 121

Partie 1

Introduction

Les méthodes Montè Carlo en physique des particules représentent à l'heure actuelle 50 pour cent du temps CPU consommé au CERN (Centre Européen de Recherche Nucléaire), soit 10000 heures d'IBM 370/168. Des prévisions pour 1989 [23] annoncent un chiffre de 120000 heures. C'est pourquoi il devient très intéressant de chercher de nouvelles solutions matérielles et logicielles pour ce type d'algorithme.

Nous avons voulu, avec les informaticiens du CERN, tenter de développer sur différents types de machines un des algorithmes de simulation les plus utilisés (3 des 4 expériences LEP).

Cette présentation des travaux effectués comprend quatre parties.

Le chapitre 1 présente ce logiciel de simulation, il permet de se familiariser avec les structures de données et les procédures séquentielles qui les manipulent.

Le chapitre 2 propose une parallélisation de type multi – évènements, où chacun est simulé simultanément sur plusieurs processeurs. Un algorithme à pile partagée ou répartie est ensuite présenté, il peut s'appliquer aussi à d'autres algorithmes à pile.

Le chapitre 3 est consacré au développement d'algorithmes vectoriels et à leur application sur un Cyber 205.

Le chapitre 4 propose une approche de réalisation en Ada, avec en particulier l'utilisation des types d'objets et du système multi – tâches.

Partie 2

Développement

CHAPITRE 1
ASPECT SEQUENTIEL DU SIMULATEUR GEANT 3

La complexité des expériences en physique des particules augmentant sans cesse, la simulation devient (de plus en plus) primordiale afin de concevoir et d'optimiser les détecteurs ainsi que d'interpréter les résultats expérimentaux produits.

GEANT3 [11] a été développé au CERN dans cet objectif. Depuis 1974, où seules quelques particules pouvaient être suivies dans des détecteurs relativement simples, il n'a cessé d'être amélioré. Il est à l'heure actuelle, le seul logiciel généralisé de simulation utilisable pour plusieurs types d'expériences. Il est écrit en FORTRAN 77 et peut être développé sur la plupart des machines : IBM, VAX, Apollo ...

Lors de cet exposé, certaines références seront faites à OPAL, il s'agit de l'une des expériences en cours de mise au point pour le LEP et utilisant GEANT3. Son utilisation a permis de tester les différentes propositions faites sur un exemple concret et représentatif. La complexité de la géométrie et des mécanismes mis en jeu sont en fait assez semblables pour toutes les manipulations prévues pour le LEP.

1.1 Introduction à GEANT3

1.1.1 Applications

Les principales applications de GEANT3 sont :

- le suivi de particules à travers un environnement expérimental (fig 1) afin d'approuver les résultats théoriques, ou de simuler les réponses de détecteurs.

- la représentation graphique de l'environnement et des trajectoires de particules. (fig 2)

L'orientation de la simulation peut être à double sens : soit interactive (et ainsi peut – on suivre sur écran l'évolution d'une particule dans l'environnement et donc facilement corriger la géométrie et la direction initiale de la particule), soit productive, les règles de simulations ayant été choisies, il faut simuler un grand nombre d'évènements semblables utilisant des séries de nombres aléatoires

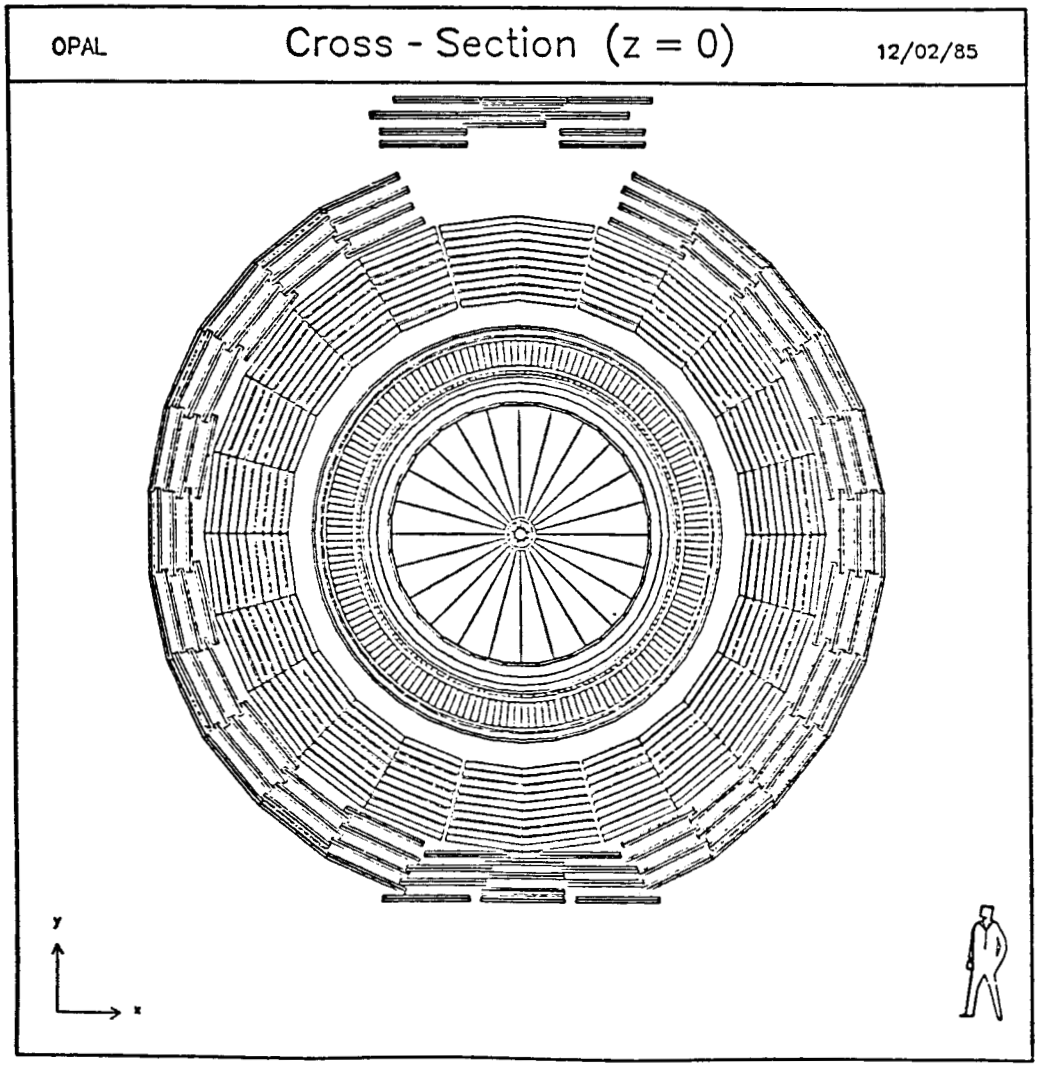


Figure 1: Géométrie de OPAL : section en z=0

différentes, afin de produire des statistiques représentatives. Chaque évènement est caractérisé par la donnée du type de particule initiale, de sa position dans l'environnement et de son énergie initiale.

Pour permettre ces différentes applications, GEANT3 propose :

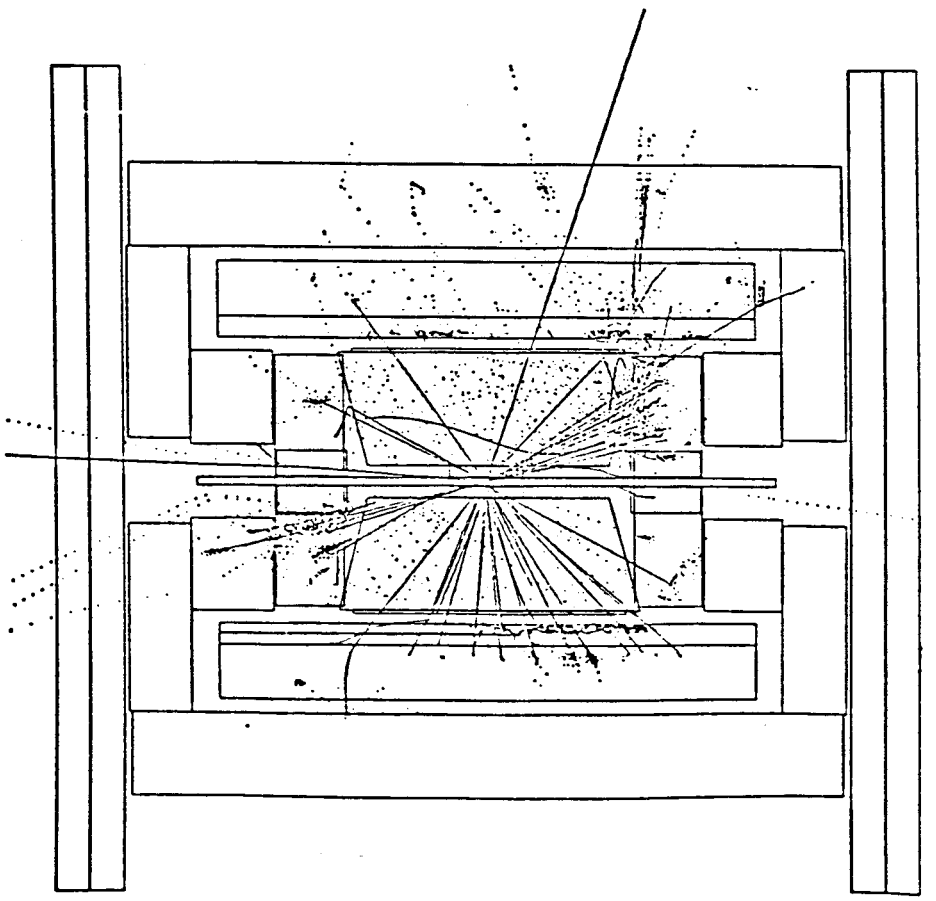


Figure 2: Développement d'un évènement dans OPAL

BU LILLE

- de décrire un environnement expérimental de façon explicite et rigoureuse. Il est représenté par une structure arborescente de volumes élémentaires. Chaque volume est associé à un ensemble de paramètres définissant son milieu.

- de simuler un ou plusieurs évènements suivant un générateur de nombres aléatoires, en application des méthodes Montè Carlo.

- de contrôler le déplacement de particules à travers les différentes régions de l'environnement en respectant les effets physiques dûs au type de la particule et à son interaction avec la matière du volume dans lequel elle se trouve, et aux champs magnétiques.

- de récupérer les éléments des trajectoires des particules et les réponses des détecteurs sensitifs.

- de visualiser les détecteurs et les trajectoires de façon interactive ou en fin de simulation.

Plusieurs procédures doivent être écrites par l'utilisateur. Elles sont appelées depuis GEANT3 et doivent exécuter des tâches simples et précises dépendant de l'expérience (définition de la géométrie, gestion des particules secondaires, etc ...).

1.1.2 Décomposition séquentielle de GEANT3

La structure algorithmique de GEANT3 comprend trois parties (fig 3) :

- *Initialisation:* cette phase est sous contrôle de l'utilisateur, elle permet après initialisation des variables de GEANT3 et du gestionnaire dynamique de mémoire ZEBRA (cf 1.2) par des appels à des procédures prédéfinies de GEANT3, de définir la géométrie de l'environnement, les propriétés des milieux utilisés pour chaque volume et de préciser quels volumes sont considérés comme détecteurs sensitifs. Les histogrammes doivent aussi être créés par l'utilisateur.

- *Exécution:* pour chaque évènement simulé, un travail similaire sera entrepris (GRUN). Après initialisation de l'environnement local à un évènement, celui – ci sera déclenché (GTRIG), puis l'espace mémoire nécessaire à cette simulation sera restitué. Lors de la simulation d'un évènement, quatre actions séquentielles sont entreprisent :

  - Initialisation des structures de données décrivant la cinématique de l'évènement courant par lecture ou calcul. (GUKINE)

  - Contrôle du suivi de toutes les particules produites depuis l'évènement (GTREVE). Chaque particule est suivie l'une après l'autre, lorsqu'un détecteur sensitif est atteint, l'utilisateur peut stocker toutes informations utiles caractérisant le choc de cette particule avec ce détecteur. L'évènement sera considéré comme terminé, lorsque toutes les particules secondaires produites par une particule déjà suivie, stockées par l'utilisateur dans une pile temporaire, auront été elles aussi suivies.

  - Simulation des réponses des détecteurs pour l'évènement en utilisant l'information préalablement rangée par l'utilisateur (GUDIGI). Les résultats produits sont ensuite assemblés dans une nouvelle structure de données. Ces résultats sont très dépendants de l'expérience simulée.

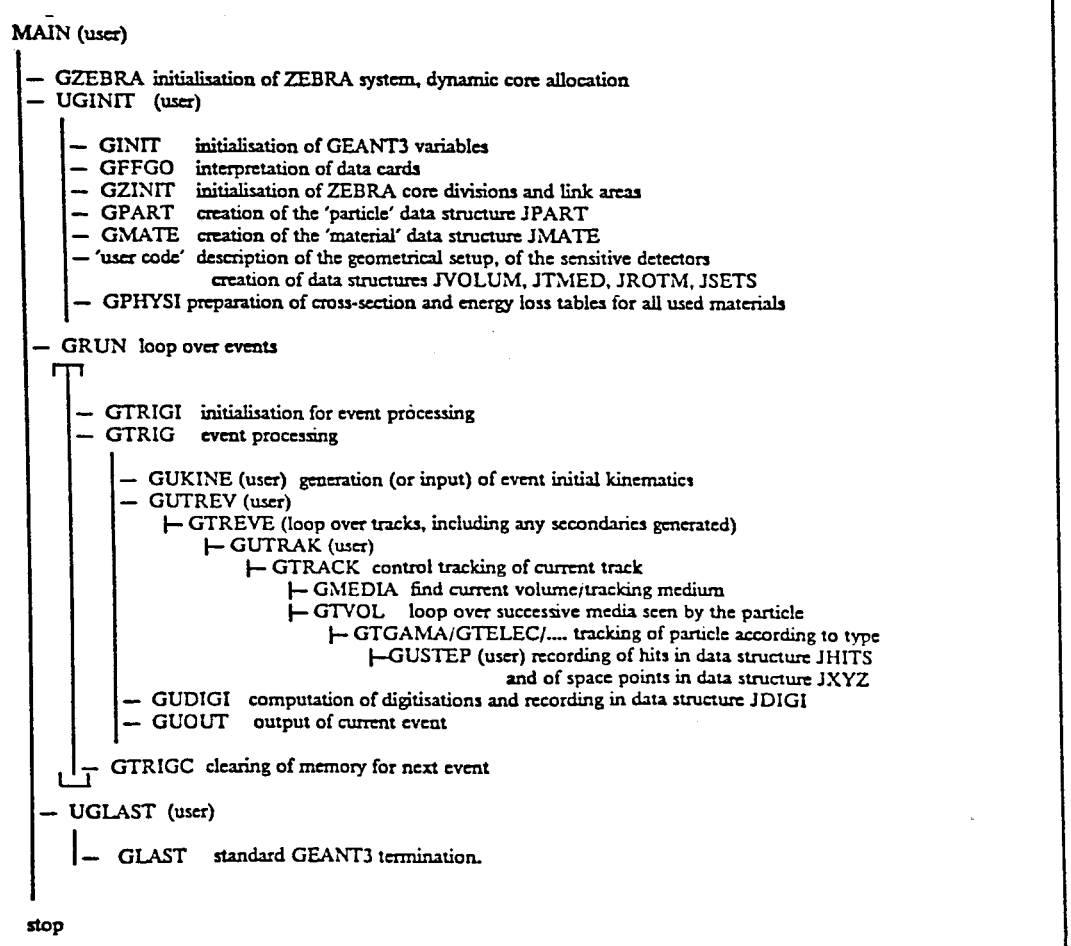


Figure 3: Algorithme de GEANT3

BU LILLE

- Sauvegarde des structures de données produites par l'évènement courant avant remise à zéro (GUOUT).

Après chaque pas de progression d'une particule, le contrôle est rendu à l'utilisateur (GUSTEP). Il peut décider du stockage des particules secondaires, des réponses des détecteurs et de l'évolution géométrique de la trajectoire.

- *Terminaison:* sous contrôle de l'utilisateur (UGLAST), il s'agit de construire et d'imprimer les statistiques produites par tous les évènements simulés. C'est par l'intermédiaire de procédures spécialisées de HBOOK [10] que cela est réalisé.

1.2 Structure de données de GEANT3

A part quelques commons FORTRAN contenant surtout des variables liées à la simulation d'un évènement, le reste des données est rangé dans une partie de la mémoire gérée par le système de gestion de données ZEBRA. [12] Cela permet d'implémenter toutes structures de données sans avoir à modifier le code de GEANT3.

1.2.1 Présentation de ZEBRA

Tous les logiciels scientifiques ont été écrits, pour différentes raisons, en FORTRAN. Ce langage, bien que très utilisé, ne propose pas de manipulation de données structurées dynamiques. Les seules types de données manipulables sont des tableaux homogènes et des commons de données partagées. Ces données ne peuvent être manipulées en temps qu'objets, et ne peuvent être gérées dynamiquement pendant l'exécution. Les pointeurs entre structures doivent être simulés par l'utilisateur avec des indices de tableaux. Cela oblige à recompiler à chaque changement de taille des tableaux, l'ensemble des logiciels.

C'est afin d'éviter ce genre de désagrément que ZEBRA a été conçu. Il permet non seulement une véritable création dynamique de structures de données, mais assure aussi leur portabilité entre plusieurs ordinateurs de types différents. ZEBRA est utilisable à partir de FORTRAN par l'appel de procédures. ZEBRA procure à FORTRAN l'outil indispensable qui lui manquait, avec un temps d'exécution souvent négligeable.

Les données ZEBRA sont stockées dans une banque, espace contiguë de mémoire. Par un simple appel, l'utilisateur peut en créer une de taille quelconque. Une banque pourra contenir des informations de différents types: entier, réel, ...

La structure de données générale de ZEBRA peut être décrite par deux liens : *DOWN, NEXT.* Cela permet de construire tout arbre binaire (fig 4) ou quelconque si le nombre de liens *DOWN* est supérieur à un. Souvent, il est ausssi nécessaire de manipuler des structures plus complexes de type graphe dans lesquelles d'autres liens existent entre les données, ZEBRA propose alors d'utiliser les liens de références. Leur nombre peut être quelconque, il est défini par l'utilisateur en fonction de ses besoins. Des liens vers les origines sont aussi disponibles *UP* et *ORIG.* Les liens et les données sont stockés dans la même banque ZEBRA. (fig 5)

L'utilisateur est chargé de gérer l'information contenue dans une banque, il doit aussi assurer les liens logiques, si il y en a, entre chaque banque. ZEBRA assure la gestion de l'espace mémoire et permet aussi l'écriture et la reconstruction de structures complexes à partir d'un point d'accès. Les informations peuvent être codées dans un format indépendant de la machine.

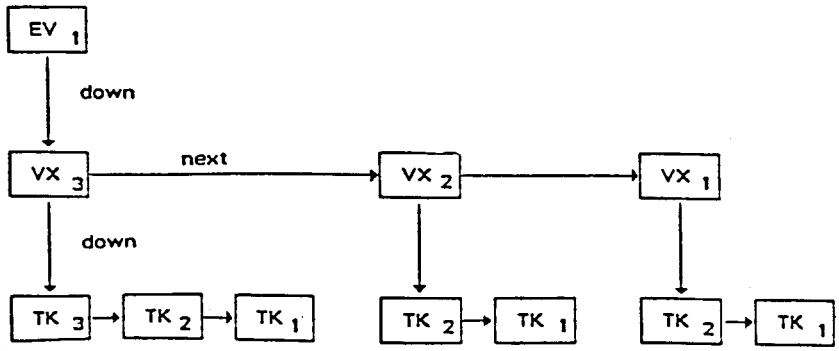


Figure 4: Structure arborescente binaire

1.2.2 Structures de données statiques

L'environnement expérimental est donc représenté par une structure de volumes. Un milieu est associé à chaque volume, un même milieu pouvant être attribué à des volumes différents. Le milieu est défini par un ensemble de paramètres qui le caractérisent et par le type de matériau qui le compose. Lors du suivi d'une particule, les informations suivantes sont nécessaires :

- les caractéristiques des matériaux utilisés
- les paramètres caractérisant le milieu
- les propriétés de la particule
- la définition de la géométrie

1.2.2.1 Les matériaux

Les constantes définissant les matériaux sont stockées dans la structure de données JMATE. Une banque ZEBRA est créée pour chaque matériau.

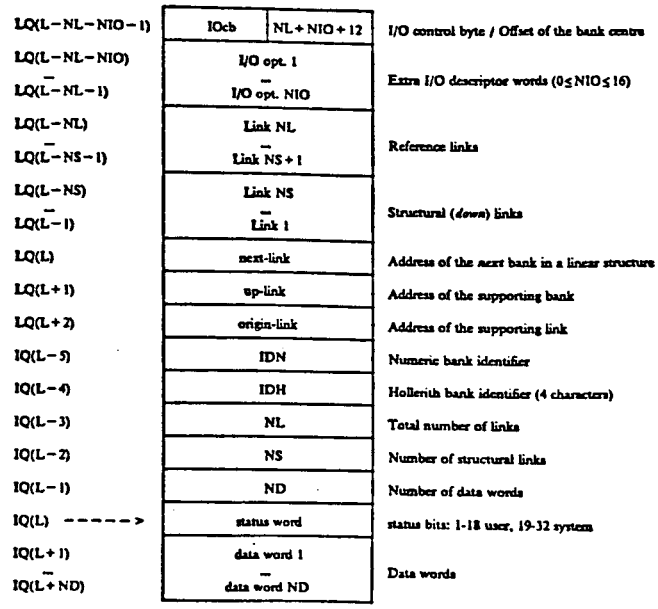


Figure 5: Gestion interne des liens

L'utilisateur peut aussi définir des nouveaux matériaux par mélange de matériaux de base ou par composition moléculaire. Plusieurs procédures permettent de créer, imprimer, retrouver les caractéristiques d'un matériau. Dans la structure, seront aussi conservées les propriétés physiques associées aux matériaux, perte d'énergie par type de particule et tables de "cross – section" (fig 6).

1.2.2.2 Les paramètres caractérisant le milieu

Pour chaque milieu défini, les paramètres qui le caractérisent sont rangés dans la structure JTMED. Les procédures de stockage, d'extraction et d'impression sont aussi disponibles. Les paramètres définissant les niveaux minimums d'énergie acceptables par type de particule et les indicateurs sélectionnant les processus physiques sont définis pour tous les milieux, l'utilisateur peut changer ces valeurs pour un ou plusieurs milieux, les nouvelles valeurs sont alors elles aussi rangées dans JTMED. (fig 7)

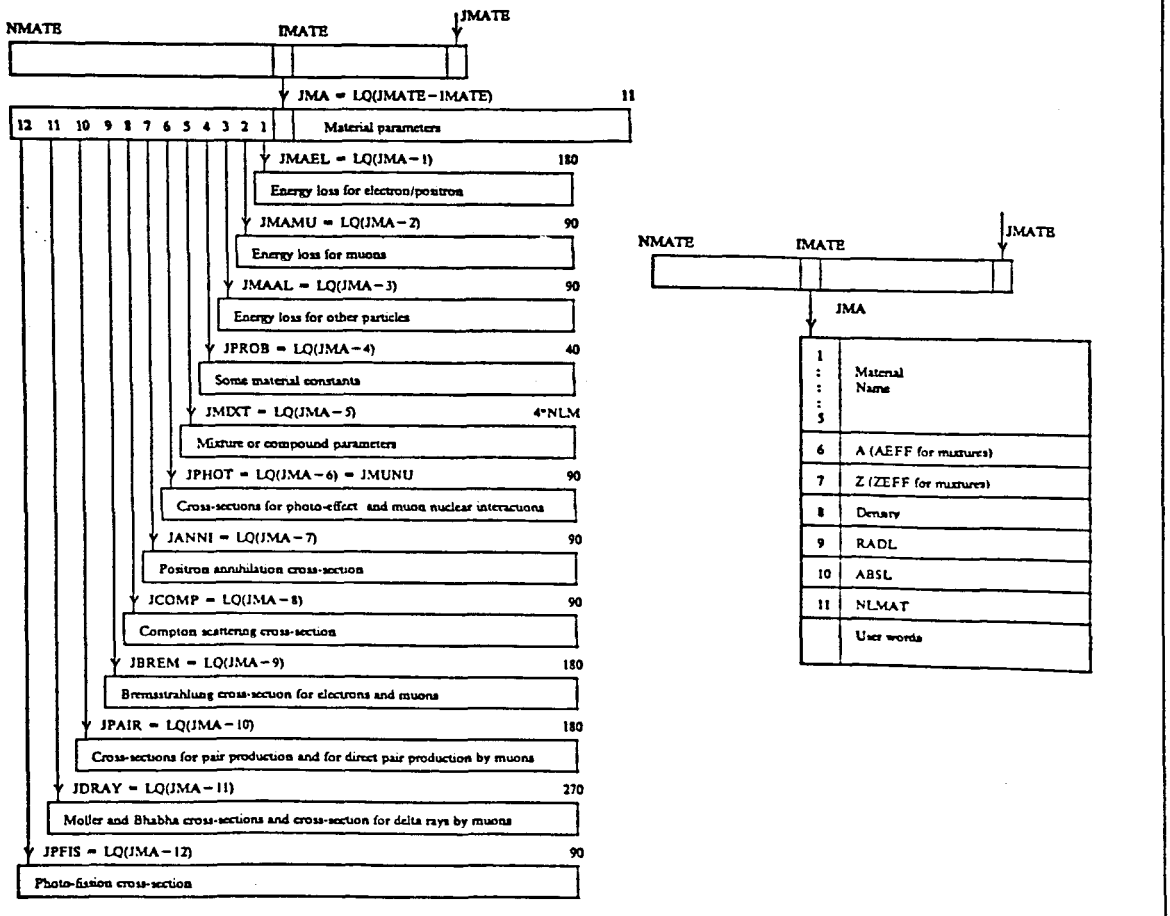


Figure 6: Structure des matériaux : JMATE

1.2.2.3 Les particules

Les constantes physiques des particules sont stockées dans la structure JMATE. La liste des particules secondaires engendrées par désintégration et la répartition de l'énergie sur chacune d'entre elles, sont aussi rangées pour chaque particule. Un ensemble de particules standards est défini automatiquement par GEANT3. Des procédures de manipulation sont disponibles. (fig 8)

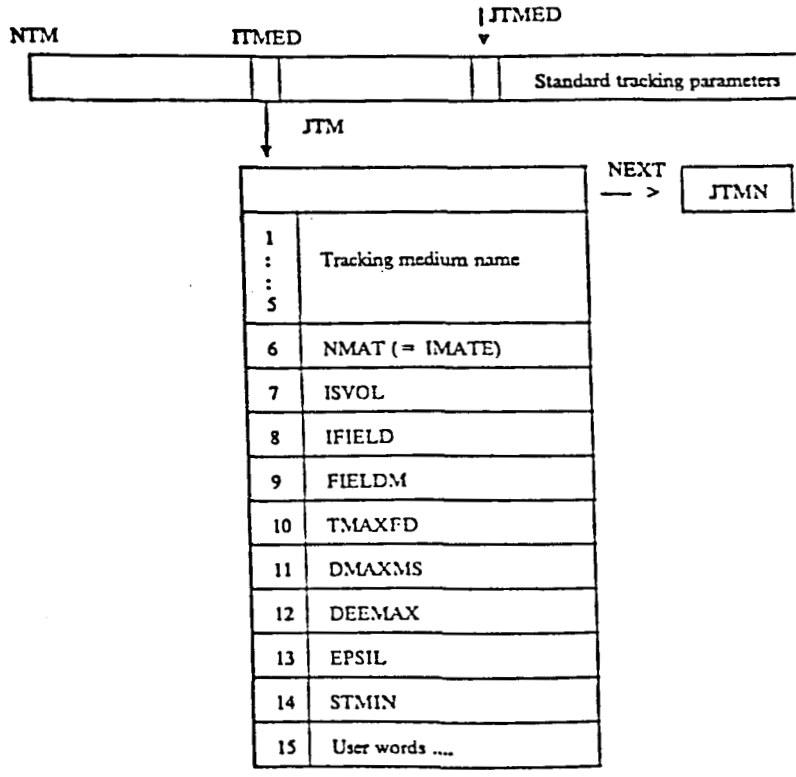


Figure 7: Structure associée aux milieux : JTMED

BU LILLE

1.2.2.4 La géométrie

L'environnement expérimental est défini par l'utilisateur pendant la phase d'initialisation. Aussi complexe qu'il puisse être, il est toujours possible de le décomposer en un ensemble de volumes élémentaires. Une structure d'arbre peut être définie entre tous ces volumes (fig 9).

Cet arbre est construit à partir de ses feuilles : ce sont les volumes les plus internes de la géométrie. Souvent la géométrie d'une expérience est composée de parties identiques, aussi pour éviter de dupliquer l'information et ainsi minimiser l'allocation d'espace mémoire, des entités représentant des sous – arbres sont définis : une entité présente plusieurs fois dans l'environnement ne sera présente en mémoire qu'une seule fois. Chaque entité est définie par :

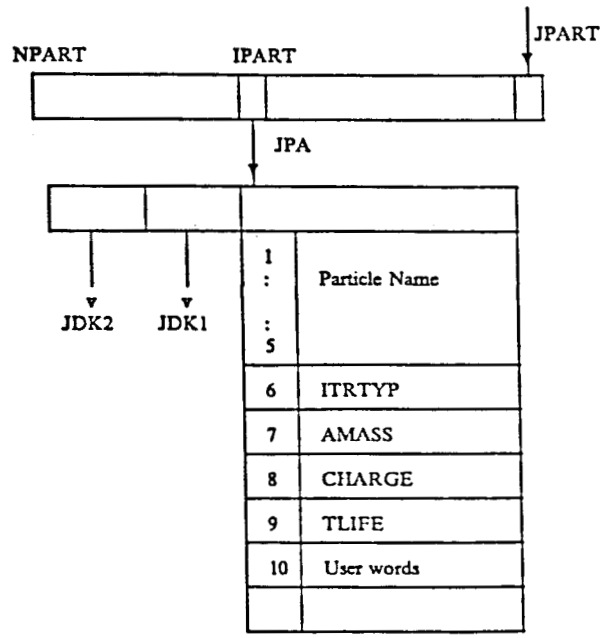


Figure 8: Structure des particules : JPART

BU Lille

- son nom
- un identificateur de forme, spécifiant quelle forme est utilisée parmi celles possibles (tube, sphère, boite ...) pour le volume de base de l'entité. Celui – ci est en fait la racine du sous – arbre défini par cette entité.
- un système de référence associé au type de forme pour ce volume de base (cartésien, cylindrique ou sphèrique)
- ses propriétés physiques inhérentes aux propriétés du matériau qui compose ce volume
- les paramètres spécifiant la taille du volume de base. Le nombre de paramètres est dépendant du type de forme utilisé
- la liste des entités déjà définies, accessibles directement à partir de la racine du sous – arbre. Cette liste peut être vide pour un volume ne contenant aucun autre volume.

L'arbre associé à la géométrie complète de l'environnement expérimental est une entité privilégiée. Elle ne peut être incluse dans aucune autre entité, elle sert de référence pour l'environnement. Elle contient tous les volumes présents dans la

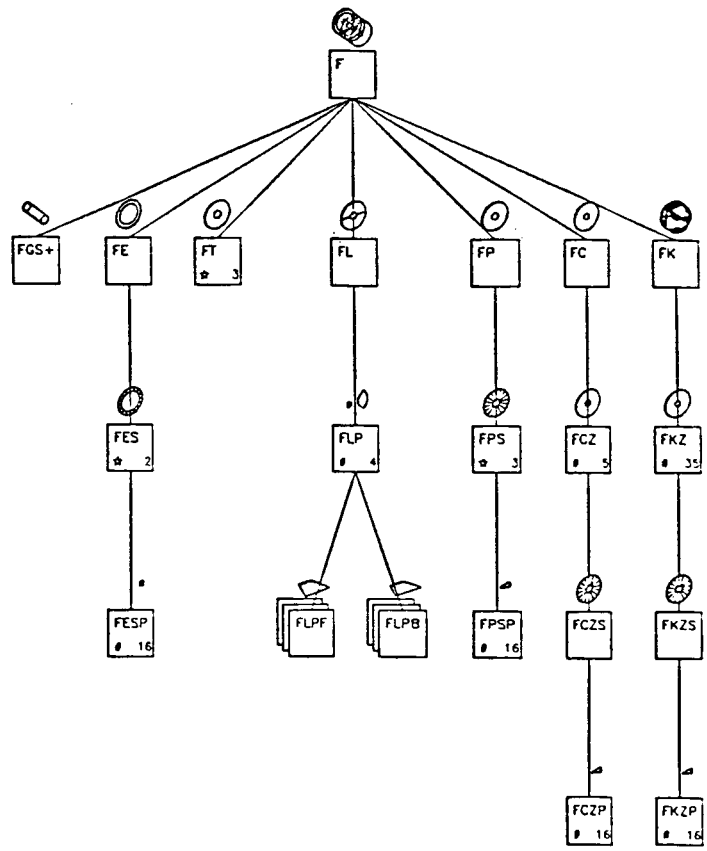


Figure 9: Exemple d'arbre géométrique

BU LILLE

géométrie, qu'ils soient ou non accessibles directement à partir de cette entité. (fig 10)

Une entité est définie en deux phases distinctes. L'utilisateur doit d'abord définir les caractéristiques du volume de base lié à cette entité. Puis s'il le faut, il peut y placer d'autres entités, deux types de placements sont alors possibles, un seul étant utilisable à la fois pour une même entité :

- *Par positionnement:* L'utilisateur peut positionner des entités déjà existantes en spécifiant le nom de l'entité, sa position dans le repère de l'entité "mère" et

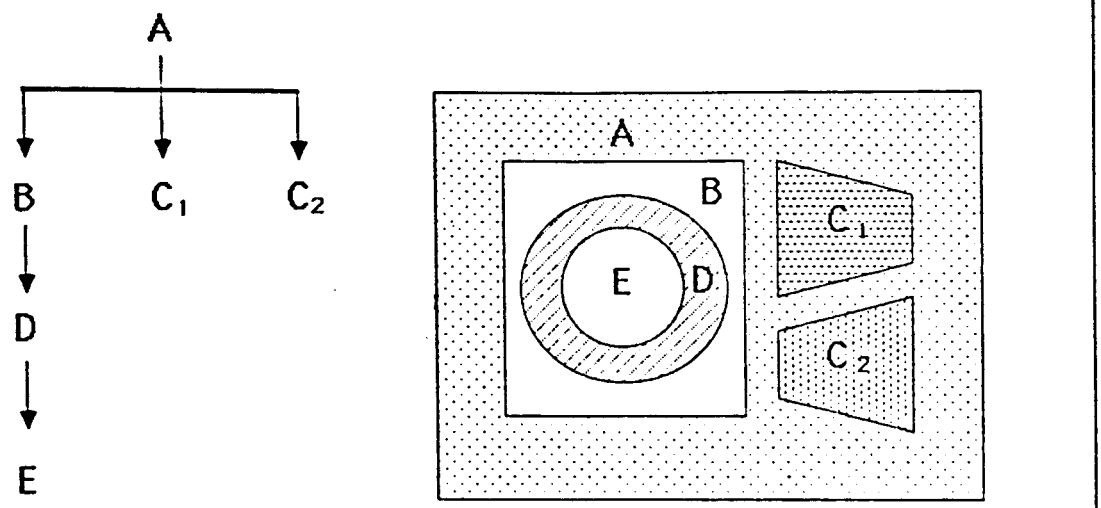


Figure 10: Arborescence de volumes

un numéro. Le couple (nom , numéro) doit être unique pour une même entité "mère", ainsi l'on peut positionner plusieurs fois la même entité avec un numéro différent. Les entités positionnées peuvent ne représenter qu'une partie de l'espace défini par le volume "mère", mais elles ne peuvent être extérieures à cet espace. Deux entités peuvent se chevaucher afin de permettre des décompositions en volumes artificiels pour délimiter des régions de la géométrie, un indicateur devra toutefois le préciser lors du positionnement.

- *Par division:* Le volume associé à l'entité "mère" peut être divisé suivant l'un de ses axes, en général le nombre de divisions est donné par l'utilisateur, une nouvelle entité est ainsi créée. Tout l'espace du volume "mère" est partagé de façon régulière, entre toutes les divisions. Chaque division correspond au positionnement de la même entité. La taille de cette entité n'est pas toujours constante (division d'un trapèze parallèlement à sa base) mais elle peut être calculée pendant la simulation. Cette entité ainsi construite, possède les mêmes propriétés que les entités construites par positionnement : elle peut contenir d'autres entités placées par positionnement ou par division.

Les procédures de manipulation d'entités sont définies par GEANT3. La structure de volumes est représentée en mémoire par un graphe ou chaque entité peut référencer plusieurs autres entités (fig 11).

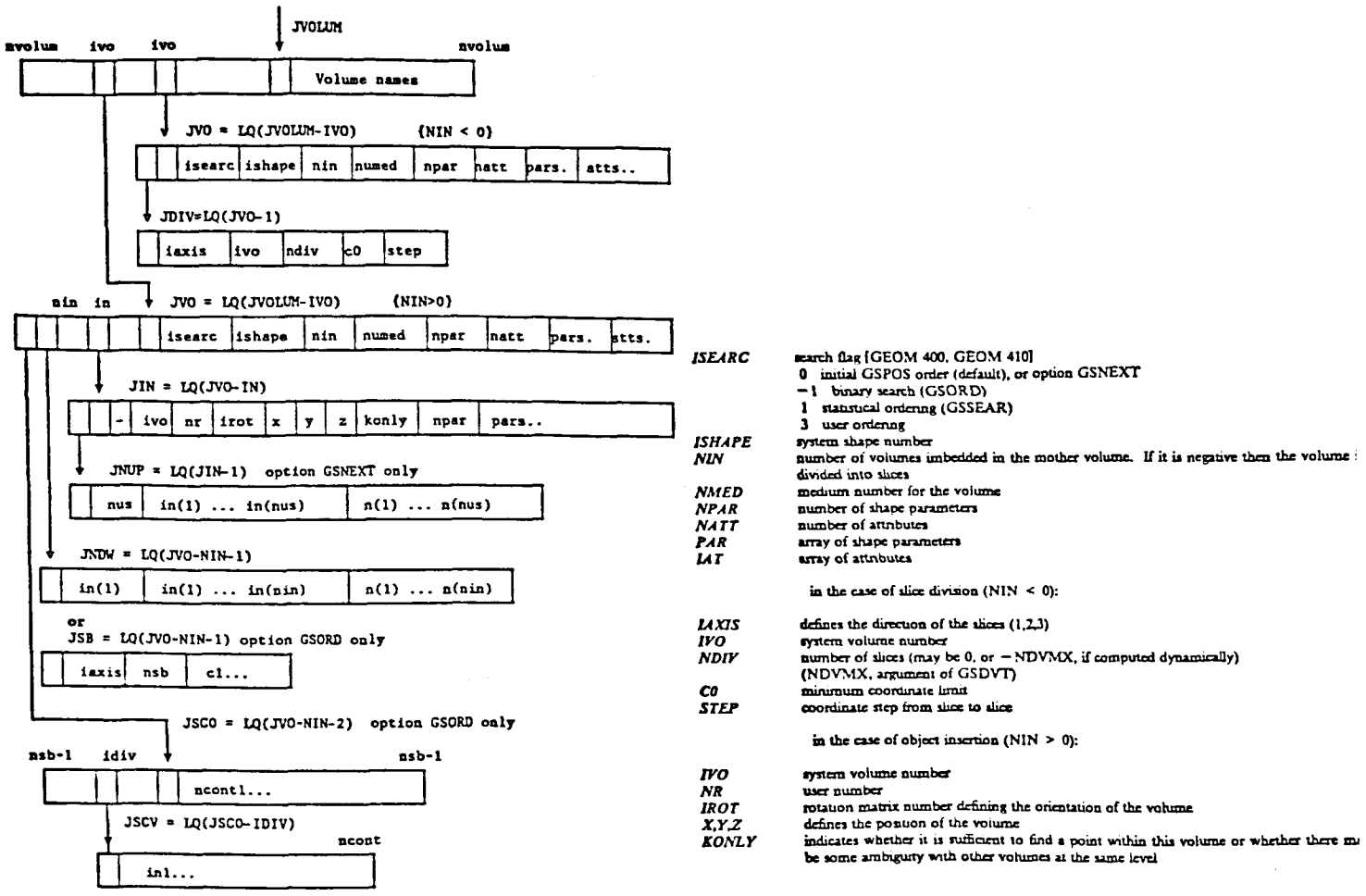


Figure 11: Structure de la géométrie : JVOLUM

1.2.2.5 Matrices de rotation

Lors du positionnement d'un volume, l'utilisateur peut orienter le volume dans le repère local du volume mère en définissant la matrice de rotation entre ce repère et celui du volume positionné. Ces matrices sont rangées dans la structure JROT. (fig 12)

1.2.2.6 Les détecteurs sensitifs

L'utilisateur, après définition de sa géométrie, peut identifier certains volumes comme étant des détecteurs sensitifs. Ces détecteurs peuvent être regroupés. Lors de la simulation, un ou plusieurs groupes de détecteurs seront actifs. Cette structure de données est créée lors de la phase d'initialisation. (fig 13)

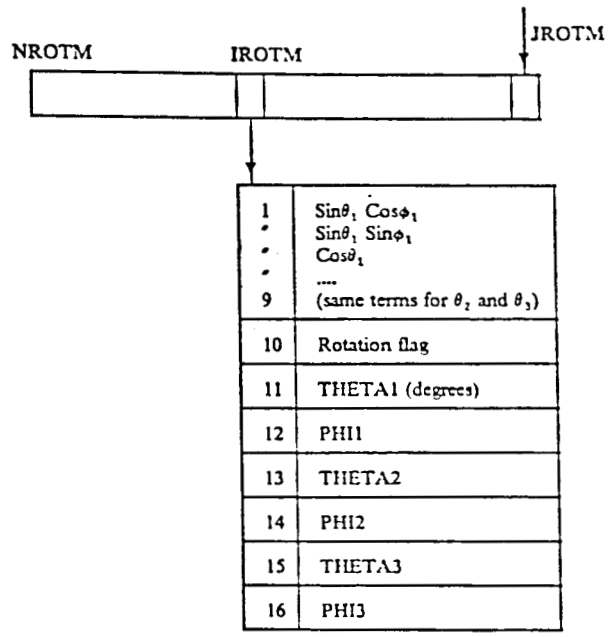


Figure 12: Structure des matrices de rotation : JROT

BU LILLE

1.2.3 Structures de données dynamiques

Contrairement aux structures précédentes, celles – ci vont être modifiées pendant la simulation de chaque évènement. Elles sont placées dans une division de ZEBRA différente pour permettre une gestion de l'espace mémoire plus simple.

1.2.3.1 Les particules générées

Lors de la simulation d'un évènement, un certain nombre de particules secondaires peuvent être générées. L'utilisateur peut, après chaque génération, décider de conserver ces particules dans une pile temporaire JSTAK (fig 15) pour la suivre ultérieurement, ou encore dans une file avec mémorisation JKINE/JVERTX (fig 14) : permettant de conserver la trace de certaines particules générées pendant la simulation pour une analyse ultérieure. L'ordre dans lequel sont suivies les particules n'a pas d'importance, les notions de pile et de file ont été choisies afin de simplifier la gestion de ces deux stockages de particules. Les particules non conservées ne peuvent en général intervenir sur les détecteurs et ne

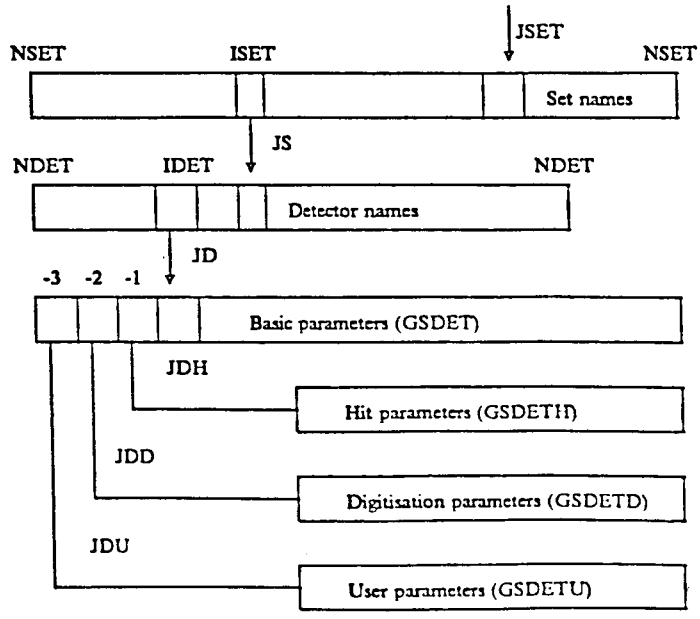


Figure 13: Structure des détecteurs : JSET

peuvent générer d'autres particules qui elles pourraient déclencher une réponse de l'un de ces détecteurs. Ces structures sont donc de tailles variables, elles sont vidées entre chaque évènement simulé. L'allocation et la restitution de l'espace mémoire sont assurées par ZEBRA.

1.2.3.2 Les trajectoires de particules

La structure JXYZ permet de conserver des éléments de trajectoires. Une banque ZEBRA est créée pour chaque trajectoire des particules générées depuis une même particule extraite de JKINE/JVERTX. (fig 16)

1.2.3.3 Réponses des détecteurs

Lors du déroulement d'un évènement, des particules peuvent rencontrer des détecteurs sensitifs. Les informations caractérisant ces chocs peuvent être conservées dans la structure JHITS après chaque pas de progression de la particule dans le détecteur. C'est par la procédure GUSTEP que l'utilisateur décide ou non de conserver ces informations. (fig 17)

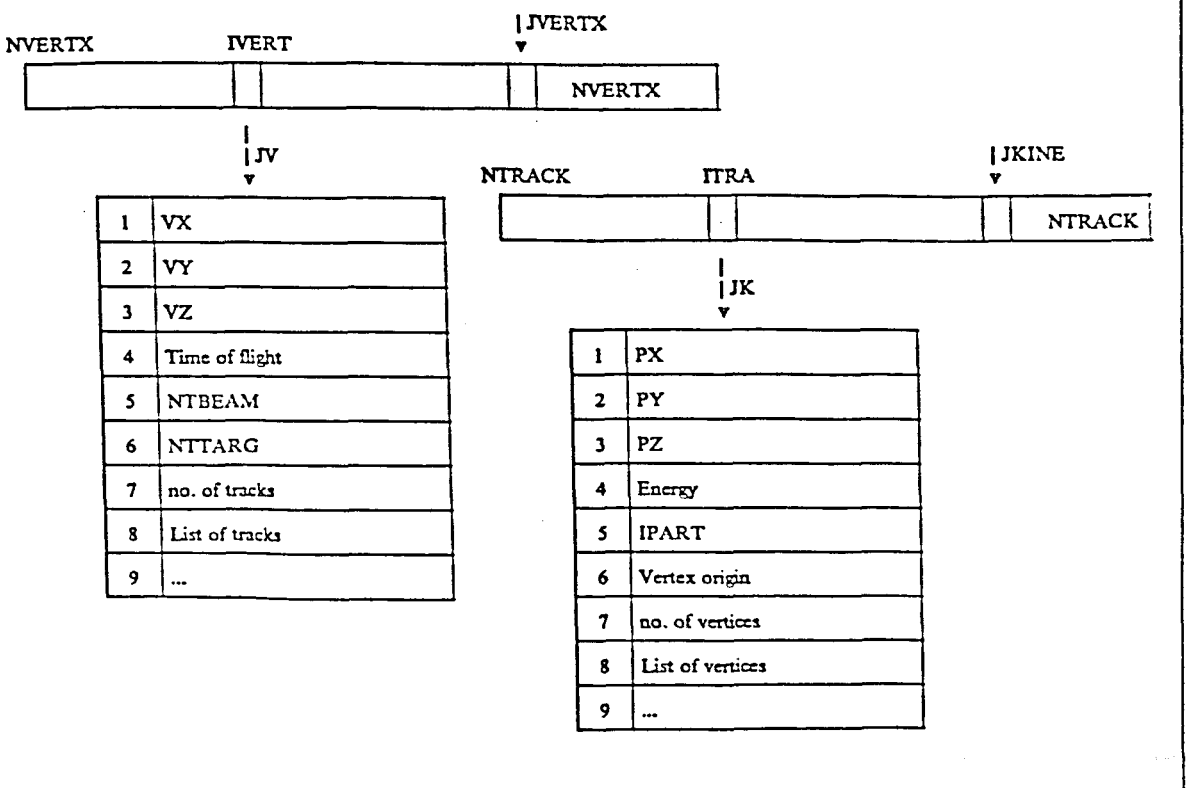


Figure 14: Structure de file des particules : JKINE/JVERTX

Quand la simulation de l'évènement est terminée, la digitisation de l'information précédemment conservée peut être entreprise. Il s'agit de produire une information simulant la réponse d'un ensemble de détecteurs donnés pour cet évènement en fonction des chocs précédemment enregistrés. Une nouvelle structure de données JDIGI contient alors ces résultats. (fig 18)

1.3 Algorithme de simulation

Comme nous l'avons vu auparavant, la procédure GRUN est chargée d'assurer la simulation de tous les évènements les uns après les autres. (fig 19)

C'est la procédure GTRIG qui simule chaque évènement (fig 20), la procédure GTRIGI initialise l'environnement associé à un évènement, GTRIGC restitue l'espace mémoire utilisé par les structures de données.

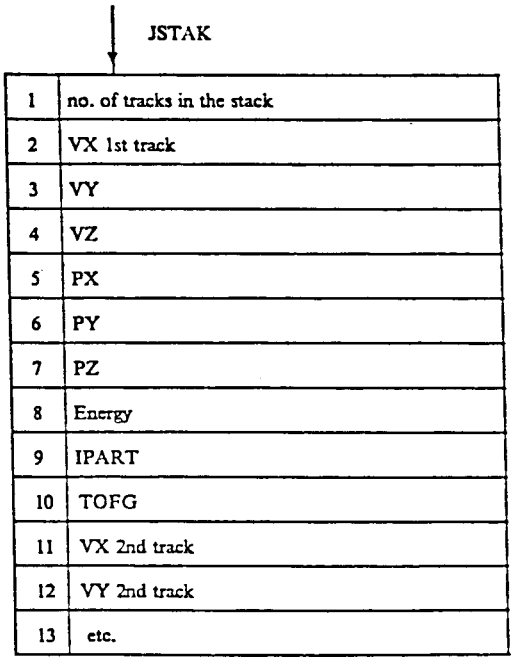
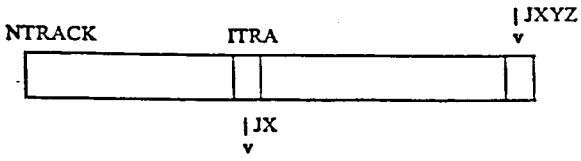


Figure 15: Structure de pile de particules : JSTAK

BU LILLE

La simulation d'un évènement dans l'environnement expérimental consiste en particulier, à prévoir les coordonnées spaciales d'un ensemble de points définissant la trajectoire des particules. Chacune sera suivie pas à pas en tenant compte des perturbations introduites par la nature du milieu. Un ensemble de procédures de GEANT3 contrôlent et exécutent le suivi de toutes les particules de l'évènement et des particules secondaires éventuellement générées. Entre chaque nouvelle phase de simulation, le contrôle est rendu à l'utilisateur, il peut ainsi entreprendre diverses actions telles que sauvegarde, lecture ou impression de données. (les procédures utilisateurs sont toujours appelées GUxxxx)

Lors du suivi d'une particule à travers un milieu complexe, l'une des tâches critiques est d'estimer à priori la taille du pas de progression. Elle dépend en fait des effets physiques en action et de la nature géométrique. Une particule ne peut franchir en un seul pas une frontière entre deux volumes. Les structures de données construites pendant l'initialisation sont naturellement utilisées.



1	no. of free words in the bank
2	Pointer to current stack track
3	no. of points for the primary track = N
4	IPART Particle code
5	X1, Y1, Z1
8	X2, Y2, Z2
11	etc.
•	no of points for the 1st stack track
•	IPART Particle code
•	X1, Y1, Z1
•	X2, Y2, Z2
•	etc.
•	:

Figure 16: Structure des trajectoires : JXYZ

BU LILLE

1.3.1 Simulation d'un évènement

La procédure GUTREV appelle GTREVE. Celle-ci effectue la simulation proprement dite d'un évènement. Elle suit successivement toutes les particules présentes dans la file JKINE. Pour chacune, il faut extraire les propriétés associées à son type de particule depuis la structure JPART. La procédure GTRACK assure ensuite le suivi de cette particule jusqu'à la fin : arrêt, désintégration ou sortie de l'environnement. Pendant cette phase, des particules secondaires ont pu être générées et stockées par l'utilisateur dans la pile temporaire JSTAK, ou dans la file mémorisante JKINE/JVERTX. Une particule ne modifiant pas, lors de sa progression, les caractéristiques de l'environnement expérimental et la probabilité pour que deux particules puissent entrer en collision étant nulle, il n'est pas utile de considérer le temps comme un paramètre de la simulation. Chaque particule générée peut être suivie l'une après l'autre et cela de façon complètement indépendante. Avant de passer à une particule suivante de JKINE, GTREVE simule d'abord toutes les particules présentes dans JSTAK. (fig 21)

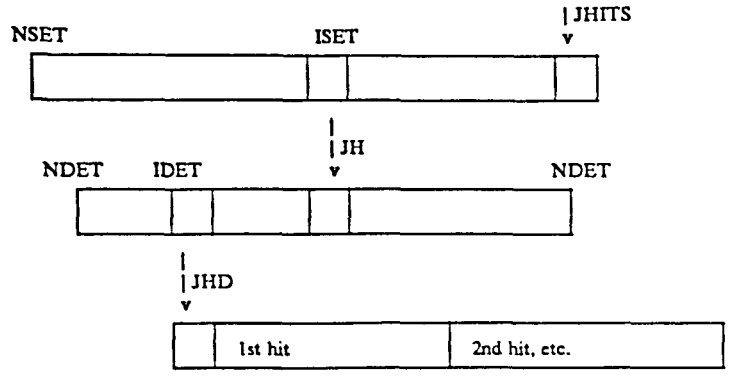


Figure 17: Structure des chocs avec détecteurs : JHITS

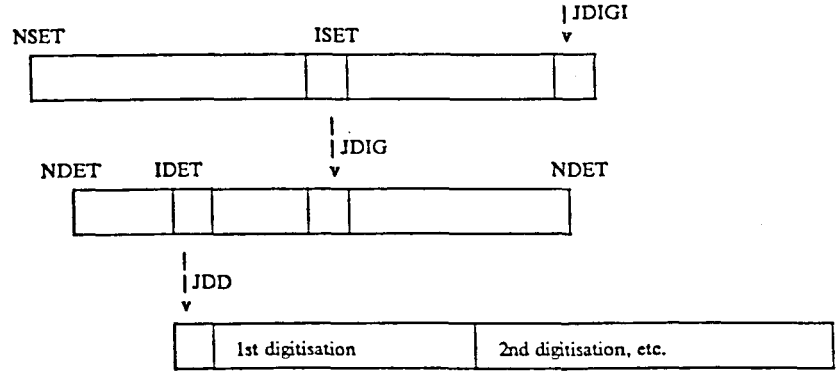


Figure 18: Structure de digitisation : JDIGI

1.3.2 Progression d'une particule

GTRACK, appelée par GUTRAK, effectue le suivi d'une particule dans tous les volumes qu'elle rencontre. Il faut d'abord identifier le volume dans lequel se

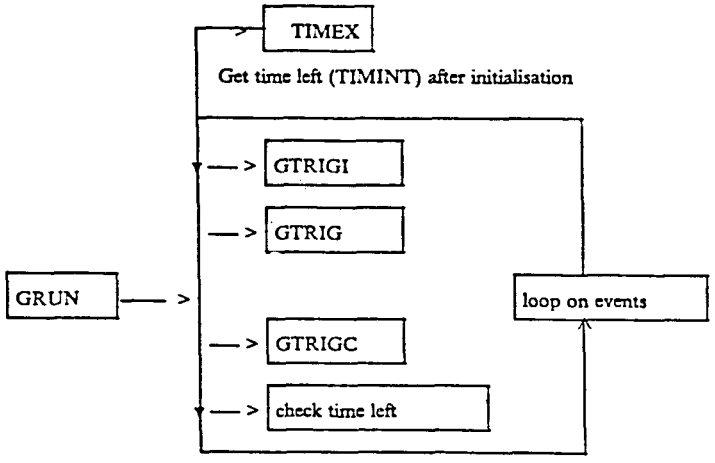


Figure 19: Algorithme de GRUN

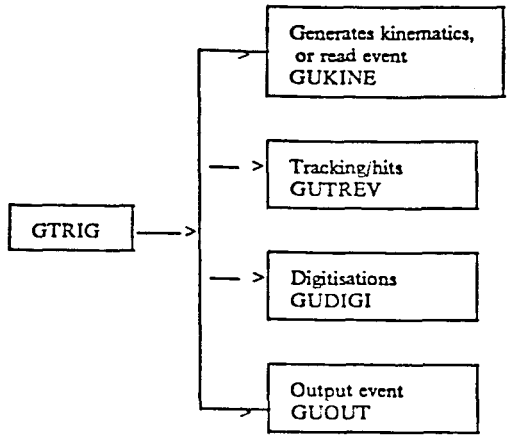


Figure 20: Algorithme de GTRIG

BU LILLE

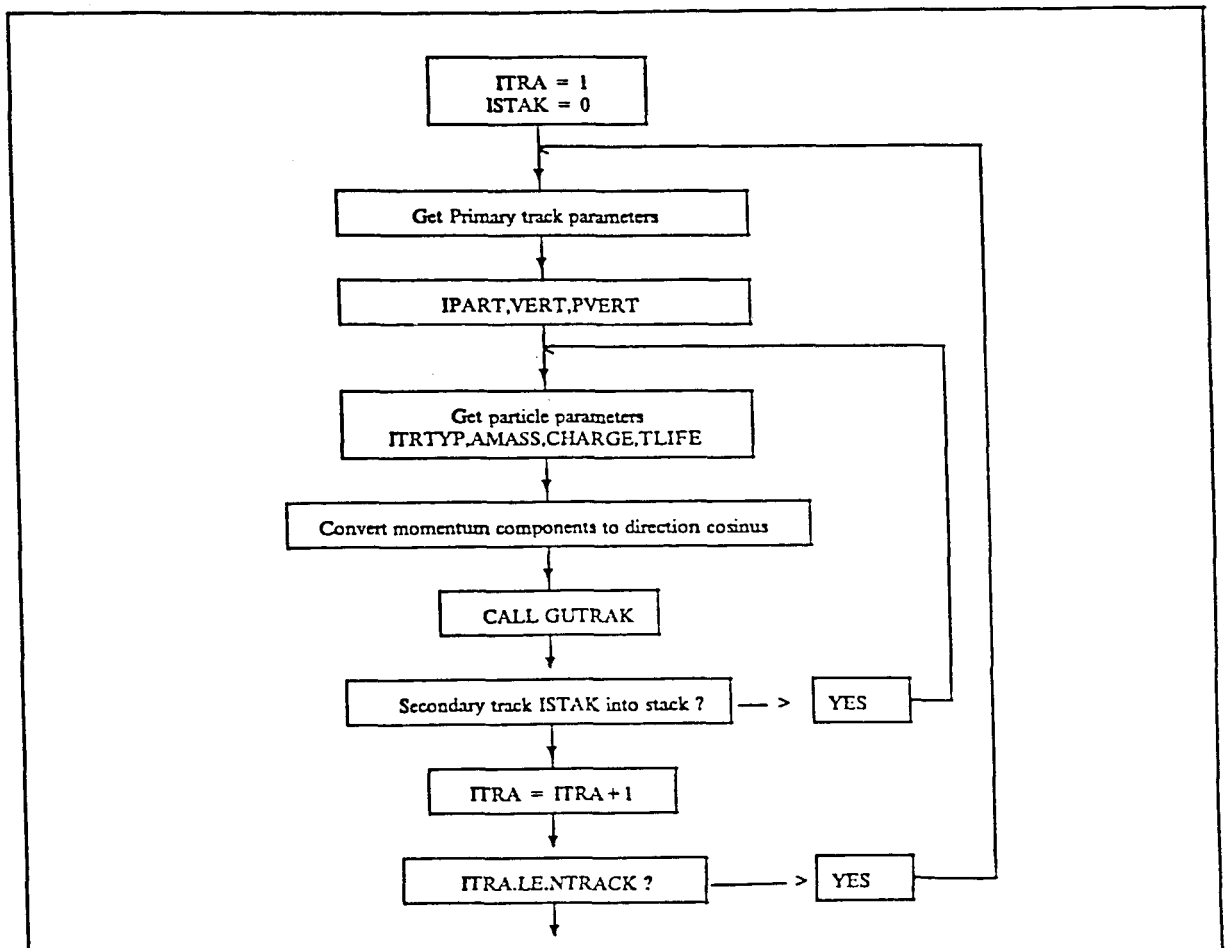


Figure 21: Algorithme de GTREVE

trouve la particule (GMEDIA) afin d'obtenir les paramètres caractérisant le milieu. Le contrôle est ensuite donné à GTVOL qui assurera le suivi de cette particule à travers le volume courant ou jusqu'à son arrêt. (fig 22)

1.3.3 Algorithme du suivi d'une particule dans un volume

La procédure GTVOL effectue le suivi d'une particule pas à pas dans le volume courant. Par la procédure GNEXT, les limites géométriques pour un pas de progression sont calculées. Puis en fonction du type de particule, les limites physiques sont calculées par l'une des procédures : GTELEC, GTGAMA, GTNEUT, GTHADR, GTMUON et GTNINO. La particule est propagée d'une distance n'excédant pas le minimum des limites calculées. (fig 23)

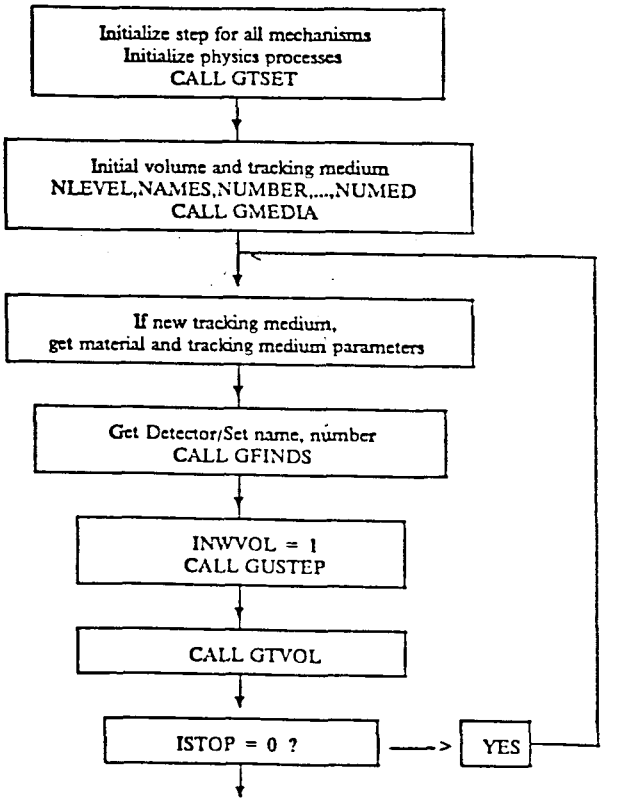


Figure 22: Algorithme du suivi d'une particule : GTRACK

BU LILLE

Après chaque pas, le contrôle est rendu à l'utilisateur (GUSTEP), il peut alors décider des actions à entreprendre pour stocker les résultats produits par cette progression. (choc avec un détecteur, particules secondaires, trajectoire)

1.3.4 Interface avec la géométrie

Le suivi d'une particule à travers les différents milieux de la géométrie peut être exécuté sans l'intervention de l'utilisateur. La connexion entre géométrie et algorithme de simulation est assurée par trois procédures :

- *GMEDIA:* scrute l'arborescence géométrique afin de trouver dans quel milieu la particule se trouve. A partir du volume courant, il s'agit de remonter dans l'arbre jusqu'à obtention d'un volume contenant cette particule. Ensuite

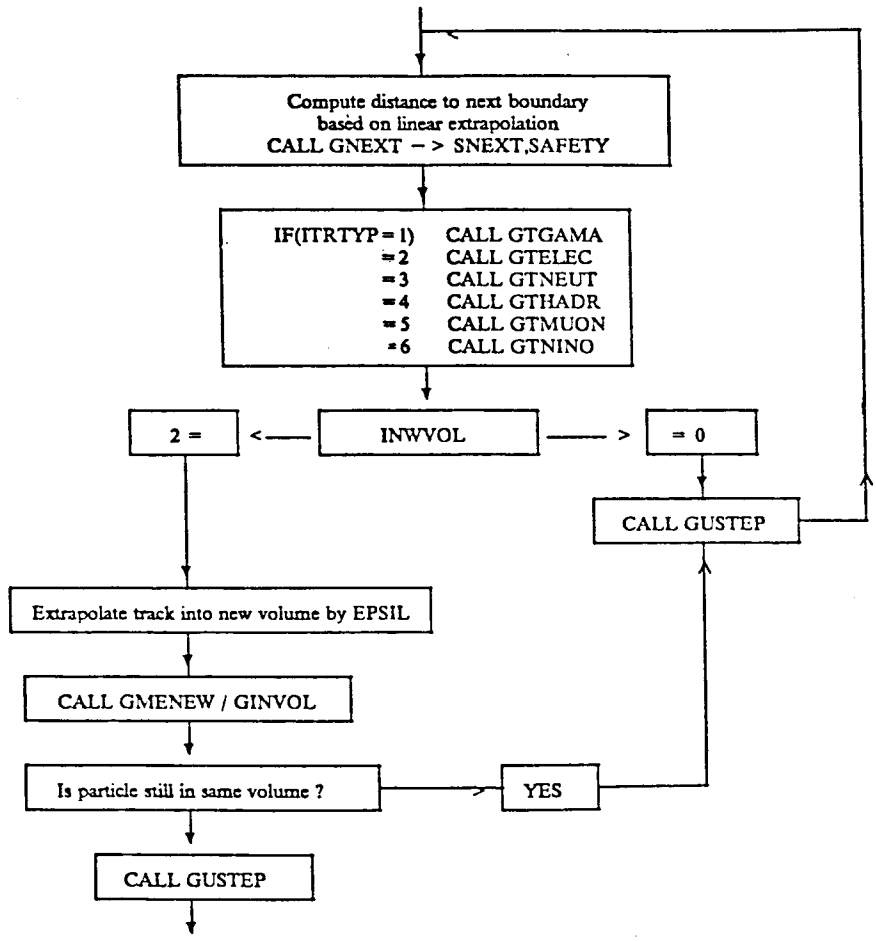


Figure 23: Algorithme de propagation dans un volume : GTVOL

la descente dans l'arbre peut être entreprise pour trouver quel volume contient effectivement cette particule. Lorsque le volume courant contient d'autres volumes positionnés, c'est la procédure GSEAR qui est chargée de trouver quel volume "fils" du volume courant doit être choisi pour cette descente, lorsque le volume courant est divisé c'est la procédure GHDIV qui calcule dans quelle division la particule se trouve. La recherche sera terminée lorsque le volume courant contiendra directement la particule.

- *GNEXT:* calcule la distance entre la particule et la prochaine frontière géométrique suivant la direction suivie par cette particule. Les distances pour toutes les frontières possibles sont calculées, seule la plus petite sera conservée.

- *GINVOL:* vérifie si deux points donnés appartiennent à un même volume. Elle est utilisée avec les positions de la particule avant et après le pas de progression de celle – ci.

1.4 Machines séquentielles à grande mémoire

Les mémoires des calculateurs traditionnels sont bien souvent trop petites pour supporter de grosses applications. Certains algorithmes proposent de répéter des calculs plusieurs fois, ceux – ci donnant toujours les mêmes résultats, afin d'éviter de saturer la mémoire. Sur les puissantes machines actuelles, l'espace mémoire est très large, ces calculs peuvent être faits une fois pour toutes en début d'exécution et les résultats stockés en mémoire.

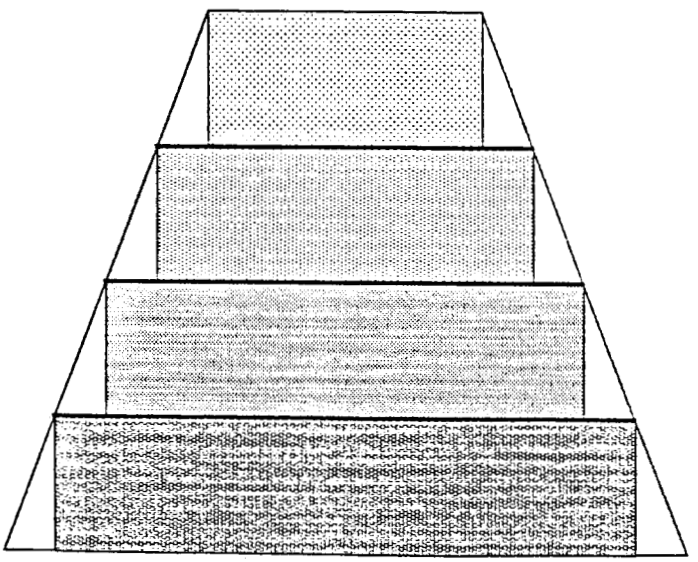


Figure 24: Exemple où GPCHK est nécessaire

BU LILLE

Dans GEANT3 chaque calcul faisant intervenir les paramètres des éléments positionnés dans un volume, déclenche l'appel de la procédure GPCHK. Celle – ci est chargée de vérifier et s'il le faut, de modifier ces paramètres. Ces calculs sont indépendants de tout autre chose et pour un même volume, donneront toujours les mêmes résultats, ils peuvent donc être exécutés une fois pour toutes en début de simulation.

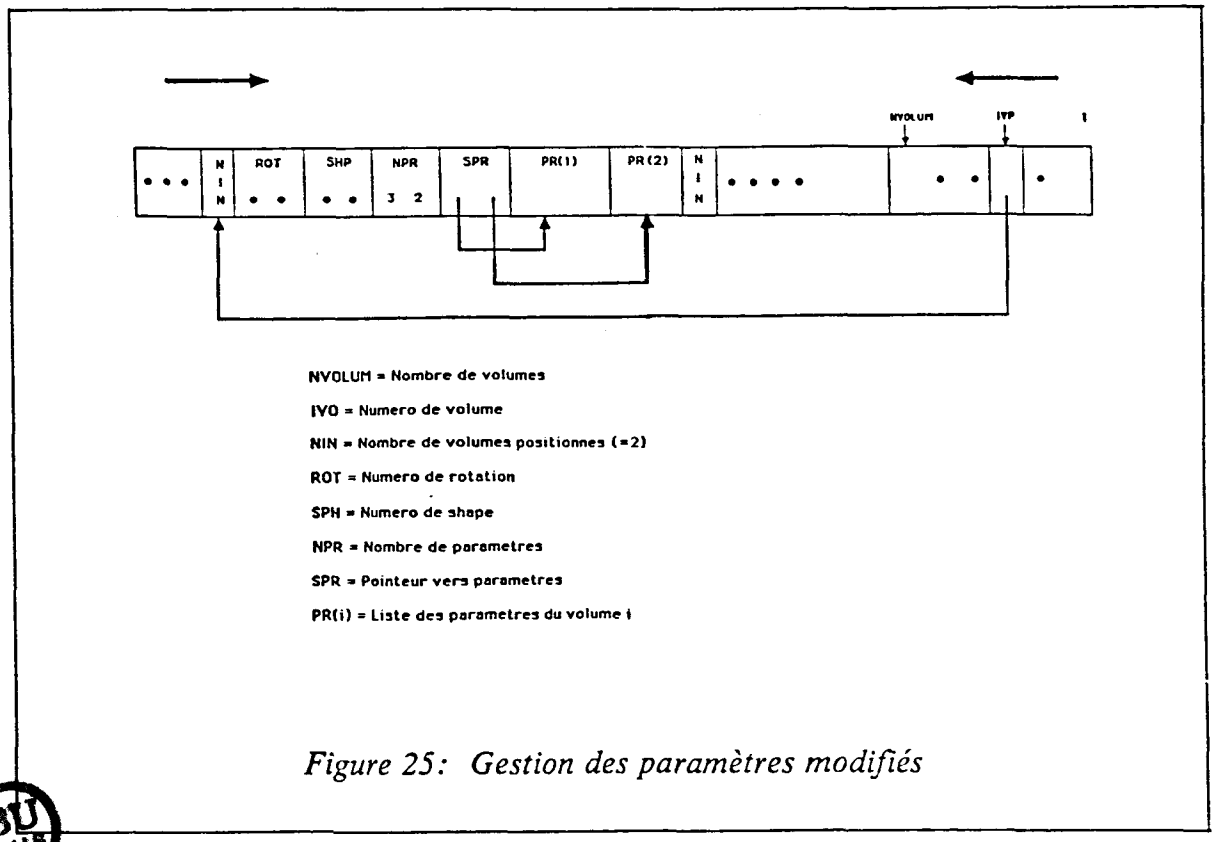


Figure 25: Gestion des paramètres modifiés

BU LILLE

Ce type de solution n'est en fait applicable que sur des expériences à géométrie simple. Pour la géométrie de OPAL, composée d'environ 150000 éléments, la taille nécessaire au stockage des résultats produits par GPCHK serait de l'ordre de 3Mwords (20 mots par volume) soit pour une machine 64 bits, 24 MO. Cela devient, même pour un super – computer, assez encombrant. Dans un volume créé par division, il est possible de positionner un et un seul volume, d'une forme donnée pour lequel les paramètres ne sont pas initialisés. Ceux – ci sont calculés lors de la simulation par GPCHK, en fonction de la taille de la division dans laquelle le volume se trouve. (fig 24) En dehors de ce cas particulier, nous pouvons remarquer que les résultats produits par la procédure GPCHK sont indépendants de la place

effective des volumes dans l'environnement. Le nombre de résultats différents pour ces cas particuliers, est proportionnel au nombre d'entités géométriques créées et non au nombre de volumes positionnés. Pour ce type de géométrie, on peut donc éviter d'appeler GPCHK à chaque pas de progression.

Un common FORTRAN ou une banque ZEBRA peuvent être définis pour stocker ces résultats qui sont : le type de forme, le nombre de paramètres et la listes des paramètres modifiés.(fig 25) Tous les résultats relatifs aux volumes positionnés dans un même volume, seront regroupés dans cet espace. Il faut ensuite pouvoir retrouver cette information en fonction du numéro de volume, d'où l'utilisation d'une liste de pointeurs d'accès conservée elle aussi dans cet espace mémoire.

Cette modification peut apporter un gain de 10 à 25 pour cent pour des expériences réelles, respectant les hypothèses précédemment énoncées.

CHAPITRE 2
DEVELOPPEMENT SUR UNE MACHINE MULTIPLE

Les temps CPU nécessaires pour la simulation d'un grand nombre d'évènements sont, pour des géométries réalistes, souvent importants. Pour obtenir les résultats dans un temps raisonnable, il convient d'opter pour l'utilisation de calculateurs plus performants. L'association de plusieurs calculateurs permet théoriquement de diviser le temps d'exécution. Si les communications entre machines sont faibles, les gains pourront être proportionnels au nombre de calculateurs utilisés.

2.1 Particularités de GEANT3 liées aux Multi – Calculateurs

L'implémentation de GEANT3 sur plusieurs machines travaillant en parallèle, reliées par un réseau de communication devra répondre quelque soit le type d'architecture, à un ensemble de règles précises imposées par l'algorithme.

2.1.1 Initialisation des structures de données statiques

Bien évidemment, chaque élément du Multi – Calculateurs (MC) devra utiliser le même environnement expérimental. Celui – ci n'étant utilisé qu'en lecture, il n'y aura pas de problème de mise à jour des mêmes structures sur plusieurs calculateurs. Deux solutions sont possibles pour construire cet environnement :

- soit par construction sur chaque calculateur des mêmes structures définies par l'utilisateur. Cette solution ne permet pas de relire l'information déjà construite ultérieurement depuis un disque comme le propose GEANT3, mais par contre elle permet d'associer des machines différentes n'utilisant pas le même codage de l'information. En pratique, il est peut probable de définir de telles architectures bien que souvent des réseaux d'ordinateurs non utilisés pendant la nuit, pourraient servir pour générer une production massive d'évènements.

- soit par construction unique et diffusion des structures: un calculateur privilégié possédant un système d'entrées/sorties, est chargé de construire l'environnement puis ensuite de le diffuser à tous les autres éléments du MC. A la différence de la solution précédente, nous pourrons ici utiliser le stockage sur disque des informations. Le temps nécessaire pour le transfert vers les autres calculateurs s'ajoute au temps d'initialisation.

2.1.2 Restructuration des résultats

Les résultats produits par des méthodes Montè Carlo sont toujours des résultats statistiques. Si l'exécution est partagée entre plusieurs calculateurs, les statistiques produites sur chacun d'entre eux seront moins représentatives que celles produites par la même simulation sur un seul calculateur. Il faut donc récolter les résultats sur chaque élément puis les réunir. Les résultats sont en fait des histogrammes, il suffit de sommer ceux – ci sur un des calculateurs après leur transmission. Le temps nécessaire pour ce travail est proportionnel au nombre d'éléments du MC, l'acheminement ne peut être fait que de façon séquentielle élément par élément. Une reconstruction partielle pourrait être assurée pour des sous – ensembles de calculateurs, cela permettrait de minimiser le nombre de transferts et le temps utile pour la reconstruction totale.

2.1.3 Accès à un fichier commun

Après chaque simulation d'un évènement, l'utilisateur peut demander à sauvegarder les structures produites avant la remise à zéro pour la simulation d'un autre évènement. Ces structures sont alors rangées dans un fichier. Il faut donc qu'un calculateur puisse récupérer l'information sur les autres après chaque évènement simulé ou puisse reconstruire les structures évènement par évènement en fin de simulation.

2.1.4 Conséquence pour l'architecture

De ces trois premières remarques, nous pouvons conclure que l'un des calculateurs devra être privilégié et centraliser toutes les requêtes d'entrées/sorties en début, en cours et en fin de simulation. Ainsi tous les autres calculateurs du MC peuvent être réduits à des unités de calcul munies d'une mémoire locale, seule une interface vers un réseau de communication est en plus nécessaire. Nous avons vu que cette solution augmenterait le temps d'initialisation mais, étant indépendant du nombre d'évènements simulés, cela n'interviendra pas de façon critique sur les performances.

2.1.5 Génération de nombres aléatoires

Pour conserver l'intérêt des méthodes Montè Carlo sur un MC, il faut s'assurer que deux calculateurs n'utilisent pas la même suite de nombres aléatoires. Sinon les résultats statistiques seraient faux : les mêmes actions seraient exécutées et produiraient les mêmes phénomènes physiques.

La solution idéale pour éviter ce genre de surprise, serait que chaque calculateur utilise une partie distincte de la suite des nombres générés. Il faut pouvoir accéder directement au Iième nombre aléatoire sans avoir à connaître les I – 1 premiers. Une solution existe pour les générateurs de type

$$X_n = a\, X_{n-1}$$

2.1.5.1 Définition

Soit le générateur sur une machine de mots de p bits

$$X_n = a\, X_{n-1} \bmod 2^p$$

Montrons par récurrence que

$$X_n = A^n\, X_0 \bmod 2^p$$

pour n = 1 on a bien

$$X_1 = A\, X_0 \bmod 2^p$$

supposons la relation vraie pour n alors

$$X_{n+1} = A\, X_n \bmod 2^p$$

$$X_{n+1} = A\, (\, A^n\, X_0 \bmod 2^p\,) \bmod 2^p$$

– propriétés des modulos

(1) (ab) mod c = (a mod c * b mod c) mod c
(2) (ab) mod c = (a * b mod c) mod c
(3) (ab) mod c = (a mod c * b) mod c

d'où par application de (3) et (1)

$$X_{n+1} = A^{n+1}\, X_0 \bmod 2^p$$

Par l'application de (3) on obtient

$$X_n = (A^n \bmod 2^p * X_0) \bmod 2^p$$

la seule partie dépendant de n est indépendante de X_0 soit

$$A^n \bmod 2^p$$

or

$$n = n_0 + 2n_1 + 2^2 n_2 + 2^3 n_3 + \ldots\ldots + 2^{p-1} n_{p-1}$$

avec

$$n_i = 1 / 0$$

donc

$$A^n = A^{(n_0 + 2n_1 + 2^2 n_2 + 2^3 n_3 + \ldots\ldots + 2^{p-1} n_{p-1})}$$

$$A^n = A^{n_0} * A^{2n_1} * A^{2^2 n_2} * \ldots\ldots * A^{2^{p-1} n_{p-1}}$$

d'où

$$A^n \bmod 2^p = (A^{n_0} * A^{2n_1} * A^{2^2 n_2} * \ldots\ldots * A^{2^{p-1} n_{p-1}}) \bmod 2^p$$

par généralisation de (1)

$$A^n \bmod 2^p = ((A^{n_0} \bmod 2^p) * (A^{2n_1} \bmod 2^p) * .. * (A^{2^{p-1} n_{p-1}} \bmod 2^p)) \bmod 2^p$$

or

$$A^{2^i n_i} = (A^{n_i})^{2^i} \qquad \text{avec} \quad A^{n_i} = A \text{ si } n_i = 1 \\ = 1 \text{ si } n_i = 0$$

la formule finale est

$$A^n \bmod 2^p = (d_0(A \bmod 2^p) * d_1(A \bmod 2^p) * .. * d_{p-1}(A \bmod 2^p)) \bmod 2^p$$

avec

$d_i(x) = x$ si $n_i = 1$

$= 1$ si $n_i = 0$ soit $d_i(x) = n_i(x-1) + 1$

Pour connaître le Nième nombre aléatoire, il suffit de calculer la décomposition binaire de n, le reste étant défini une fois pour toutes en fonction du choix du générateur.

2.1.5.2 Exemple d'application

Pour A = 69069 et p = 32.

```
      FUNCTION NGRND(NX0,I)
      DIMENSION IT(32)
      DATA IT/32* 0 /
      IF (IT(1).NE.0) GO TO 6
c     initialisation au premier appel des puissances de A
        IA=69069
        DO 5 I=1,32
          IF (IA.LT.0) IA=IA+2147483647+1
          IT(I)=IA
          IA=IA*IA
    5   CONTINUE
    6 N=I
      NB=1
      DO 100 J=1,32
        IF (N.EQ.0) GO TO 110
        NR=(N/2)*2
        IF (NR.NE.N) NB=NB*IT(J)
        N=N/2
  100 CONTINUE
  110 NB=NB*NX0
      IF (NB.LT.0)NB=NB+2147483647+1
      NGRND=NB
      END
```

2.2 Multi – Calculateurs de type maître – esclaves

2.2.1 Présentation de l'architecture

Plusieurs calculateurs sont liés par un système de communication, l'un d'entre eux est privilégié, il est chargé de contrôler ce système. Il sera appelé Machine Maître (MM) les autres sont appelés Machine Esclave (ME).(fig 26)

Un seul transfert d'information entre maître et esclave ou esclave et maître est possible à un instant donné. La communication directe de ME à ME doit obligatoirement passer par la MM.

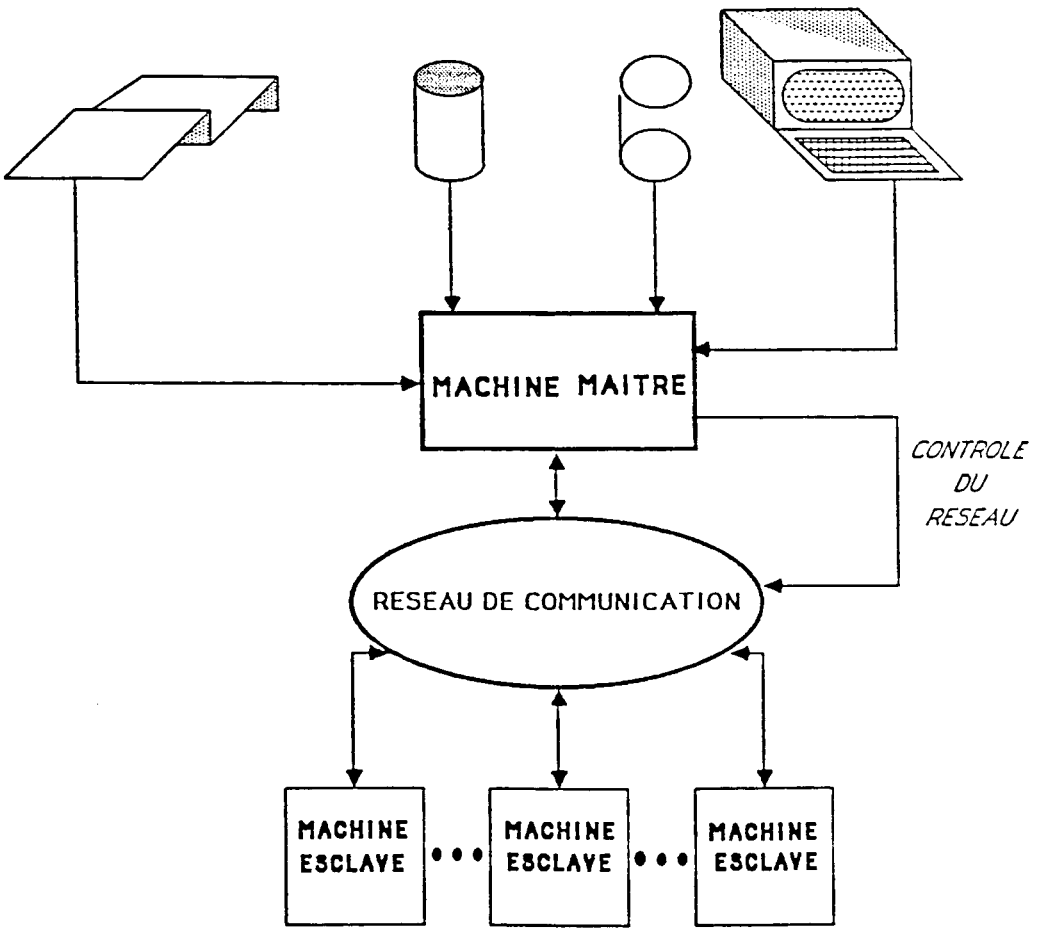


Figure 26: Machine maître – esclaves

BU LILLE

La MM est chargée de scruter les différentes ME qui ne peuvent interrompre la MM pour satisfaire leurs requêtes. Une ME est en général stoppée jusqu'à prise en compte de sa requête.

L'intérêt d'un tel système est sa simplicité de réalisation. Les ME sont en fait chargées de n'exécuter que des tâches de productions, c'est la MM qui contrôle l'ensemble par l'envoi de tâches et la récupération de résultats. Ce type de machine est souvent utilisé pour de gros calculs scientifiques. Cette architecture répond parfaitement aux règles précédemment énoncées, il suffit de munir la MM d'un système d'entrées/sorties.

2.2.2 Parallélisation de type multi – évènements

Le système de gestion des communications étant élémentaire, il convient de limiter au maximum les transferts. Nous avons vu que GEANT3 permet à l'utilisateur d'intervenir après chaque simulation d'évènement pour contrôler la sauvegarde des informations produites. La simulation d'un évènement est donc le processus le plus long qui ne demande aucun transfert de données.

Chaque ME pourra exécuter un évènement complet sans avoir à s'arrêter. Si l'utilisateur ne désire pas intervenir pour exécuter des lectures ou écritures, c'est la totalité des évènements qui pourront être simulés d'un seul coup.

- *Algorithme général:*

L'initialisation de l'environnement est assuré sur la MM comme pour une exécution séquentielle, les structures construites ou lues à partir d'un disque sont envoyées vers les ME. Suivant le type de connexion, le chargement pourra ou non être effectué simultanément sur toutes les ME. La simulation proprement dite peut alors commencer. Deux gestions différentes des évènements sont possibles

- soit par l'envoi d'un évènement sur chaque ME, celui – ci est alors simulé suivant sa propre série de nombres aléatoires, puis la ME s'arrête pour permettre à la MM de récupérer les informations produites, si des évènements n'ont pas encore été simulés la MM en enverra un d'entre eux vers la ME. La charge en temps CPU par ME est alors bien équilibrée, la différence de temps CPU total entre deux ME ne peut être supérieure au temps CPU nécessaire pour simuler un évènement.

- soit par l'envoi d'un paquet d'évènements sur chaque ME, chacun recevant un nombre semblable d'évènements (à ± 1 près). Chaque requête de lecture provoque l'arrêt de la ME jusqu'à sastifaction. Une requête d'écriture provoque elle, la mise en buffer de l'information dans la mémoire de la ME. Lorsque le buffer est plein la ME s'arrête pour le transférer vers la MM. La taille du buffer pouvant être égale à celle de la taille mémoire non utilisée, cela permet de minimiser le nombre de transferts. Ce type de développement s'adapte très bien à la production massive de statistiques demandant beaucoup d'écritures et peu de lectures. Si chaque évènement demande un temps d'exécution à peu près identique, la différence en temps CPU pour chaque ME sera faible mais pour certaines applications ou le temps de simulation d'un évènement varie beaucoup, elle peut être importante.

Dans les deux cas, lorsque tous les évènements ont été simulés, il faut reconstruire les résultats finaux depuis chaque ME. Les transferts étant exécutés séquentiellement, le déséquilibrage entre deux ME peut être en partie amorti.

Aucun transfert entre deux ME n'est nécessaire, cet algorithme peut donc parfaitement s'adapter à une architecture de type maître – esclaves comme décrite en 2.2.1.

2.3 Développement sur une batterie d'émulateurs 3081/E

Une équipe du CERN en collaboration avec le laboratoire de Standford a construit des émulateurs d'IBM 370. Ces émulateurs ont été assemblés en réseaux de type Maître – Esclaves. Cette machine servit de support pour appliquer le parallélisme multi – évènements sur GEANT3.

2.3.1 Présentation technique de la machine

L'ensemble de la machine est composé d'une machine maître munie d'un système d'entrées/sorties, d'un ensemble d'émulateurs 3081E [15] et d'un bus de connexion reliant chaque élément. (fig 27)

Les processeurs 3081/E sont construits suivant une structure modulaire. Cela permet de définir une machine plus simple, fiable et facile à maintenir et à mettre au point. Il y a quatre unités de calculs interfacées à deux bus de 64 bits, le ABUS et le BBUS. A ces bus sont aussi interfacées une unité de contrôle et de registres, une mémoire et une interface vers le réseau de communication extérieur. L'unité de contrôle et de registres sert à contenir le compteur d'adresse de micro – programme, assurer la logique de branchement, à calculer les adresses mémoire et à gérer les registres. Une micro – instruction peut transférer simultanément deux opérandes sur le ABUS et le BBUS depuis la mémoire ou depuis les registres vers une unité de calcul. Le résultat produit par une unité de calcul est transféré sur le BBUS vers un registre ou vers la mémoire, il peut aussi être envoyé directement vers une autre unité par le ABUS, il devient alors une des opérandes d'une nouvelle instruction : l'enchaînement d'opérations est ainsi possible. Un bus de 32 bits permet de charger les instructions depuis la mémoire. Une horloge unique de cycle 120 ns contrôle l'ensemble. (fig 28)

La mémoire de chaque processeur possède un cycle de scrutation de 55 ns. Un espace mémoire de 7 MO peut être défini; il est prévu dans l'avenir d'utiliser des mémoires statiques de 64 K au lieu de 16 K, et ainsi obtenir un espace mémoire de 28 MO.

Plusieurs types de pipe – lines ont été développés :

- Calcul de l'adresse dans l'unité de contrôle.
- Pipe – lines internes des unités de calcul addition/soustraction et multiplication permettent de traiter un nouveau jeu de données à chaque cycle machine et de produire un résultat à chaque cycle après un temps de démarrage de 2 cycles.

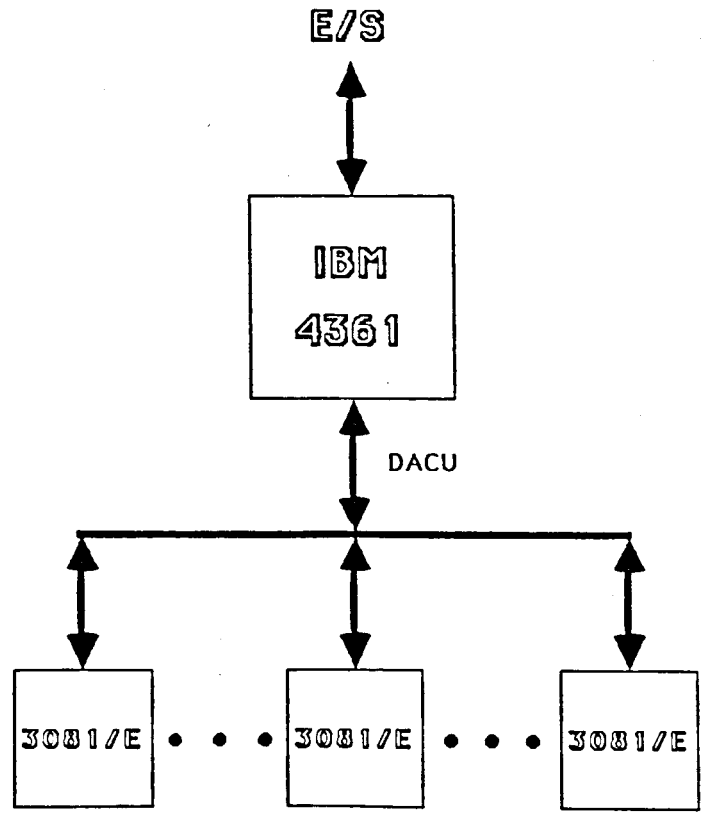


Figure 27: Emulateurs 3081E

BU LILLE

- Dans le même cycle, des résultats peuvent être envoyés d'une unité de calcul et récupérés par la mémoire ou l'unité de contrôle et registre.

Le transfert des données depuis ou vers un processeur peut atteindre 64 MO/s. L'interface peut être connectée à plusieurs types de réseaux externes. Une liaison spécifique doit être définie pour chaque type.

Les programmes exécutables sur les 3081E sont dérivés des programmes IBM. Un translateur est chargé de modifier le code objet pour les quelques instructions non reconnues par les 3081E. Ainsi les logiciels IBM tel que le compilateur FORTRAN sont utilisables pour les émulateurs et n'ont pas dû être réécrits.

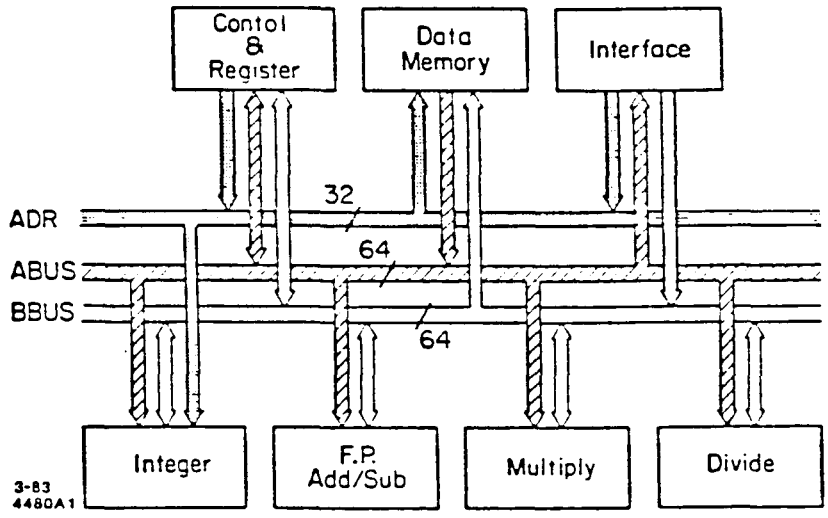


Figure 28: Architecture interne d'un 3081/E

Au CERN, la MM est un 4361 IBM, pouvant contrôler jusqu'à 5 émulateurs. Le bus extérieur reliant les 3081/E au 4361 est un canal géré par l'interface du système VME, il accepte un débit de 3 MO/s.

Le coût de cette machine est de l'ordre de 10000 $US par processeur et de 5000 $US par MO de mémoire. Les performances obtenues dépendent de l'application et des possibilités d'utilisation des pipe – lines. Pour un émulateur, elles doivent néanmoins être supérieures à celles d'un IBM 370/168. Le rapport prix/performance a été l'un des facteurs décisifs pour la réalisation de ces machines.

2.3.2 Procédures de communication

Les communications sont gérées à partir de la MM. Elles sont exécutables à partir de programmes FORTRAN par des appels à des procédures prédéfinies. Quatre d'entre elles sont essentielles :

- *SYEDWN:* elle permet de charger la mémoire des émulateurs à partir d'un fichier disque, les programmes préalablement transcrits en micro – code sont envoyés vers un ou plusieurs émulateurs.

- *SYEPUT:* elle permet d'envoyer l'information contenue dans un bloc mémoire de la MM vers un bloc mémoire d'une ou toutes les ME. En fonction de l'option choisie, le transfert peut être suivi du lancement d'un programme préalablement chargé, ou d'une reprise de l'activité à l'endroit où elle a été interrompue.

- *SYEGET:* elle permet de récupérer l'information contenue dans un bloc mémoire d'une ME vers la MM. Suivant l'option choisie, cela peut être fait n'importe quand, même si la ME est active ou seulement quand elle est stoppée. Si la deuxième option est choisie, la MM attendra qu'une ME quelconque ou identifiée s'arrête.

- *SYEOPT:* elle permet diverses choses telles que calcul du temps d'exécution sur chaque ME, réservation d'émulateurs. . .

Une seule procédure est disponible à partir d'une ME, la procédure BREAK. Elle permet d'arrêter l'émulateur qui l'exécute, dans des conditions autorisant un redémarrage.

2.3.3 Entrées/sorties.

Sur les émulateurs, les ordres FORTRAN *READ/WRITE* ne peuvent être interprétés puisque aucun système d'entrées/sorties n'est défini. Aussi le translateur doit – il remplacer ces ordres par des instructions nulles NOP. Pour GEANT3 certains de ces ordres de lecture ou écriture sont nécessaires et ne peuvent être supprimés : c'est le cas pour la sauvegarde ou la reconstruction de structures de données par l'intermédiaire de procédures de ZEBRA. Il faut donc mettre en place un protocole de communication entre le programme exécuté sur un émulateur et celui exécuté sur la MM. En effet, seule la MM possède un système d'entrées/sorties et peut donc réaliser effectivement les écritures et lectures d'informations. Pour cela, un buffer en entrée et un autre en sortie sont nécessaires sur chaque émulateur. Les ordres *READ* et *WRITE* FORTRAN sont remplacés par des ordres de vidage et de remplissage d'un des deux buffer. Lorsqu'un buffer en lecture est vide ou lorsqu'un buffer en écriture est plein, il faut établir une communication avec la MM pour remplir depuis le disque ou vider sur le disque le buffer de l'émulateur concerné.

2.3.3.1 Buffer en entrée.

Ne sachant pas à l'avance quelles informations seront lues, il faut pour chaque ordre de lecture interrompre la ME, envoyer une requète à la MM qui effectuera la lecture proprement dite, puis transférera le buffer ainsi rempli vers l'émulateur demandeur. La taille de ce buffer n'a pas besoin d'être supérieure à celle d'un enregistrement du fichier.

L'anticipation en lecture n'est pas possible ne sachant pas à l'avance quel type d'information désirera lire l'utilisateur.

Si trop de demandes de lectures étaient faites, les performances risqueraient de chuter rapidement: une structure de données peut être représentée par un grand nombre d'enregistrements et donc provoquer beaucoup d'interruptions.

2.3.3.2 Buffer en sortie.

Chaque ordre d'écriture provoque la mise en buffer de l'information. Lorsque le buffer est plein celui – ci est envoyé vers la MM qui effectue réellement l'écriture sur disque. Quand le buffer a été envoyé, l'émulateur producteur peut tout de suite reprendre son activité. De la taille du buffer dépend le nombre de transfert de ME vers MM. En fonction de l'espace mémoire disponible sur les 3081E, il est judicieux de choisir la taille maximale pour ce buffer. (fig 29)

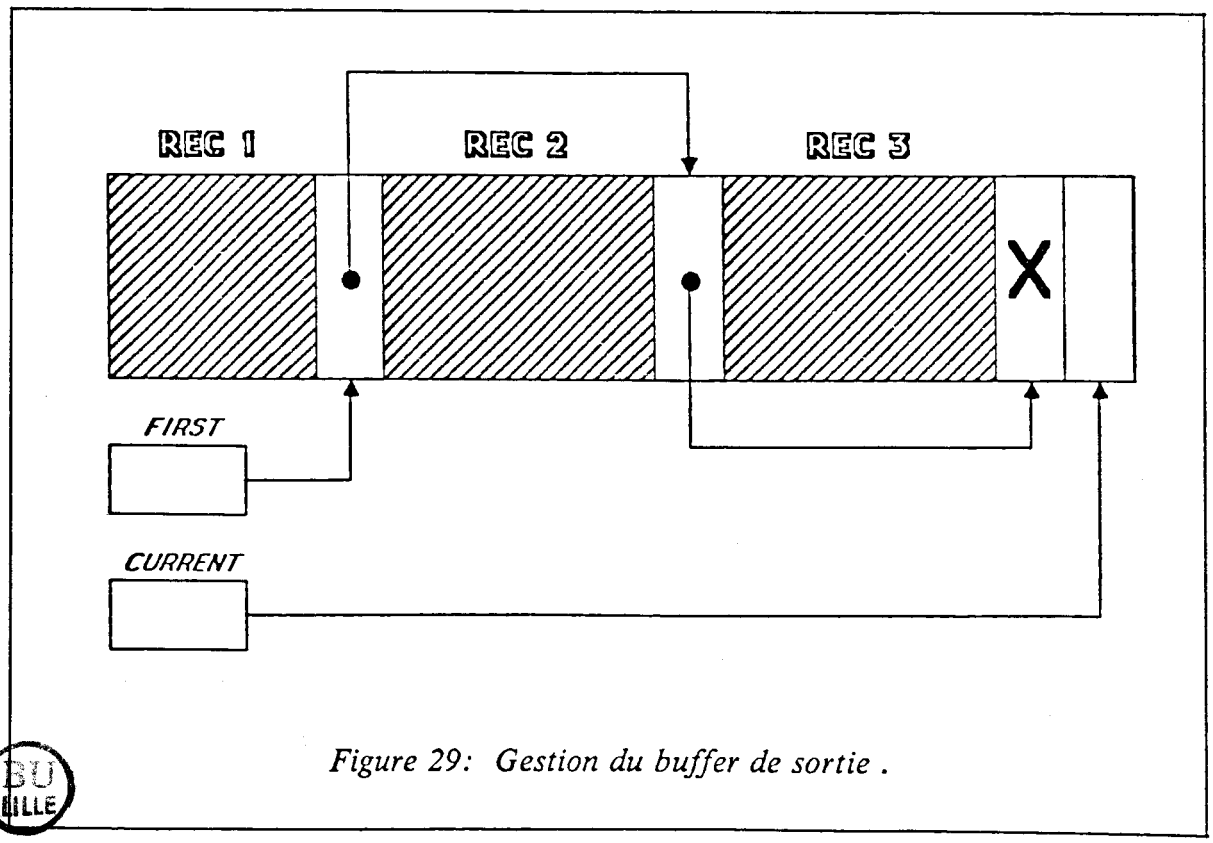


Figure 29: Gestion du buffer de sortie .

BU LILLE

2.3.3.3 Partage des fichiers d'entrées/sorties

Afin d'éviter le mélange d'informations d'évènements différents, il faut pour chaque accès à un fichier par l'intermédiaire de GGET et GSAVE, (procédures de

GEANT3 chargées de la lecture et de l'écriture des structures de données) assurer l'exclusion mutuelle entre deux émulateurs quelconques désirant intervenir sur ce fichier. Les procédures GGET et GSAVE doivent en début d'exécution, réserver le fichier et attendre qu'il soit disponible, une fois le travail terminé, elles doivent le libérer. La MM gére ces partages de fichiers, étant elle – même une ressource partagée en mutuelle exclusion: aucune ME ne pouvant l'interrompre. Un protocole de communication est utilisé entre MM et ME. Une ME prépare un message dans sa mémoire, et appelle BREAK. Rappelons qu'elle sera relancée éventuellement plus tard par la MM lorsque celle – ci aura traité le message.

Plusieurs types de messages réservés ont été définis :

- *FIN:* permet d'indiquer à partir d'une ME que la simulation est terminée.
- *DEB_SC(LU):* demande d'accès à l'unité logique LU en temps que ressource partagée.
- *FIN_SC(LU):* libération d'une unité logique LU en fin de réservation.
- *ES_OUT(LU,LG):* demande de transfert pour écriture d'une partie du buffer de sortie de longueur LG sur l'unité logique LU préalablement réservée par cette ME.
- *ES_IN(LU,LG):* demande de lecture dans le buffer d'entrée et de le transférer ensuite vers la ME.
- *FOUT_SC(LU,LG):* libération d'une unité logique LU après vidage d'une partie du buffer en écriture de longueur LG.

Description des algorithmes de communication :

```
MACHINE MAITRE
==============
 .
 .
 SYEDWN ( ..... )                    % chargement des programmes   %
 STOPR = 0
 SYEPUT ( GO , ALL , INIT )          % structure sur tous les ME   %
 repeter
 SYEGET ( FIRST , MESS , IPROC )     % le premier 3081 arrete (IPROC)%
 cas  MESS.TYPE de
   FIN : STOPR := STOPR + 1
   DEB_SC : debut
            si STATUS(LU) = LIBRE alors
              SYEPUT ( CONT ,IPROC )   % relance de l'emulateur IPROC %
              STATUS(LU) := OCCUPE
            sinon
```

```
            METTRE_EN_FILE ( IPROC , LU ) % FIFO associee a l'unite  LU %
          fsi
          fin
FIN_SC  : debut
          SYEPUT ( CONT , IPROC )
          si FILE (LU) = VIDE alors
            STATUS(LU) = LIBRE
          sinon
            PRENDRE_EN_FILE ( IPR , LU )
            SYEPUT ( CONT , IPR )        % debloquer un emulateur %
          fsi
          fin
ES_OUT  : debut
          SYEGET ( IRPOC , BUF_OUT, LG ) % transfert du buffer    %
          SYEPUT ( CONT , IPROC )
          ECRIRE ( LU , LG , BUF_OUT )   % ecriture sur disque    %
          FIN
ES_IN   : debut
          LIRE ( LU , LG , BUF_IN )      % lecture sur disque     %
          SYEPUT ( CONT , IPROC , BUF_IN ) % transfert du buffer  %
          fin
FOUT_SC : debut                            % = ES_OUT + FIN_SC    %
          SYEGET ( IRPOC , BUF_OUT , LG )
          SYEPUT ( CONT , IPROC )
          si FILE(LU) = VIDE alors
            STATUS(LU) = LIBRE
          sinon
            PRENDRE_EN_FILE ( LU , IPR )
            SYEPUT ( CONT , IPR )
          fsi
          ECRIRE ( LU , LG , BUF_OUT )
          FIN
fcas
jusque STOPR = NB_EMULATEUR
.
.


MACHINE ESCLAVE EN LECTURE : GGET
=================================

GGET ( LU , STRUCTURE ... )
.
.
MESS=DEB_EC ( LU )
call BREAK
tantque NON_FIN ( STRUCTURE ) faire
```

```
  .
  MESS = ES_IN ( LU, LG )
  call BREAK
  % lecture dans le buffer d'entree BUF_IN %
  .
  .
ftq
MESS = FIN_EC ( LU )
call break
.
MACHINE ESCLAVE EN ECRITURE : GSAVE
==================================

GSAVE ( LU , STRUCTURE ... )
.
.
PREM = 0
si LU <> LU_OLD et NON_VIDE ( BUF_OUT ) alors
  MESS = DEB_SC ( LU_OLD )            % vidage du buffer associe a %
  call BREAK                          % l'ancienne unite logique   %
  MESS = FOUT_SC ( LU_OLD , LG )
  call BREAK
  RAZ ( BUF_OUT )
  LU_OLD = LU
fsi
.
tantque NON_FIN ( STRUCTURE ) faire
  .
  % ecriture dans le buffer de sortie BUF_OUT %
  si PLEIN ( BUF_OUT ) alors      % vidage du buffer %
    si PREM = 0 alors
      PREM = 1                    % premier vidage pour cet event %
      MESS = DEB_SC ( LU )
      call BREAK
    fsi
    MESS = ES_OUT ( LU , LG )
    call BREAK
    RAZ ( BUF_OUT )
  fsi
  .
ftq
si PREM = 1 alors                 % vidage du buffer     %
  MESS = FOUT_SC ( LU , LG )      % pour assurer la continuite %
  call BREAK                      % de l'info par event  %
  RAZ ( BUF_OUT )
fsi
.
```

La réservation d'unité logique est gérée par un système du type sémaphores. La demande de réservation correspond à l'appel de la procédure P(mutex(LU)), la libération à l'appel de V(mutex(LU)); mutex étant un tableau de sémaphores de mutuelle exclusion initialisés à un.

2.3.4 Performances

Liste des notations utilisées

- *N:* nombre d'évènements
- *Tinit:* temps cpu pour la phase d'initialisation
- *Tlast:* temps cpu pour la phase de terminaison
- *T4361:* temps cpu pour simuler un évènement sur le 4361
- *T3081:* temps cpu pour simuler un évènement sur un 3081E
- *P:* nombre d'émulateurs
- *Q:* taille des données à transférer en début
- *q:* taille des données à transférer en fin
- *d:* débit de la communication
- *Bo:* taille du buffer de sortie
- *Rm:* taille moyenne d'un enregistrement
- *Nro:* nombre moyen d'enregistrements en sortie par évènement
- *Nri:* nombre moyen d'enregistrements en entrée par évènement
- *sc:* temps de start – up pour une communication
- *des:* débit en entrées/sorties

On peut définir une formule de calcul de temps d'exécution séquentielle :

$$Ts = Tinit + Trun + Tlast$$

avec

$$Trun = N * (T4361 + Tecr)$$

avec

$$Tecr = (Nri + Nro) * Rm / des$$

Pour une exécution parallèle on obtient :

$$Tp = Tinit + Tload + Trun + Tfetch + Tlast$$

avec

$$Tload = Q / d + sc$$ chargement en parallèle

ou

$$Tload = (Q / d + sc) * P$$ chargement en série

$$Tfetch = (q / d + sc) * P$$

$$Trun = N / P * (T3081 + Tcom)$$

avec

$$Tcom = (Nri + Nro) * Rm / d + sc * ((Nro * Rm) \text{ Div } Bo + 1 + Nri)$$

De cette deuxième formule, il apparaît que la taille du buffer Bo n'a d'importance que dans le cas où le temps de start – up n'est pas négligeable.

La définition théorique du nombre maximal d'émulateurs sans saturer le système de communication est :

$$Pmax * (Tcom + Tecr) = T3081 + Tcom$$

donc pour $Tcom + Tecr \neq 0$

$$Pmax = (T3081 + Tcom) \text{ Div } (Tcom + Tecr)$$

si $Tcom + Tecr = 0$ alors

$$Pmax = \infty$$

- *Résultats pratiques:*

Deux essais différents ont été faits, avec GEXAM1 exemple proposé par GEANT3 et avec GOPAL, l'application de GEANT3 pour OPAL.

Les résultats obtenus pour l'exemple 1 ont permis de comparer les performances des émulateurs sans utiliser les entrées/sorties.

	1 emulateur	2 emulateurs	3 emulateurs
10 ev	22.9 sec	12.6 sec	11.2 sec
100 ev	22.2 sec	11.2 sec	7.6 sec

Temps d'exécution par évènement avec chargement en série.

Si le nombre d'évènements est assez grand, pour une simulation sans entrées/sorties, le temps d'exécution d'un évènement est inversement proportionnel au nombre d'émulateurs.

La configuration utilisée ne comprenant que trois émulateurs, il n'a pas été possible de faire apparaître une baisse des performances pour un exemple avec entrées/sorties.

2.3.5 Conclusion sur les machines Maître – Esclaves

Ce type d'algorithme peut être développé sur d'autres machines, en particulier, nous pensons l'appliquer sur un réseau d'Apollo, l'un d'entre eux serait chargé de gérer les entrées/sorties. Le projet de développer GEANT3 sur les machines ACP [17] en développement à FERMILAB pourrait aussi avoir recours aux mêmes solutions algorithmiques. La parallélisation multi – évènements s'adapte en fait sur toutes les machines possédant au moins un système de communication élémentaire.

2.4 GEANT3 en tant qu'algorithme à pile

L'algorithme général de GEANT3 présenté au chapitre 1 a montré que la simulation d'un évènement conduisait à générer des particules mises en pile en vue de leur traitement ultérieur. L'indépendance entre ces traitements permet d'envisager une approche de parallélisme au niveau des particules et non au niveau des évènements. Il faut alors partager la pile de particules entre plusieurs calculateurs. La pile est d'ailleurs un système de gestion d'information trop puissant puisque l'ordre dans lequel seront suivies les particules n'est pas important. Ce type de solution peut aussi s'appliquer à tout autre algorithme à pile, où la pile sert de stockage temporaire pour des tâches secondaires indépendantes. Le système de communication devra être plus performant que précédemment. Une architecture adaptée sera proposée en 2.4.2.2

2.4.1 Pile partagée, pile répartie

Lorsqu'une particule secondaire est générée, au lieu de l'empiler sur la pile locale au calculateur, elle est envoyée vers une pile générale partagée entre tous les calculateurs. Quand un calculateur termine la simulation d'une particule, c'est de cette pile générale que sera extraite la particule suivante. (fig 30)

La pile générale devient une ressource partagée entre tous les calculateurs, son accès doit être fait au travers d'une section critique. L'utilisation des vecteurs sémaphores présentés en appendice, empêchera la libération de la pile par un calculateur qui ne l'aurait pas réservée. Il est alors impossible de forcer l'accès à la pile générale.

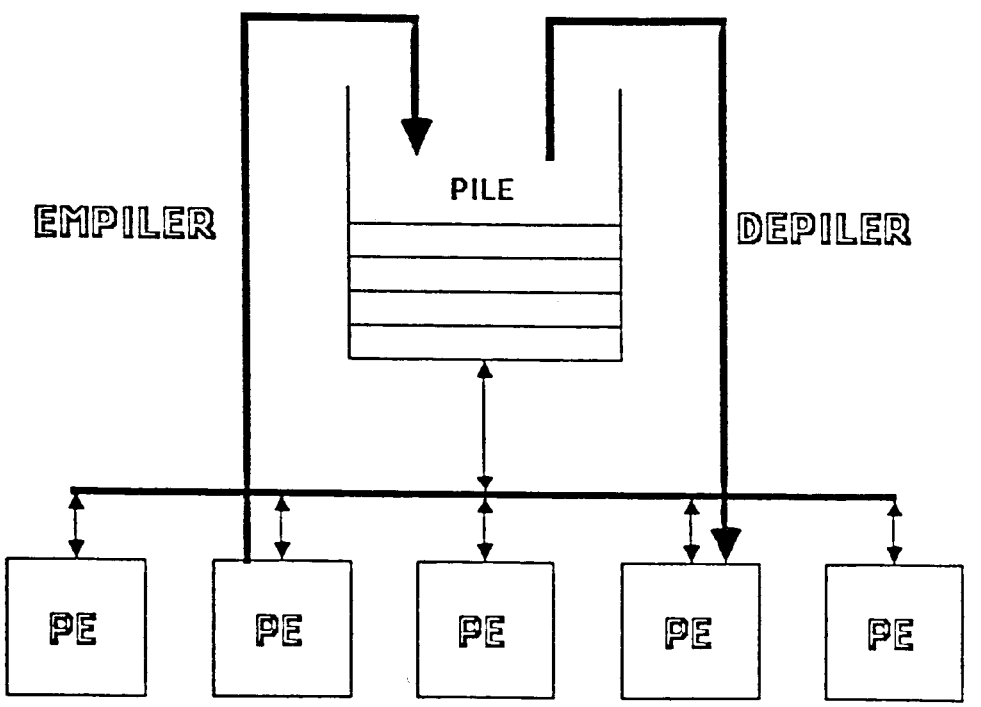


Figure 30: Pile partagée

Afin de diminuer le nombre de transferts d'un calculateur vers la pile partagée, une pile locale à chaque calculateur peut être utilisée pour le stockage de quelques particules générées par celui – ci. Lorsque la pile locale est pleine, une partie est transférée vers la pile générale. Lorsqu'elle est vide une partie de la pile générale est transférée vers la pile locale. (fig 31)

Les algorithmes empiler et dépiler de chaque calculateur deviennent :

```
procedure EMPILER ( PART )
debut
EMPILER_BLOC ( PART , 1 )
si PLEIN ( PILE_LOCALE ) alors
    DEPILER_BLOC ( MESS_PART , N )
    MESS.TYPE = EMPILER
    MESS.SIZE = N
```

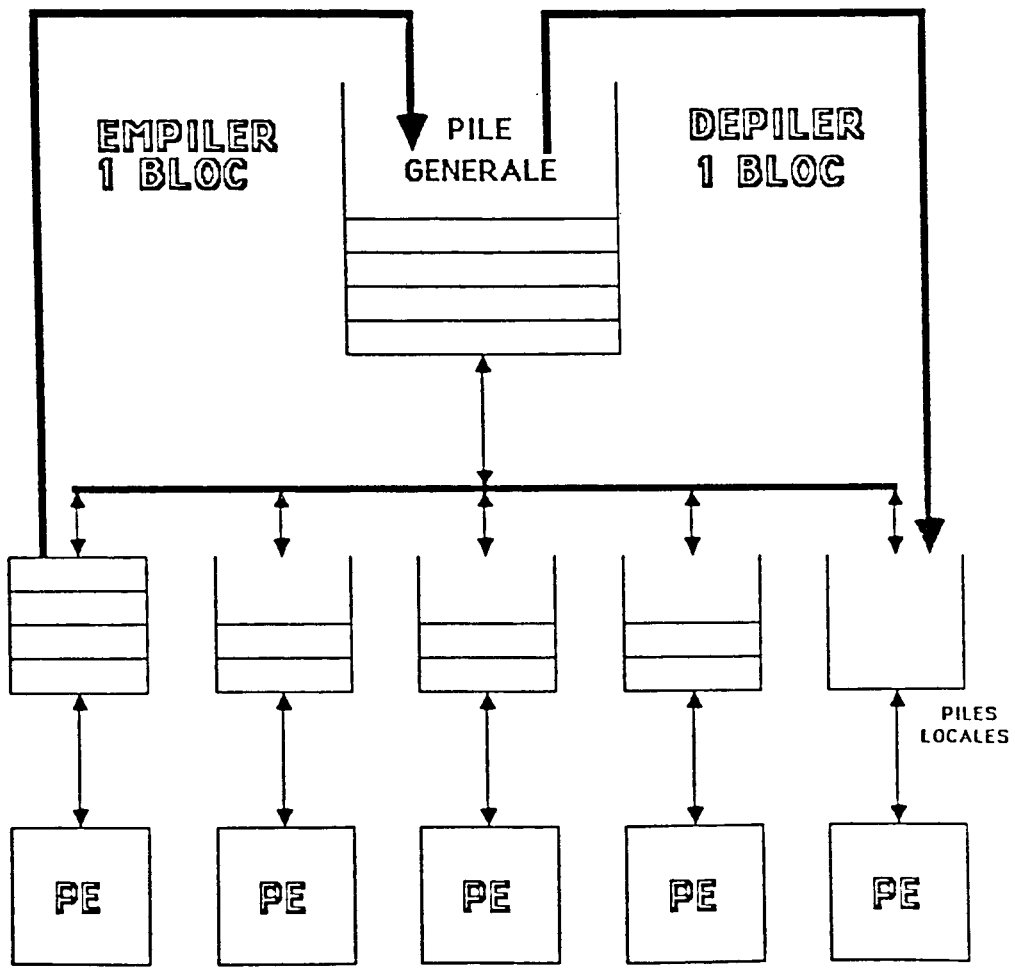


Figure 31: Pile répartie

BU LILLE

```
    ENVOYER ( MESS )
fsi
fin

procedure DEPILER ( PART )
debut
si VIDE ( PILE_LOCALE ) alors
    MESS.TYPE = DEPILER
    MESS.SIZE = N
    MESS.DEST = NUM_EMULATEUR
```

```
    ENVOYER ( MESS )
    ATTENDRE ( PART_BLOC , M )
    EMPILER_BLOC ( PART_BLOC , M )
fsi
si VIDE (PILE_LOCALE ) alors
    FIN_EVENEMENT
sinon
    DEPILER-BLOC ( PART , 1 )
fsi
fin
```

Algorithme de gestion de la pile générale :

```
tantque FIFO.TAILLE <> NB_EMULATEUR faire
    WAIT ( MESS )
    cas MESS.TYPE OU
    DEPILER :
        debut
        M = MIN ( MESS.SIZE , PILE.SIZE )
        si M > 0 alors
            DEPILER_BLOC ( PART , M )
            SEND ( MESS.DEST , PART , M )
        sinon
            METTRE_FIFO ( MESS.DEST , MESS.SIZE )
        fsi
        fin
    EMPILER :
        debut
        si PILE.SIZE > 0 alors
            EMPILER_BLOC ( MESS.PART , MESS.SIZE )
        sinon
            M = MESS.SIZE
            PART = MESS.PART
            tantque FIFO.TAILLE > 0 faire
                PRENDRE_FIFO ( DEST , SIZE )
                si SIZE >= M alors
                    SEND ( DEST , PART , M )
                    M = 0
                sinon
                    SEND ( DEST , PART , SIZE )
                    PART ( 1..M-SIZE ) = PART ( SIZE+1..M )
                    M = M - SIZE
                fsi
            ftq
        si M > 0 alors
            EMPILER_BLOC ( PART , M )
```

```
            fsi
            fin
        fcas
    ftq
    pour I de 1 a NB_EMULATEUR faire
        SEND ( I , 0 , 0 )
    fait
    fin
```

2.4.2 Algorithme à pile et architecture

2.4.2.1 Machine séquentielle

Les particules secondaires générées lors du suivi d'une autre particule ne peuvent pas être tout de suite simulées sur une machine séquentielle, elles sont empilées temporairement. L'une après l'autre dans un ordre imposé par le gestionnaire de pile, elles seront en suite suivies. (fig 32)

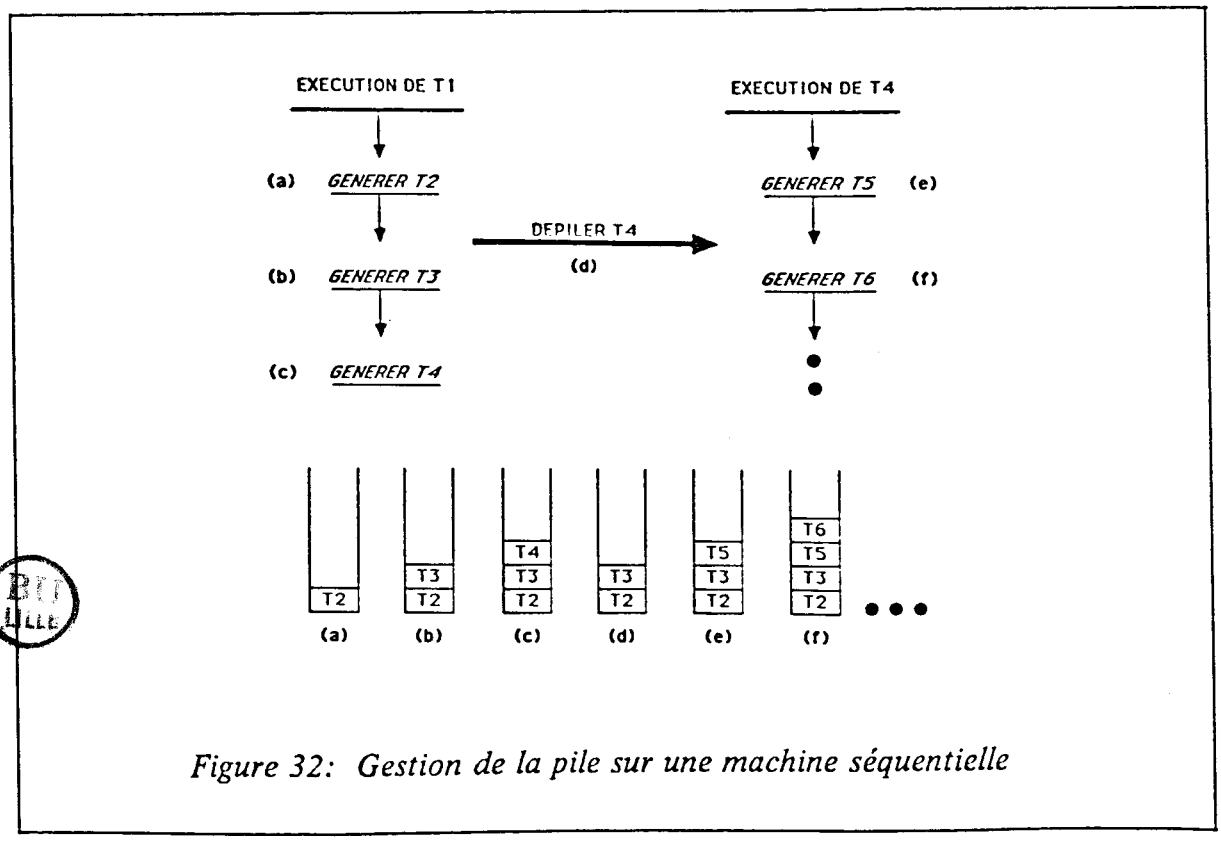
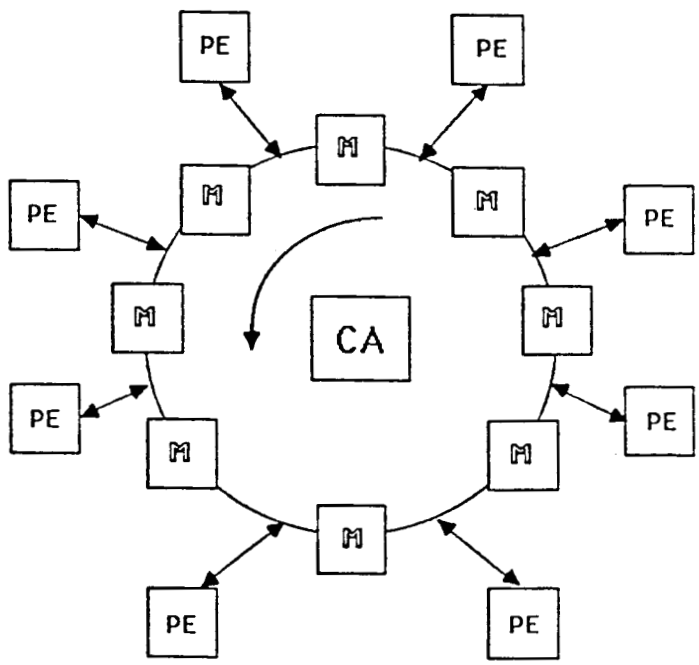


Figure 32: Gestion de la pile sur une machine séquentielle

2.4.2.2 Architecture pour pile partagée ou répartie

La gestion de la pile générale peut être assurée de plusieurs façons :

- *machine privilégiée:* sur un ensemble de machines l'une d'entre elles est chargée de gérer la pile générale. Elle peut être spécialisée pour ce travail et ainsi diminuer les temps d'exécution. Elle peut en même temps être chargée des entrées/sorties.



PE : PROCESSEUR ELEMENTAIRE
M : BLOC DE MEMOIRE CIRCULANTE
CA : CONTROLEUR D'ANNEAU

Figure 33: Architecture à mémoire circulante

BU LILLE

- *Mémoire circulante:* des blocs de particules de tailles variables circulent sur un anneau. Lorsque la pile locale est vide, le calculateur attend qu'un bloc de particules passe devant lui pour l'extraire de l'anneau. Lorsque la pile est pleine, une partie de ses éléments est placée sur l'anneau dès qu'un emplacement vide passe devant le calculateur. Le contrôleur d'anneau doit pouvoir déceler la fin de simulation, quand tous les calculateurs sont stoppés et que l'anneau est vide. Il doit aussi pouvoir décharger l'anneau lorsque celui – ci est plein et que plusieurs calculateurs sont en attente d'un bloc vide. (fig 33) Ce type de communication a déjà fait l'objet de recherches pour les architectures Data – Flow [20]. Les réalisations matérielles proposées, devraient pouvoir directement s'appliquer à cette gestion d'algorithme à pile.

- *Machine Ada:* Pour chaque particule secondaire générée, il est possible en Ada de créer une tâche indépendante pour suivre cette particule. La pile de particules est alors remplacée par une pile de tâches en attente d'un processeur libre. La gestion de cette pile doit être assurée par le compilateur. Les solutions précédentes peuvent être utilisées pour gérer cette pile de tâches. Dans le chapitre 4 cette génération de tâche par particule en Ada sera présentée.

2.4.3 Extension à plusieurs niveaux

Pour permettre d'augmenter le nombre de calculateurs sans risquer de saturer le système de communication, une solution consiste à appliquer l'algorithme sur plusieurs niveaux de piles. (fig 34)

Deux types de gestion sont alors possibles, soit les noeuds intermédiaires ne font que gérer les piles intermédiaires, soit ils participent au suivi de particules. Les architectures à mémoire circulante s'adaptent parfaitement à la première solution. (fig 35)

Un ou plusieurs anneaux supplémentaires sont construits autour des contrôleurs d'anneau des niveaux inférieurs. Lorsqu'un anneau est saturé, le contrôleur de celui – ci peut faire remonter une partie de l'information circulant sur son anneau vers l'anneau supérieur. S'il est sous – alimenté, le contrôleur devient demandeur de particules au niveau supérieur. Cet algorithme est totalement récursif entre les niveaux.

Le choix de la configuration est difficile à prendre, le nombre de contrôleurs et de calculateurs par anneau dépend de trois facteurs :

- les caractéristiques techniques de chaque élément.
- le temps CPU moyen pour le suivi d'une particule.
- le nombre moyen de particules générées

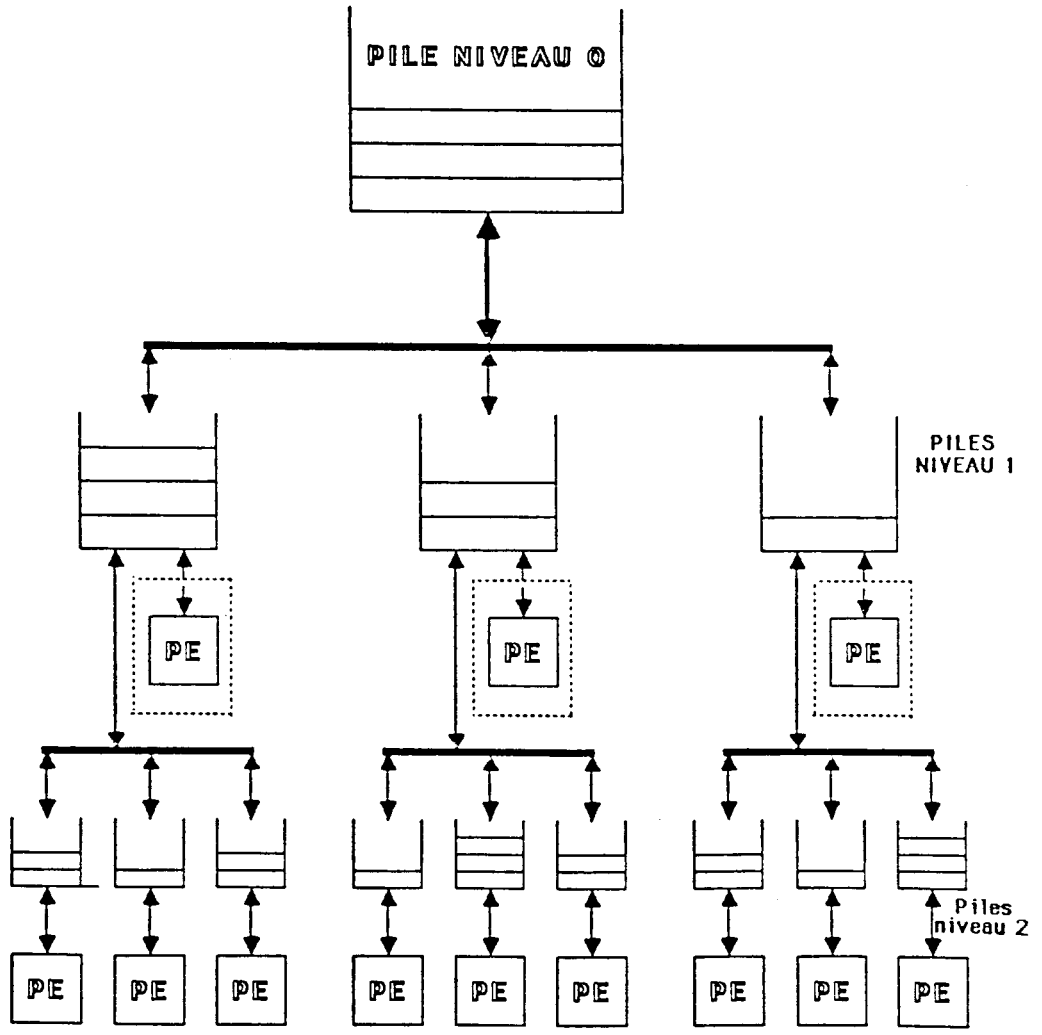


Figure 34: Pile répartie à plusieurs niveaux

BU LILLE

2.5 Conclusion

Les algorithmes de GEANT3 s'adaptent parfaitement aux architectures construites à partir de plusieurs processeurs. L'aspect indépendant des simulations de chaque évènement, le nombre restreint de communications entre les différentes

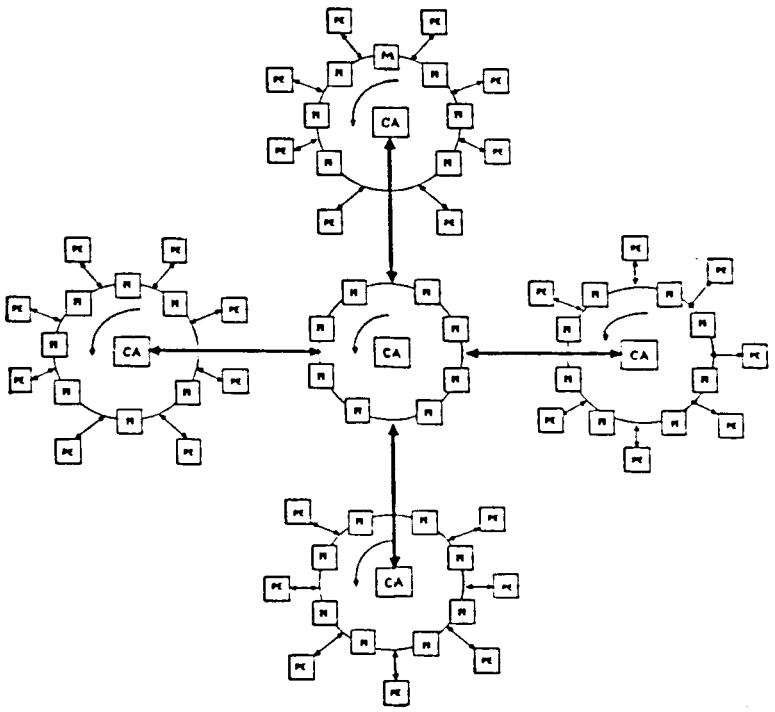


Figure 35: Architecture à mémoire circulante à plusieurs niveaux

BU LILLE

parties de GEANT3 autorisent l'association d'un grand nombre de processeurs. Seuls les problèmes de lecture, sauvegarde et reconstruction des données imposent certaines contraintes à l'architecture.

Deux types d'algorithmes ont été envisagés :

- L'algorithme multi – évènements permet de simuler un grand nombre d'évènements lorsque la géométrie et les paramètres de simulation ont été choisis. Il sera surtout intéressant pour la production massive de statistiques en traitement différé.
- L'algorithme à pile partagée ou répartie permet de simuler un évènement beaucoup plus rapidement et ce de façon interactive. Il permet aussi la production massive d'évènements. La mise au point de la géométrie et des paramètres de simulation peut être réalisée plus aisément. Cette solution demande un réseau de communication plus performant.

CHAPITRE 3
DEVELOPPEMENT SUR UN CALCULATEUR VECTORIEL

La définition même des méthodes Montè Carlo semble aller de pair avec les principes de vectorisation : des opérations identiques doivent être exécutées plusieurs fois sur des données différentes. Néanmoins, l'algorithme séquentiel utilisant plusieurs niveaux de boucles imbriquées et de nombreuses actions conditionnelles (cf Chap 1), il n'est pas possible d'obtenir une vectorisation directe efficace. Aussi les outils de vectorisation automatique appliqués sur les algorithmes séquentiels ne pourront donner des résultats satisfaisants. Il faut en fait définir de nouveaux algorithmes pour remplacer ceux existant, et non pas simplement les adapter.

Depuis une dizaine d'années, divers travaux ont été réalisés pour vectoriser des méthodes Montè Carlo. Les premières machines utilisées étaient des ILLIAC IV [27]. Depuis 1979 plusieurs groupes de travail ont développé des idées générales pour ces types d'algorithmes [5] [8] mais pour des utilisations simples et donc sur des programmes assez "courts" (GEANT3 = 40000 lignes, ZEBRA etc = 30000). Les machines utilisées sont maintenant des CRAY ou des Cyber 205, les performances obtenues sont de l'ordre de 5 à 10 fois plus rapides que les algorithmes séquentiels [4] [22]. Des travaux récents atteignent des facteurs de 40 a 60 mais toujours pour des simulations très particulières dans des géométries simples [6].

Etant donné la complexité des algorithmes développés dans GEANT3, il n'est pas possible d'envisager une vectorisation globale sur le nombre d'évènements simulés comme le proposent la plupart des travaux existants. Deux solutions différentes semblent pouvoir convenir pour GEANT3 :

- Soit par vectorisation des procédures de la partie géométrie manipulant des volumes qui contiennent un grand nombre de volumes positionnés de forme identique. Sans avoir à réécrire tous les algorithmes de simulation, la vectorisation n'interviendra que de façon ponctuelle sur un sous – ensemble de procédures qui n'agissent pas sur l'environnement de simulation d'un évènement.

- Soit par vectorisation du suivi des particules de même type générées depuis un même évènement et dans un même volume de l'environnement expérimental, la vectorisation devra être appliquée à l'ensemble des procédures de l'interface géométrique et physique.

3.1 Machine vectorielle

Depuis l'invention des calculateurs, nous avons assisté à une augmentation continuelle de la puissance de calcul. Aujourd'hui les plus puissants calculateurs qui développent plusieurs centaines de MFLOPS (Millions of flotting point operations per second) sont des machines de type vectoriel. Les physiciens étant des grands consommateurs de temps CPU, il confient d'adapter les algorithmes écrits pour des machines séquentielles afin d'utiliser au mieux les possibilités de ces supercomputers.

3.1.1 Concept de base

Flynn [16] a proposé une classification des différents types de calculateurs en fonction de la relation entre instructions et données.

- *SISD:* simple instruction, simple donnée
- *SIMD:* simple instruction, multiples données
- *MIMD:* multiples instructions, multiples données

Les machines classiques sont de la classe SISD. Les instructions sont exécutées une à une, et une instruction ne peut altérer qu'une seule donnée.

Les machines SIMD permettent d'exécuter la même instruction sur un grand nombre de données différentes. Les machines de ce type peuvent soit être construites par duplication de l'unité de calcul, l'ensemble étant piloté par une unité de contrôle spécialisée, soit utiliser une unité de calcul de type pipe – line, chaque unité fonctionelle étant alors segmentée en phases successives et pouvant être actives simultanément. C'est cette solution qui est en général choisie par les grands constructeurs de super – calculateurs tel CRAY et Control Data pour Cyber 205.

Les machines MIMD sont essentiellement construites à partir d'un ensemble de processeurs, chacun exécutant ses propres instructions sur des données privées ou partagées. Ce type de machine pourra présenter des caractéristiques intéressantes pour le développement d'algorithmes de type multi – tâches bien adaptés au cas de GEANT 3. (cf chap 4)

3.1.2 Machine SIMD pipe – line

Le traitement sur une machine pipe – line peut être comparé à la construction d'une voiture sur une chaîne de production. Le calcul est décomposé en un ensemble de sous – tâches indépendantes mais ordonnées qui s'exécutent sur un flux continu de données.

Le flux de données circule à travers le pipe – line. Après un temps de start – up égal au temps nécessaire pour obtenir le premier résultat, un résultat sera produit à chaque cycle machine.(fig 36)

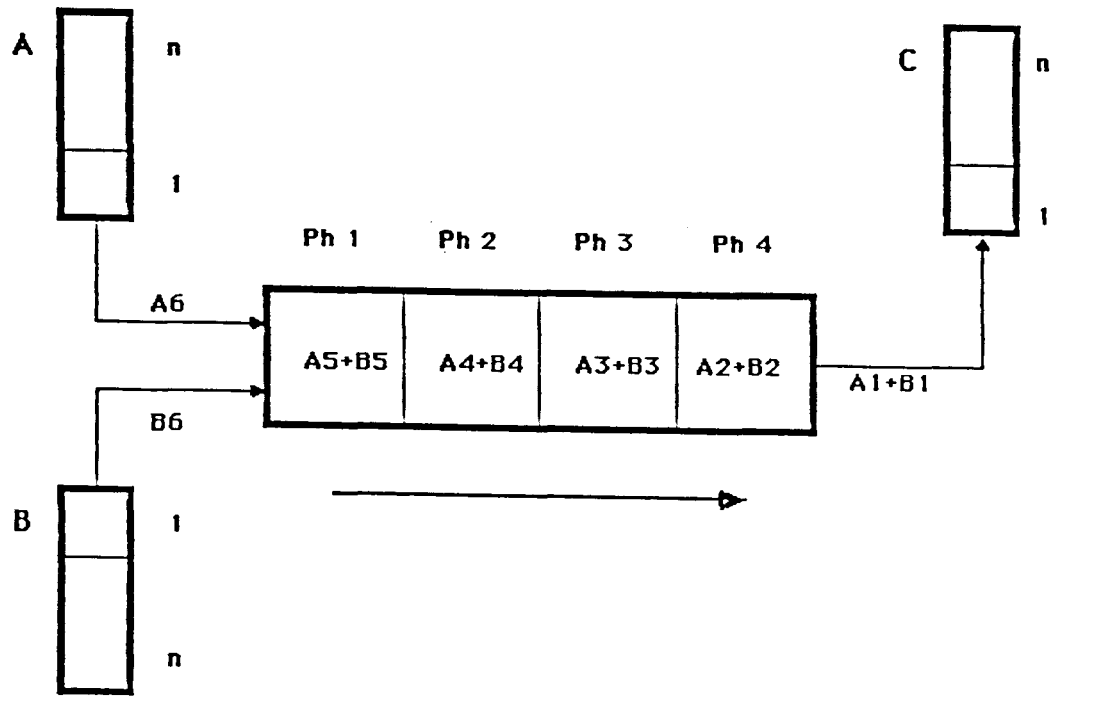


Figure 36: Evolution d'une addition en mode pipe – line

L'exécution d'une instruction vectorielle est constituée de deux phases

- la phase de remplissage du pipe – line
- la phase de production de résultats

Le temps total d'une instruction vectorielle i est égal à

$$T(i) = S(i) + L / R(i)$$

T(i) = temps d'exécution total de l'instruction i
S(i) = temps de start – up pour l'instruction i
R(i) = nombre de résultats produits par seconde dans la deuxième phase par l'instruction i
L = maximum des longueurs des vecteurs données et résultats

R(i) et S(i) sont les caractéristiques du calculateur pour l'instruction i. La relation précédente permet d'affirmer que lorsque la longueur L est assez grande, le temps de start – up devient négligeable. Aussi les algorithmes vectoriels devront tenir compte des longueurs des vecteurs traités pour ainsi approcher la puissance maximale de la machine.(fig 37)

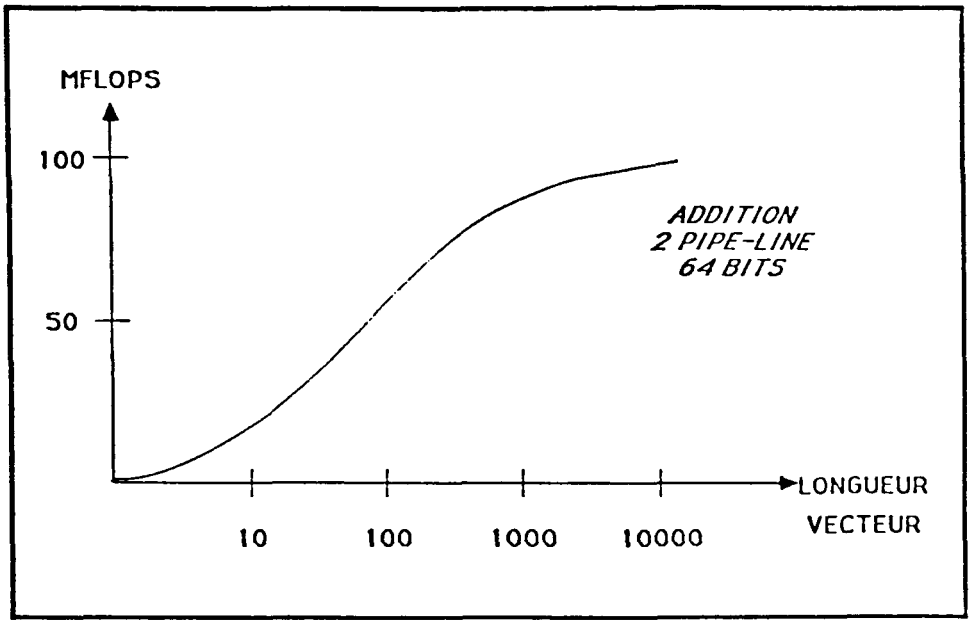


Figure 37: MFLOPS en fonction de la longueur du vecteur sur CY205

Une technique de mesure des gains obtenus par vectorisation peut être simplement de calculer le rapport entre le temps d'exécution séquentielle et le temps d'exécution vectorielle. Cette technique n'est exacte que dans le cas où les résultats produits par les deux algorithmes sont exactement les mêmes, ce qui n'est pas toujours nécessaire pour les méthodes Montè Carlo.

3.1.3 Machine SIMD de type tableau

Une autre façon de réaliser des machines vectorielles est d'associer plusieurs processeurs élémentaires (PE), chacun possédant sa mémoire locale, son jeu de registres et une interface avec un système de communication. Chaque PE exécute la même instruction sur des données préalablement envoyées par une unique unité de contrôle et de commande très spécialisée. Ce type de solution a été choisi pour l'unité vectorielle de la machine ISIS de BULL [26].

Ce super – calculateur est prévu pour atteindre une puissance de crête maximale de 1 GFLOPS pour son unité vectorielle. Les particularités de cette machine sont

- une unité scalaire de type multi – flots pouvant exécuter simultanément quatre tâches indépendantes
- une mémoire hiérarchisée à trois niveaux [13].
- Une unité vectorielle composée de 8 à 32 PE. Chaque PE traite des mots de 64 bits à raison de un mot toutes les 30 nanosecondes. Une bascule d'activité permet d'inhiber l'action de chaque PE. Le séquenceur centralisé, recevant des modèles vectoriels initialisés par l'un des flots de l'unité scalaire, est chargé de gérer l'ensemble.

Les opérations sur les registres de chaque PE sont soit des opérations flottantes, soit des opérations de conversions soit encore des opérations entières. Les résultats sont des mots de 64 bits ou 1 bit, il est possible d'exécuter des opérations vectorielles conditionnelles.

3.2 Vectorisation des procédures géométriques de GEANT3

Parmi les deux approches possibles pour la vectorisation de GEANT3, celle que nous avons choisi de réaliser en premier est celle liée à la géométrie. Il semblait être plus facile de commencer par ce type de vectorisation qui n'entraînait pas de modifications essentielles des algorithmes généraux de simulation, mais juste la création de nouveaux algorithmes relativement ou complètement indépendants de l'environnement associé à un évènement.

Le temps passé dans l'ensemble des procédures géométriques n'étant pas négligeable,(fig 38) pouvant atteindre de 25 à 75 pour cent du temps CPU total, les résultats obtenus seraient rapidement satisfaisants et ce pour un moindre coût en hommes par année.

Ce travail a été complètement réalisé sur un Cyber 205 en collaboration avec les équipes de recherche de Florida State University et Supercomputer Computations Research Institute of Tallahassee (Florida) spécialisées dans la vectorisation d'algorithmes en physique des particules [18].

3.2.1 Détection des algorithmes vectorisables

Certains travaux ont prétendu que les algorithmes de progression d'une particule dans un volume ne pouvaient être vectorisés [3]. En fait pour **certaines** géométries, nous allons voir qu'il n'en est rien et que de bons résultats peuvent être obtenus.

Comme nous l'avons montré dans le chapitre 1, l'utilisateur doit définir la géométrie de son environnement expérimental par la définition d'un ensemble de volumes élémentaires. En fait, il définit des types de volumes élémentaires de certaines formes et de certaines tailles, ces types sont ensuite utilisés pour créer son arborescence de volumes par positionnement de un ou plusieurs éléments de un ou

Repartition temps CPU %	GOPAL 4 evts	GEXAM1 20 evts
PHYSIQUE	1.5	10.1
TRACKING	3.0	10.1
GEOMETRIE	86.5	55.3
DIVERS	9.0	24.5

Figure 38: Répartition du temps CPU pour GOPAL, GEXAM1 (cf 3.2.5)

plusieurs types dans d'autres éléments déjà positionnés.(fig 9 Chap 1) Chaque élément peut aussi être divisé et ainsi créer de nouveaux volumes.

En pratique, les géométries utilisées dans les détecteurs sont construites de façon homogène, une seule sorte de forme est positionnée dans un même volume. D'ailleurs, si plusieurs sortes de formes sont présentes, nous pouvons obtenir l'homogènéité voulue par création de volumes fictifs intermédiaires.

Le nombre de volumes positionnés dans un même volume peut être grand, il peut atteindre 400 unités (expérience LEP OPAL).

De ces deux remarques, nous pouvons extraire l'idée principale utilisée pour cette vectorisation : il suffit de trouver tous les algorithmes répondant à la question :

Quels sont les traitements qui doivent être répétés sur chaque élément positionné dans un même volume?

S'il existe de tels traitements, alors la vectorisation devient possible car il suffira d'exécuter un seul traitement en utilisant un vecteur. Chaque élément du vecteur représentera l'information caractérisant un des éléments positionnés : soit sa taille et sa position.

Du nombre de volumes positionnés dépendra la longueur des vecteurs traités. De l'homogènéité dépendra le nombre d'algorithmes à exécuter : pour chaque type de forme présent dans un volume, il faudra exécuter un traitement différent sur la partie correspondante des vecteurs. (cf 3.4.2)

Les algorithmes manipulant des volumes divisés ne présentent pas quant à eux d'aspect vectorisable. Aucun balayage n'est nécessaire puisque la régularité des divisions permet d'utiliser des procédures de calculs directs.

Deux procédures de GEANT3 ont été choisies pour être vectorisées, GNEXT et GSEAR. Il semble en effet que seules ces deux procédures répondent parfaitement aux conditions de vectorisation précédemment énoncées.

3.2.2 Construction des vecteurs en mémoire

Les algorithmes de vectorisation que nous proposons seront pour l'instant appliqués à des expériences n'utilisant que les cas particuliers de positionnement correspondant à la définition faite en 1.4 qui permettent d'éviter l'appel à la procédure GPCHK. Pour le cas général, il faudrait envisager de vectoriser cette procédure sur le nombre de volumes positionnés dans un même volume : ce qui n'a pas encore été fait.

La grande taille de la mémoire des supercomputers permet aussi de dupliquer l'information, si celle–ci n'est jamais modifiée, et de la structurer de façon différente. Les structures de données ZEBRA définies pour GEANT3, ne permettent pas de manipuler facilement des vecteurs, l'information étant dispersée dans la mémoire et accessible par des pointeurs, il faudra à chaque fois restructurer cette information de façon à la placer dans une zone mémoire contiguë (utilisation de GATHER cf 3.4.1). Cette restructuration peut elle aussi n'être effectuée qu'une fois en début de simulation et les résultats stockés en mémoire pour une utilisation ultérieure.

3.2.3 Vectorisation de GNEXT

La procédure GNEXT (cf Chap 1) calcule la distance entre un point donné et la prochaine frontière géométrique suivant une direction donnée ainsi que la distance minimale entre le point et toutes les frontières possibles indépendamment de la direction.

Chaque élément positionné dans le volume courant présente plusieurs frontières, donc pour chacun d'entre eux, il faut calculer la distance suivant la direction donnée entre le point et la plus proche de ses frontières si toutefois l'intersection est possible, puis la distance minimale quelque soit la direction. Les minimums seront ensuite calculés parmi tous les résultats obtenus pour chaque élément.(fig 39)

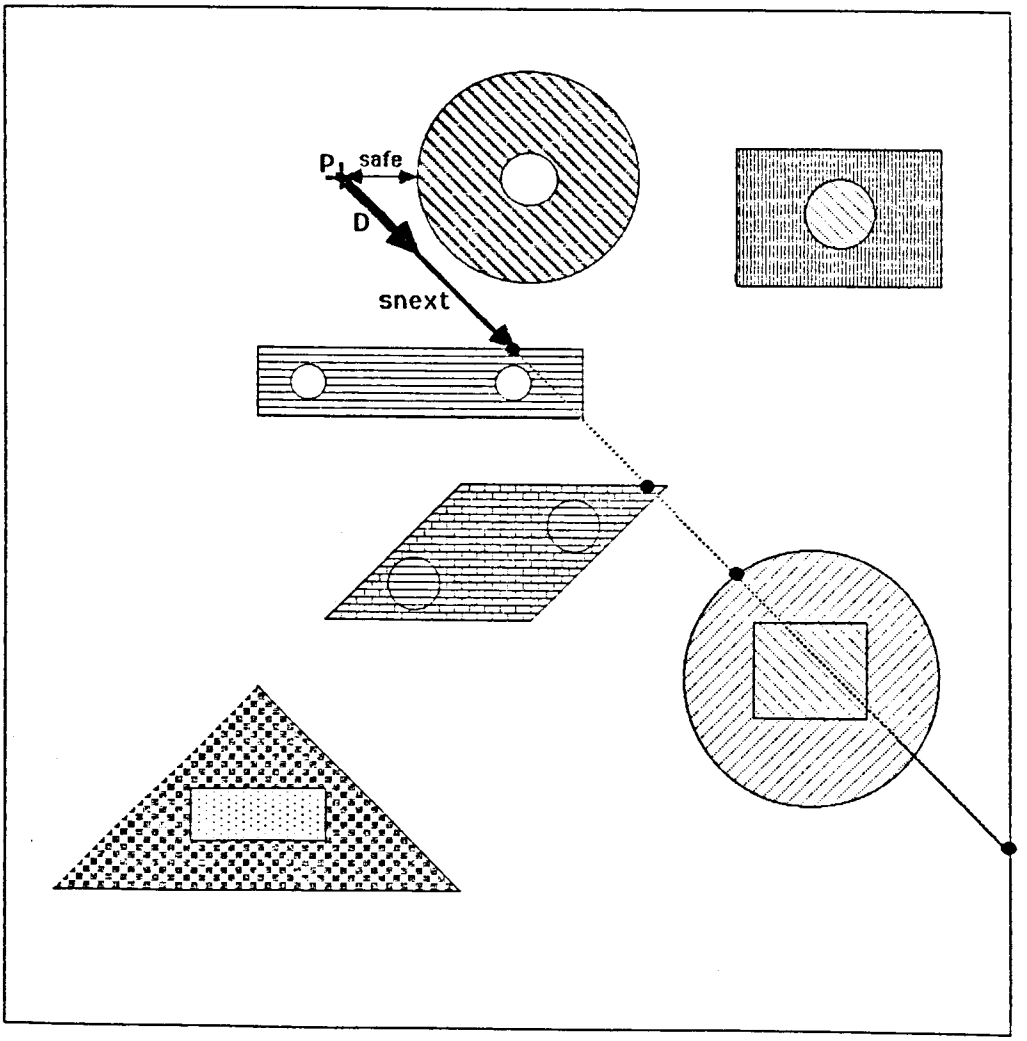


Figure 39: Utilisation de GNEXT (2 dimensions)

BU LILLE

L'algorithme est divisé en deux parties. La première consiste à calculer la distance entre le point et les frontières du volume dans lequel il se trouve; la deuxième à calculer le minimum des distances entre le point et tous les volumes positionnés directement dans le volume contenant ce point. Cette deuxième partie répond parfaitement aux conditions de vectorisation précédemment énoncées.

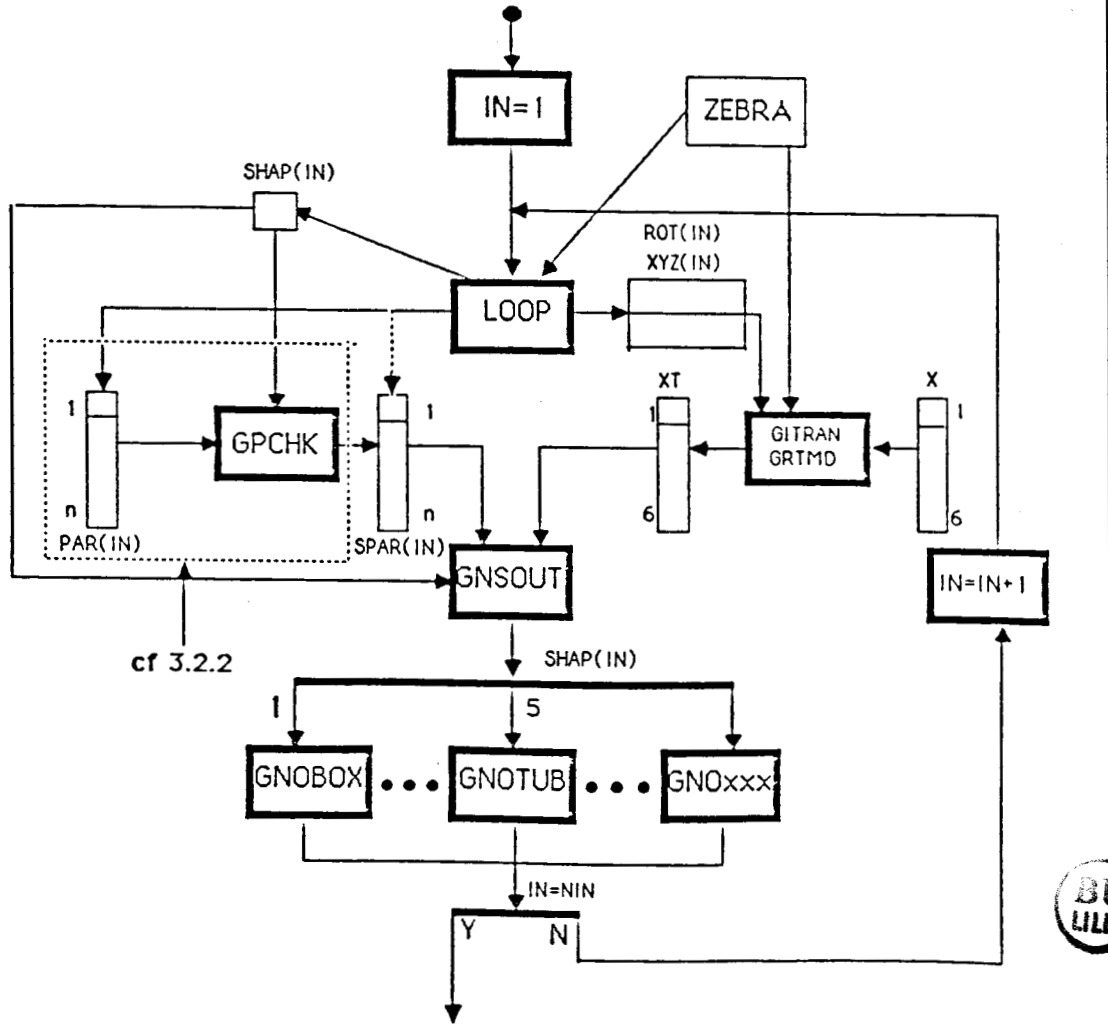


Figure 40: Partie de l'algorithme séquentiel de GNEXT

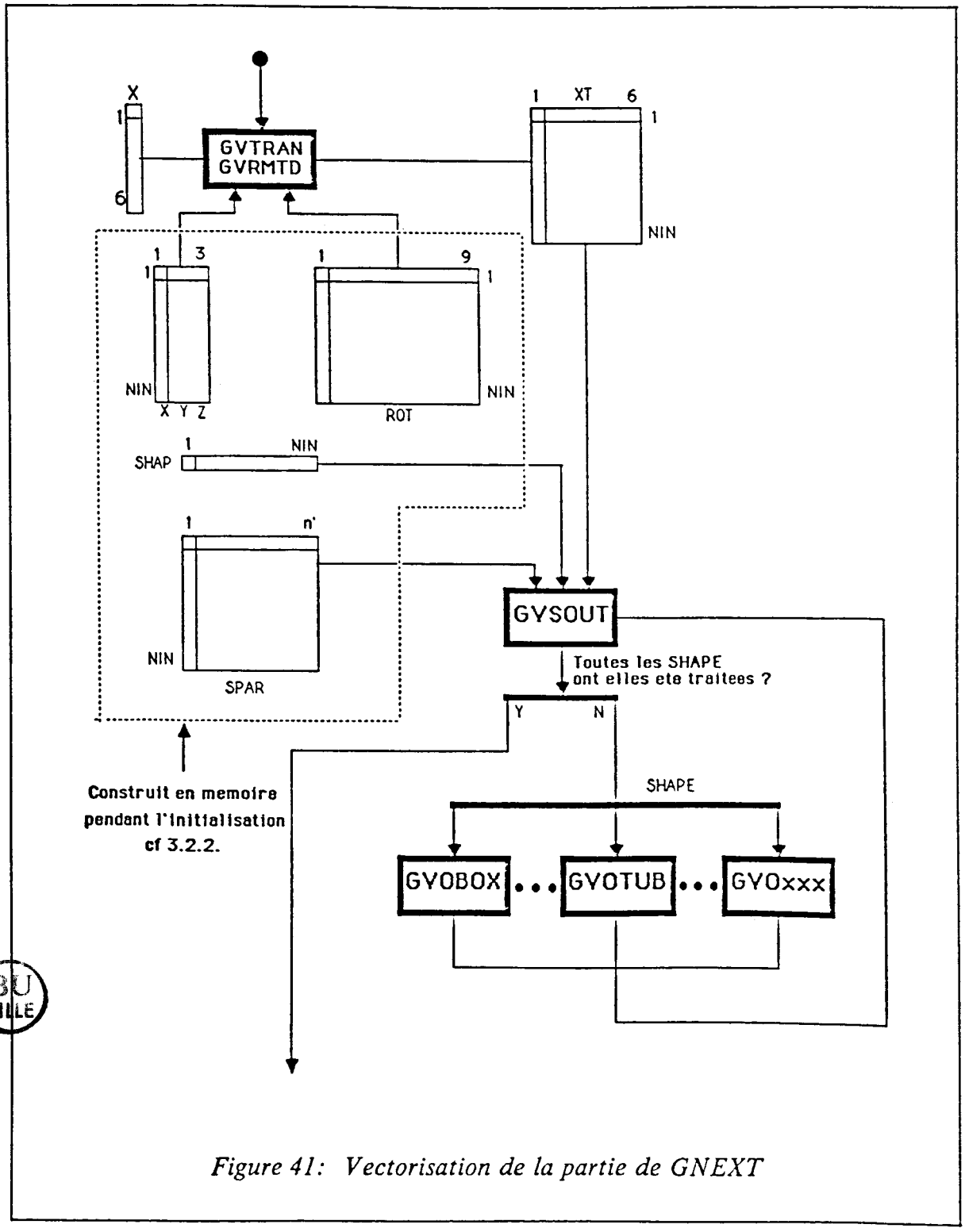


Figure 41: Vectorisation de la partie de GNEXT

L'algorithme séquentiel pour cette deuxième partie, consiste à calculer pour

chaque élément positionnés, les coordonnées du point dans le repère local de l'élément en utilisant les procédure GITRAN et GRMTD (translation – rotation de la position et rotation de la direction), les paramètres de cet élément sont vérifiés par GPCHK, puis la distance est calculée par la procédure GNSOUT qui en fonction de la forme de l'élément choisira une des procédures spécialisées GNOBOX GNOTUB GNOTRA .... (fig 40)

Deux types de vecteurs seront utilisés; ceux dépendants du point et de la direction, et ceux indépendants du point et de la direction mais dépendants de la géométrie. Ceux – ci pourront être créés une fois pour toutes en début de simulation puisqu'après l'initialisation de l'environnement expérimental, la géométrie n'est plus modifiée. (cf 3.2.2) En particulier, les paramètres, les vecteurs translations, les matrices de rotation et le type de forme de chaque élément sont rangés de façon contiguë en mémoire dans un common FORTRAN ou dans une banque ZEBRA à partir des structures initiales de ZEBRA et pourront être directement utilisés comme vecteurs.

La boucle séquentielle est remplacée par un appel aux procédures GVTRAN et GVRMTD qui à partir du point construisent un vecteur où chaque élément représente la translation et la rotation du point dans un des volumes positionnés.

Pour chaque type de forme présente dans le volume courant, une procédure spécialisée sera appelée juste une fois par l'intermédiaire de la procédure GVSOUT. Chaque procédure (GVOTUB GVOBOX GVOTRA .....) exécute les calculs sur la totalité des vecteurs, c'est seulement à la fin de celle – ci que le minimun sera calculé sur les résultats correspondant aux éléments de forme acceptée par la procédure.(fig 41) Cette solution est plus rapide que celle qui consisterait à extraire des sous – vecteurs ne contenant que des éléments de même forme pour effectuer les calculs.(cf 3.4.2.)

3.2.4 Vectorisation de GSEAR

Lors d'un changement de volume, il faut pouvoir définir dans quel nouveau volume la particule se trouve. Pour cela il faut explorer l'arbre définissant la géométrie. Quand un noeud de l'arbre est trouvé, il faut, si celui – ci posséde des volumes positionnés ou des divisions, chercher si un des volumes fils contient la particule et ce récursivement jusqu'à ce qu'aucun volume ne la contienne ou que le volume trouvé soit une feuille de l'arbre. Il s'agit en fait d'une descente dans l'arbre, le choix du chemin à suivre étant fonction de la position de la particule.

Lorsqu'un volume donné contient des volumes positionnés, c'est la procédure GSEAR qui est chargée de scruter les volumes fils et donc de définir quelle branche de l'arbre doit être suivie.

Pour chaque élément positionné, les coordonnées du point doivent être transformées pour le repère associé à chaque volume. (GITRAN) Les paramètres

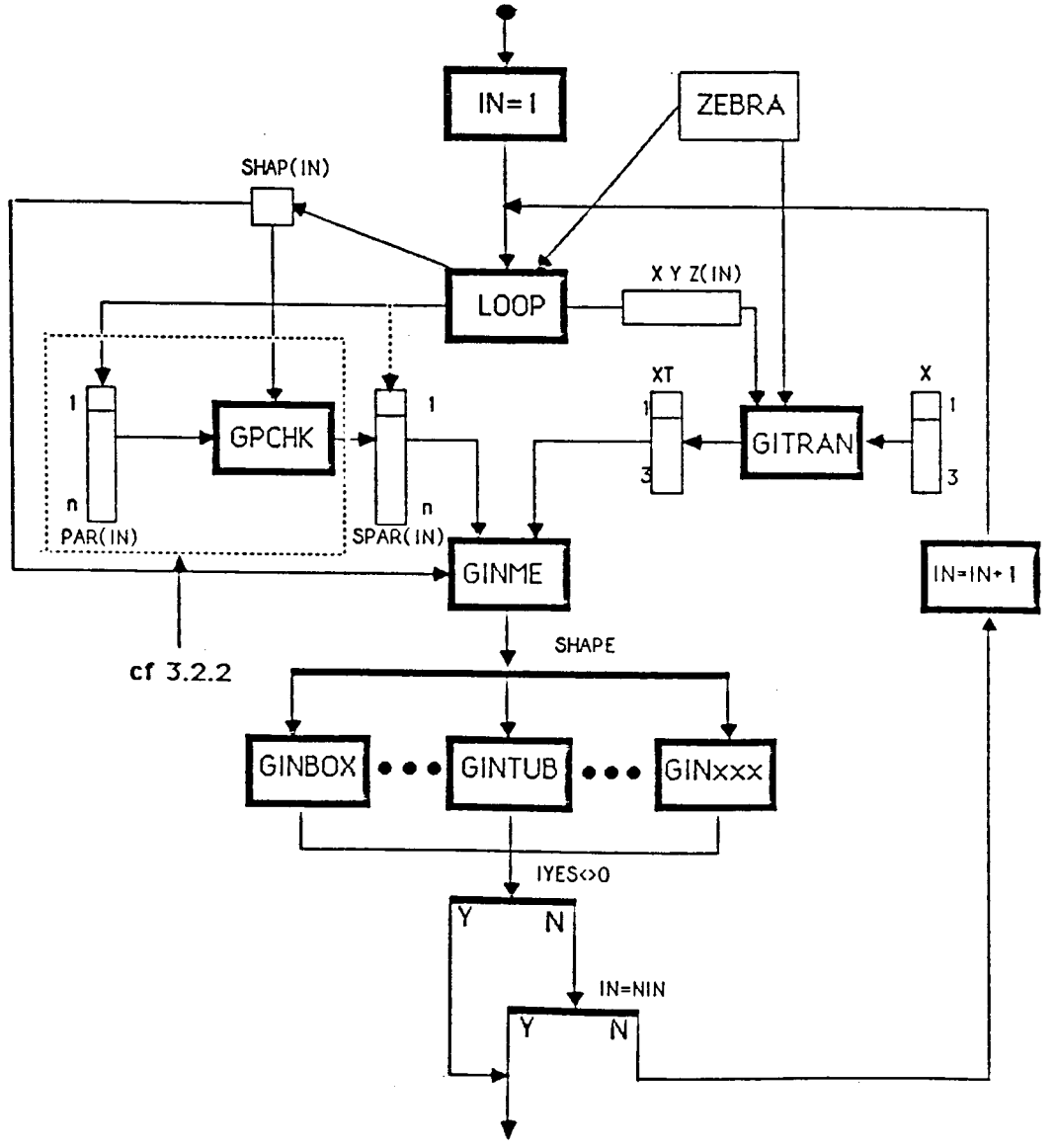


Figure 42: Partie de l'algorithme séquentiel de GSEAR

de chaque volume positionné doivent être vérifiés par GPCHK, puis la procédure GINME est appelée pour chacun de ces volumes afin de tester si la particule se trouve dans l'un d'entre eux, lorsqu'un volume la contient la recherche est arrêtée. (fig 42)

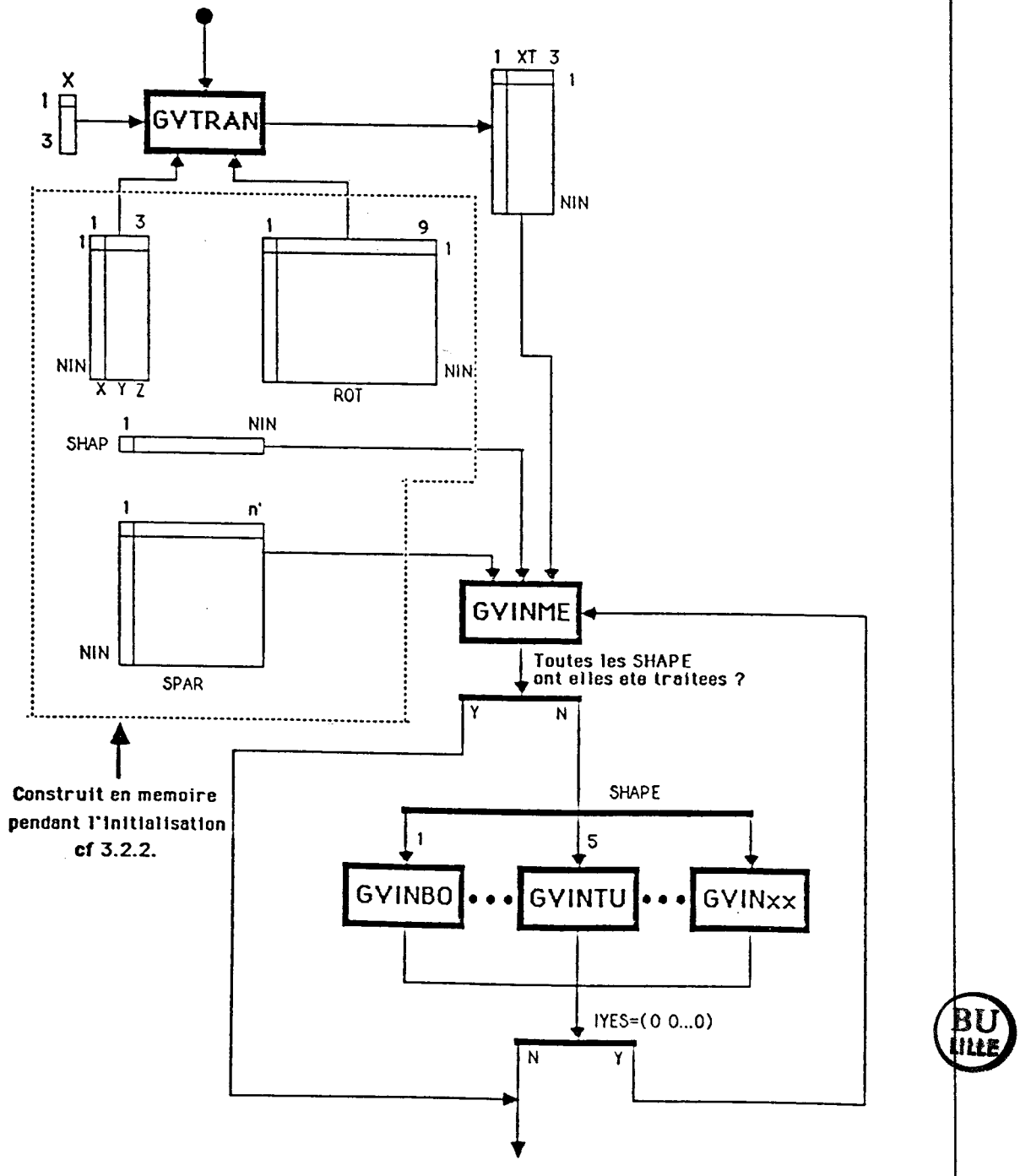


Figure 43: Vectorisation d'une partie de GSEAR

BU LILLE

Pour la vectorisation de GSEAR, les vecteurs indépendants du point et de la direction de la particule mais dépendants de la géométrie seront créés en début de simulation, les autres seront construits par la procédure GVTRAN. La boucle de l'algorithme séquentiel est remplacée par un appel à la procédure GVINME. Pour chaque forme présente dans le volume courant, une des procédures spécialisées sera appelée juste une fois. Les calculs sont effectués sur les vecteurs complets, à la fin, seuls les résultats associés à des éléments de forme reconnue par la procédure appelée servent à définir si oui ou non le point appartient à un des éléments.
(fig 43)

Le nombre d'itérations pour l'algorithme séquentiel peut être très petit (si le point appartient à l'un des premiers éléments considérés), et le temps d'exécution peut donc être inférieur à celui de l'exécution vectorielle, mais en moyenne, le nombre d'itérations pour une recherche n'utilisant pas les optimisations séquentielles, sera de l'ordre de N/2 (N étant le nombre d'éléments positionnés), en général l'algorithme vectoriel sera donc plus performant.

3.2.5 Performances de cette vectorisation sur Cyber 205

Les gains obtenus par cette vectorisation peuvent être très différents suivant le type de géométrie utilisé. Si celle – ci est construite de telle sorte que l'arité de l'arbre soit faible ou que l'homogènéité ne soit pas respectée, les gains seront nuls. En revanche, pour une bonne définition de la géométrie, les gains seront fonctions du temps CPU passé dans les procédures vectorisées GNEXT et GSEAR ainsi que dans toutes les procédures appelées par elles dans l'algorithme séquentiel. En effet, le gain ne pourra être supérieur au pourcentage de temps CPU nécessaire pour l'exécution de l'ensemble de ces procédures.

Les essais (fig 45) auxquels nous avons procédé, ont reposé sur trois types d'expériences différentes [14].

- *GEXAM1:* Il s'agit d'un exemple d'utilisation de GEANT3 qui servit d'ailleurs de support pour comparer GEANT3 et EGS3 [7]. Sa géométrie n'est pas du tout adaptée à la vectorisation puisque aucun volume n'a été construit par positionnement, tous l'ont été par division. Les parties des procédures GNEXT et GSEAR qui ont été vectorisées n'interviennent pas pour des volumes construits par division.

- *GEXJLD:* Avec les mêmes processus physiques que GEXAM1, nous avons construit une nouvelle géométrie composée d'une boite servant de volume de référence, dans laquelle nous avons positionné 64 boites, chacune contenant aussi 64 boites de taille inférieure. Deux types de positionnement ont été testés, avec et sans rotation pour les boites de troisième niveau.(fig 44)

- *D0:* Il s'agit d'une expérience en préparation à Fermilab pour laquelle une évaluation de huit heures CPU pour la simulation complète d'un évènement

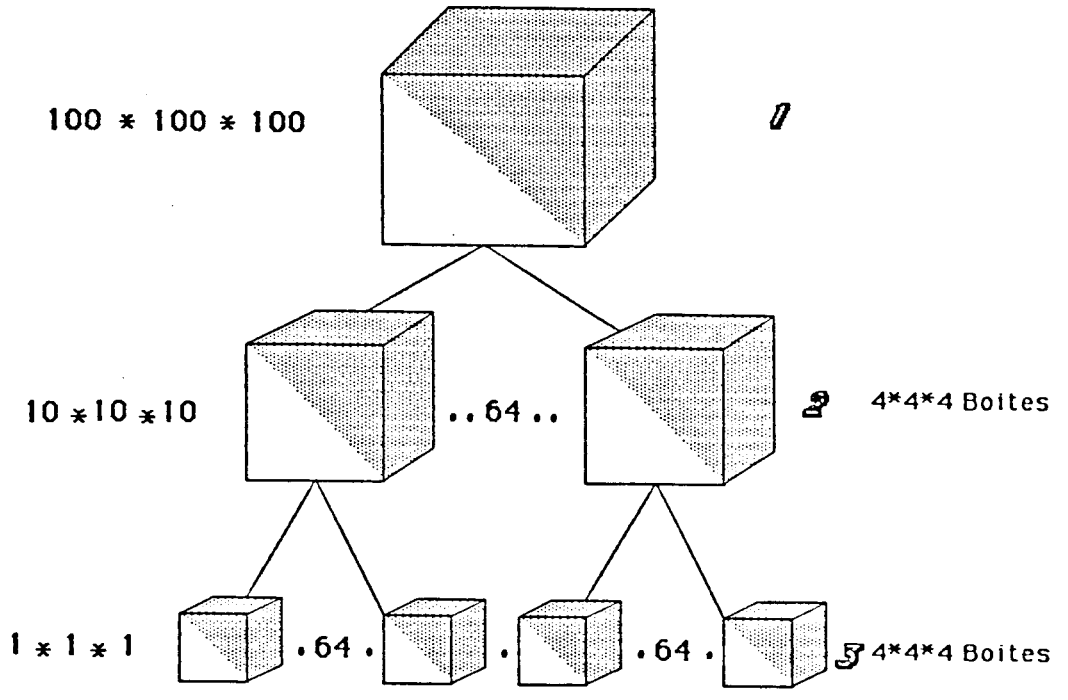


Figure 44: Arborescence de la géométrie de GEXJLD

BU LILLE

sur un VAX 11/780 a été faite [1]. La géométrie sur laquelle nous avons travaillé, était construite à partir de trois tubes, chaque tube contenant 64 ou 96 trapèzoides. Les processus physiques utilisés étaient plus complexes que ceux des exemples précédents. Plusieurs directions initiales ont été testées, deux d'entre elles représentent les deux extrêmes des gains obtenus.

Des types d'optimisations séquentielles sont définies par GEANT3, elles utilisent aussi cet aspect de la géométrie.

- *GSORD:* Elle crée des frontières fictives le long d'un axe. La particule étant entre deux frontières fictives, la procédure GNEXT n'aura à considérer que le sous – ensemble de volumes présent entre ces deux frontières pour le calcul de la distance minimale.

- *GSNEXT,GSSEAR:* l'ordre de scrutation des volumes positionnés pour la procédure GSEAR peut être soit défini par l'utilisateur en utilisant les propriétés de voisinage, soit être calculé de façon statistique pendant la simulation.

L'optimisation séquentielle GSORD a été testée sur GEXJLD. Les gains obtenus par vectorisation sont du même ordre de grandeur que ceux obtenus par l'utilisation de GSORD. Il semble néanmoins que la vectorisation soit en générale plus performante que l'optimisation par GSORD, les gains ne dépendent en fait que du nombre de volumes positionnés et non de leur répartition dans l'espace. De plus, la vectorisation produit les mêmes résultats bit à bit que l'exécution séquentielle non optimisée ce qui n'est pas le cas pour GSORD qui, par l'ajout de frontières fictives, peut générer des pas de progression différents (une particule ne peut franchir une frontière même fictive en un seul pas de progression).

TEMPS CPU PAR EVENT	GEXAM1	GEXJLD / GSORD	GEXJLD rot	DO min	DO max
SEQUENTIEL	9.5	96.55 / 10.30	143.05	25.71	47.73
VECTORIEL	9.5	9.89 / 9.89	11.35	15.3	6.18
GAIN	*1*	*9.8 / 1.05*	*12.5*	*1.7*	*7.5*

Figure 45: Tableau des gains pour la vectorisation de la géométrie

La vectorisation des procédures géométriques ne doit être en fait considérée que comme une approche des possibilités vectorielles de GEANT3. Les gains obtenus sont en général, légérement supérieurs à ceux obtenus par l'utilisation des optimisations séquentielles. Il convient d'investiguer d'autres voies de vectorisations qui le cas échéant pourront utiliser ces algorithmes.

Les outils de vectorisation automatique proposés sur le Cyber 205 ont été utilisés. (fig 46) Le gain obtenu sur GEANT3 est de l'ordre de 20 pour cent. Ce faible pourcentage s'explique par la complexité des algorithmes se trouvant à l'intérieur des boucles principales de GEANT3. (appel de procédures, instructions conditionnelles, boucles imbriquées ....)

Can Appear in DO Loop	Must Not Appear in Any Part of Do Loop
Vectorizable loops nested within the loop.	Nonvectorizable loop nested in the loop.
Total iteration count less than 65535 for a nest of loops.	Total iteration count greater than or equal to 65535 for a nest of loops.
CONTINUE statement.	Any flow control statement besides DO and CONTINUE.
Arithmetic operators +, -, *, /, and **, logical operators.	Relational operators.
Integer, real, half-precision, and logical data elements.	Any data element has a type other than integer, real, half-precision, or logical.
	Any input, output, or memory transfer statements.
Reference or calls to the following functions and subroutines: ABS, ACOS, ALOG, ALOG10, ASIN, ATAN, COS, EXP, FLOAT, LABS, IFIX, SIN, SQRT, and TAN.	References and calls to functions and subroutines other than ABS, ACOS, ALOG, ALOG10, ASIN, ATAN, COS, EXP, FLOAT, LABS, IFIX, SIN, SQRT, and TAN.
Any data elements appearing on the left side of an assignment statement that appear in an EQUIVALENCE statement if the UNSAFE compilation option is selected.	Any data elements appearing on the left side of an assignment statement which have appeared in EQUIVALENCE statements if the UNSAFE compilation option is not selected.
Any scalar assignment statement whose right side is an integer, real, half-precision, or logical expression.	Vector assignment statements.
Loop-dependent subscripts having one of the forms c, c+n, c-n, or c*n, where c is a control variable and n is an integer constant.	Loop-dependent subscripts not of one of the forms c, c+n, c-n, c*n, where c is a control variable and n is an integer constant.
References to variably-dimensioned arrays, so long as the terminal value of the loop is constant.	References to any variably-dimensioned array when the terminal value of the loop is variable (can be vectorized if the UNSAFE compilation option is selected).
Loop-independent subscripts.	

Figure 46: Critères de vectorisation de boucle sur Cyber 205

3.3 Vectorisation des algorithmes généraux de simulation

Pour obtenir des résultats représentatifs avec des méthodes Montè Carlo, il faut simuler un grand nombre d'évènements. En pratique, il est courant de simuler plusieurs milliers d'évènements identiques pour une même expérience. Dans l'ensemble, c'est donc un travail répétitif qui sera fait sur chacun d'entre eux. D'où l'idée immédiate de vectoriser sur le nombre d'évènements.

Malheureusement la vectorisation n'est pas directement concevable. Lors du suivi d'une particule, son évolution dans l'espace est dépendante du générateur de nombres aléatoires : c'est la définition même des méthodes Montè Carlo. Chaque évènement pourra évoluer de façon différente, aussi les traitements exécutés pour deux évènements ne seront pas forcément les mêmes. Etant donné le nombre important de cas possibles, les vecteurs seront vite "émiettés" en de nombreux petits vecteurs pour lesquels il faudra appliquer à chaque itération un grand nombre de traitements conditionnels.

Pour parer à cet inconvénient empêchant une vectorisation générale sur la simulation de N évènements, il suffit d'utiliser la vectorisation de façon locale; à un instant donné, la vectorisation sera appliquée non pas à toutes les particules présentes dans l'environnement mais sur un sous – ensemble de particules ayant certaines caractéristiques communes au sens géométrique et au sens physique.

3.3.1 Algorithme général vectorisé

A un instant donné, un vecteur de particules (position, impulsion et temps) appartenant à un même volume sera construit. Le volume sera choisi de façon à obtenir le plus long vecteur possible.

Les procédures de l'interface géométrique pourront être vectorisées pour toutes les particules appartenant à un même volume. Une double vectorisation est envisageable pour les procédures GNEXT et GSEAR (cf 3.2.): la longueur des vecteurs peut être le produit du nombre de particules dans le volume courant par le nombre de volumes positionnés dans ce même volume.

En fonction des types de particules présents dans le vecteur (électron pion muon ...), seront appelées les procédures physiques associées, elles aussi pourront être vectorisées.

Chaque particule aura donc progressé dans l'environnement. Si elle appartient encore au même volume, elle sera laissée dans le vecteur sinon, elle sera replacée dans la pile générale pour stockage temporaire. Si le vecteur est encore assez long, la simulation peut continuer sinon, toutes les particules sont replacées dans la pile, un autre vecteur plus long sera construit pour un autre volume à partir de la pile générale, le traitement peut ainsi continuer jusqu'à ce que la pile soit vide.

- *Algorithme général en pseudo langage:*

```
-empiler N particules dans la pile ( GUKINE * N evenements )
 TQ pile non vide FAIRE
  -extraire de la pile le plus grand vecteur de particules appartenant
     a un meme volume, soit L la longueur ( GTREVE )
  -initialisation de l'environnement associe a ce volume ( GTRACK )
      * appel a GMEDIA
      * mise a jour des caracteristiques du materiau utilise pour ce volume
  TQ L > seuil FAIRE
   -calcul d'un vecteur representant les minima des distances
      entre chaque particule et les frontieres geometriques ( GNEXT )
    POUR chaque type de particule FAIRE
     -construire un vecteur de bits indiquant quelles sont les
        particules du type courant
      SI ∃ au moins une particule du type courant ALORS
       -appel de la procedure associee au type de particule courant
          pour le vecteur de particules et le vecteur de bits de
```

```
          controle ( GTELEC GTGAMA .... )
        FSI
      FPOUR
   -mise a jour du vecteur de particules par extraction des particules
      stoppees et transfert vers la pile des particules ayant change
      de volume
   -controle a l'utilisateur apres chaque pas pour chaque particule
      ( vectorisation possible au niveau utilisateur GUSTEP )
    FTQ
  -transferer toutes les particules du vecteur dans la pile pour
     stockage temporaire
   FTQ
```

3.3.2 Prévision des performances

Les gains prévisibles par une telle vectorisation dépendent essentiellement des longueurs des vecteurs traités, et du type de particules simulées.

Soit P : le nombre de pas de progession effectués pour toutes les particules suivies lors de la simulation.

Pour chaque pas de progression $t \in [1..P]$, on peut définir

Soit N(t) : le nombre de particules présentes dans l'environnement expérimental à l'itération t.

Pi(t) : le pourcentage des particules présentes dans un volume i à l'itération t.

Le choix du volume courant est fait de telle sorte que Pi(t) soit maximal. La longueur du vecteur de particules ainsi construit à l'itération t est

$$L(t) = N(t) \times \underset{i}{\text{Max}}\ Pi(t)$$

d'où la longueur moyenne entre les itérations t_0 et $t_0 + m$

$$Lmoy = \left(\sum_{k=1,m} \left(N(t_0+k) \times \underset{i}{\text{Max}}\ Pi(t_0+k) \right) \right) / m$$

Les gains sur l'initialisation de l'environnement associé au volume sont de l'ordre

g_1 = Lmoy % un seul appel au lieu de Lmoy %

Les gains théoriques sur les procédures géométriques telle que GNEXT et GSEAR sont de l'ordre

$$g_2 = Pv(Lmoy \times NVmoy) / Ps$$

Pv(j) : puissance en MFLOPs de calcul pour des vecteurs de taille j
Ps : puissance pour un calcul séquentiel
NVmoy : nombre moyen de volumes positionnés dans tous les volumes sélectionnés pour chaque itération

Les gains théoriques sur les autres procédures géométriques sont de l'ordre

$$g_3 = Pv(Lmoy) / Ps$$

Les gains théoriques pour les procédures physiques sont de l'ordre

$$g_4 = Pv(Lmoy) / (Ps \times NPmoy)$$

NPmoy : nombre moyen de types de particule présents

Le gain théorique maximal pour l'ensemble serait donc

$$G = \frac{100}{\frac{P_1}{-}\ \frac{P_2}{+}\ \frac{P_3}{-}\ \frac{P_4}{+}\ \frac{P_5}{-}\ +}$$
$$g_1\ \ g_2\ \ g_3\ \ g_4$$

Pi est le pourcentage de temps CPU nécessaire pour chacune des 4 phases.
P_5 est le pourcentage de temps CPU pour toutes les parties non vectorisables (E/S, construction des vecteurs ...).

Le gain global sera donc proportionnel à la longueur moyenne des vecteurs de particules.

- *Prévisions numériques des équations proposées:*

  Il semble difficile de préciser des ordres de grandeurs sans prendre le risque de se tromper. Néanmoins, l'expérience de GEANT3 nous porte à croire que pour des expériences du type de OPAL, Lmoy peut être de l'ordre de 50 à 100, NVmoy entre 5 et 50 et NPmoy entre 1 et 5, avec de telles

valeurs nous pourrions espérer, sur un Cyber 205 obtenir un facteur de plus de 10 soit l'équivalent de 100 VAX 11/780. Les résultats obtenus pour la vectorisation de la géométrie nous font penser que ces prévisions sont réalistes.

Posons
Lmoy = 100
NVmoy = 20
NPmoy = 5
P_1 = 5 % g_1 = 100
P_2 = 40 % g_2 = 30
P_3 = 20 % g_3 = 20 G = 100 / 10.38 ≈ 10
P_4 = 30 % g_4 = 10
P_5 = 5 %
(Prévision théorique pour un Cyber 205.)

Cette approche fera l'objet de recherches approfondies devant aboutir à une réalisation pour fin 1987. La politique d'extension du CERN est un des facteurs décisifs pour l'investissement de plusieurs hommes par année sur ce projet.

3.4 FORTRAN 200 et la programmation vectorielle explicite

Le compilateur FORTRAN propose par Control Data sur Cyber 205 comprend une extension vectorielle. C'est sans doute un premier pas vers un FORTRAN 8X (ou FORTRAN 9X) qui permet déjà au programmeur de contrôler les opérations vectorielles de façon explicite.

Une utilisation non judicieuse de ces nouvelles instructions vectorielles, pourrait ne procurer aucune amélioration des performances. Après une présentation rapide des parties essentielles de cette extension, nous tenterons de proposer quelques règles d'utilisations efficaces.

3.4.1 Présentation de FORTRAN 200

Un vecteur est une série de valeurs stockées dans la mémoire à des adresses contiguës [9]. Un vecteur est référencé de façon explicite ou par un descripteur de vecteur. Il est parfaitement défini par la donnée de

- son premier élément (élément d'un tableau)
- sa longueur
- le type de ses données

Exemples de vecteurs :

```
A(1;300)  = A(1)..A(300)
A(5;200)  = A(5)..A(204)
A(1:100:10) n'est pas un vecteur car données non contiguës
```

Les descripteurs de vecteurs sont des objets du langage qui doivent être affectés à des définitions explicites de vecteurs tableaux ou de vecteurs dynamiques.

Exemples de descripteurs :

```
DESCRIPTOR  VA,VB
ASSIGN VA,A(1;300)
ASSIGN VB,.DYN.200  % allocation de 200 mots consécutifs %
```

Il existe aussi un type de vecteur particulier : le "vecteur de bits". Un vecteur de bits peut être construit par affectation à une constante de type BIT ou aux résultats d'une expression relationelle entre vecteurs de même longueur.

Exemple de vecteur de bits :

```
BIT B(100)
DIMENSION A(100),C(200)
B(1;100)=A(1;100).EQ.C(101;100)
```

Les vecteurs de bits et expressions relationelles seront utilisés pour l'affectation vectorielle conditionnelle. En plus des affectations de vecteurs et des opérations de vecteurs, l'instruction vectorielle WHERE permet en fonction d'un résultat de type vecteur de bits de choisir quelles affectations seront réalisées (équivallent au IF THEN ELSE).

Exemple de WHERE :

```
WHERE ( A(1;100).GT.B(101;100) )
    C(1;100)=A(1;100)-B(101;100)
OTHERWISE
    C(1;100)=B(101;100)-A(1;100)
END WHERE
```

Il existe aussi un certain nombre de procédures vectorielles prédéfinies. Plusieurs d'entre elles sont fortement utilisées.

- *Minimum et maximum:* Plusieurs procédures permettent d'obtenir le minimum et maximum de tous les éléments d'un vecteur ou d'un sous – ensemble d'éléments identifiés par un vecteur de bits, chaque bit à 1 sélectionnant un élément.

  U : 2 4 6 3 7
  b : 0 1 1 0 1 ===> min = 4

- *Affectation conditionnelle:* L'affectation de vecteurs peut être conditionnelle pour chacun des éléments en fonction d'un vecteur de bits.

  U : 1 2 3 4
  V : 5 6 7 8
  b : 0 1 0 1 ===> V = 5 2 7 4

  c'est équivalent au programme FORTRAN

```
WHERE (b) V=U
```

- *Construction conditionnelle:* Cette opération construit un nouveau vecteur à partir de deux vecteurs différents. Les valeurs d'un vecteur de bits permettent de contrôler quelles valeurs de l'un ou l'autre des deux vecteurs seront affectées au nouveau vecteur.

  U : 1 2 3 4
  V : 5 6 7 8
  b : 0 0 1 1 ===> W = 5 6 3 4

  en FORTRAN

```
WHERE (b)
  W = U
OTHERWISE
  W = V
END WHERE
```

- *Compression:* Un nouveau vecteur est construit à partir d'un vecteur par réduction de sa longueur; seuls les éléments associés à une valeur 1 d'un vecteur de bits sont conservés.

  U : 1 2 3 4
  b : 0 1 1 0 ===> W = 2 3

- *Extension:* Un nouveau vecteur est construit à partir d'un vecteur par insertion de zéros. Le contrôle est assuré par un vecteur de bits.

```
U : 5 6
b : 0 1 0 0 1    = = >   W = 0 5 0 0 6
```

- *Merge:* Un nouveau vecteur est créé par "schuffle" de deux vecteurs, le choix est contrôlé par un vecteur de bits.

```
U : 1 2 3 4
V : 5 6 7 8
b : 0 1 1 0 1 0 1 0    = = >   W = 5 1 2 6 3 7 4 8
```

- *Gather:* Cette opération est utilisée pour créer un vecteur à partir de données dispersées dans la mémoire et pointées par une liste d'index.

```
U : 5 6 7 8 9 10 11 12
V : 3 4 1 7          = = >   W = 7 8 5 11
en FORTRAN séquentiel

      DO 10 I=1,4
   10 W(I)=U(V(I))
```

- *Scatter:* Cette opération disperse les éléments d'un vecteur vers des emplacements indiqués par une liste d'index. C'est la réciproque du Gather.

```
U : 10 11 12 13
V : 5 2 7 1
W : 1 2 3 4 5 6 7 8    = = >   W = 13 11 3 4 10 6 12 8
en FORTRAN séquentielle

      DO 10 I=1,4
   10 W(V(I))=U(I)
```

- *Comptage de bits:* Cela permet de connaître le nombre de bits à 1 pour un vecteur de bits.

```
b : 0 1 1 0 1    = = >   s = 3
```

- *Construction de vecteur de bits:* Deux procédures permettent de construire des vecteurs de bits par répétition d'un certain nombre de 0 suivi par un certain nombre de 1 (ou inversement) jusqu'à remplissage du vecteur de bits.

```
b : 0 0 0 0 0 0 0 0 0
```

a1 : 3
a2 : 1 ==> b = 0 1 1 0 1 1 0 1 1
ou 1 0 0 1 0 0 1 0 0

- *Fonctions mathématiques:* Plusieurs fonctions ont été étendues pour le traitement de vecteurs : SIN COS SQRT ABS TAN SIGN LOG EXP ...

3.4.2 Codage d'une décomposition algorithmique conditionnelle.

Dans les méthodes Montè Carlo, la génération de nombres aléatoires détermine l'évolution de chaque évènement. Aussi, dans l'algorithme séquentiel nous trouvons un grand nombre d'instruction IF .... GO TO qui permettent de sélectionner les parties de l'algorithme devant être exécutées.

Dans un Montè Carlo vectorisé, à un instant donné, certaines particules devront subir un ensemble de tâches alors que d'autres non. Il faut donc mettre en place des opérations vectorielles sélectives. Ce problème n'est pas seulement propre à ce type de méthode mais peut apparaître dans tout algorithme non trivial.

Quatres solutions peuvent résoudre ce type de problème sur Cyber 205.(fig 47) Nous allons les définir et les utiliser sur l'exemple suivant :

soit 3 vecteurs respectivement de type réel, entier et bit contenant

$U(N\times3) = (X_1\ Y_1\ Z_1\ X_2\ Y_2\ Z_2\ \ldots\ldots)$
I(N) = (1 4 7 10 13 16 19 .....)
B(N×3) = (1 0 0 1 0 0 1 0 0 .....)

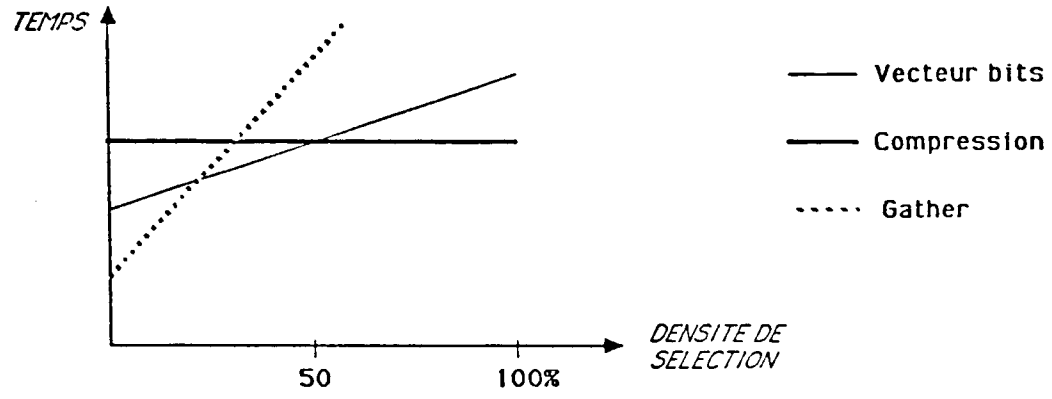
et le programme séquentiel qui calcule le minimum sur les Xi et ajoute 1 ensuite à chaque Xi

```
DO J=0,N-1
    IF ( U(3*J+1).LT.MIN ) MIN = U(3*J+1)
    U(3*J+1) = U(3*J+1) + 1
END DO
```

- *Gather – calcul – Scatter:*

Les éléments sur lesquels le calcul sera effectué sont sélectionnés à partir du vecteur de base U en fonction des adresses contenues dans J, et rangés dans un vecteur temporaire. Le calcul est alors effectué juste sur ce vecteur et les résultats sont replacés ensuite à leurs emplacements initiaux.

on obtient :



Algorithme de GVOBOX pour un vecteur de longueur 100

Figure 47: Comparaison du temps d'exécution pour 3 solutions de sélection

BU LILLE

```
TEMP = Q8VGATHR ( U , I ; N )      % U I TEMP : Descripteur %
MIN = Q8SMIN ( TEMP )              % procedure FORTRAN FT200  %
TEMP = TEMP + 1
U = Q8VSCATR ( TEMP , I ; U )
```

- *Compression – Calcul – Extension – Affectation:*

L'idée est la même que précédemment, seule la construction du vecteur temporaire et son rangement sont différents. L'utilisation du Gather – Scatter sera plus efficace lorsque les données sélectionnées seront dispersées dans la mémoire, la compression – extension lorsque les données seront localisées dans la mémoire.

on obtient

```
TEMP = Q8VCMPRS ( U , B ; N )
MIN = Q8SMIN ( TEMP )
TEMP = TEMP + 1
TEMP1 = Q8VXPND ( TEMP , B ; N*3 )
```

```
U = Q8VCTRL ( TEMP1 ,B ; N*3 )     % affectation cond. %
```

- *Opérations contrôlées par vecteur de bits:*

Pour de petits blocs d'actions conditionnels, le temps nécessaire pour les Gather – Scatter ou Compression – extension peut être supérieur au gain obtenu par la vectorisation. Dans ce cas le Cyber 205 propose un contrôle d'opérations direct par vecteur de bits. Les calculs sont effectués sur la totalité des vecteurs mais les résultats sont sélectionnés seulement si l'élément du vecteur de bits correspondant le permet.

on obtient

```
MIN = Q8SMIN ( U , B )        % les operations sont effectuees %
WHERE ( B ) U = U + 1         % sur la totalite des vecteurs %
```

- *Calcul généralisé:*

Beaucoup de blocs conditionnels dans les Montè Carlo séquentiels ne sont en fait que des optimisations des algorithmes généraux qui pour certains cas précis permettent de gagner du temps. Avec la vectorisation, il est souvent plus efficace d'appliquer les algorithmes généraux sans tenir compte de ces optimisations et ainsi ne plus avoir à sélectionner les éléments de vecteurs par Gather – Scatter ou Compression – Extension : le temps perdu par les algorithmes généraux est en règle générale inférieur au temps de construction et rangement des vecteurs.

exemple simple :

```
DO I=1,N
  IF (ALPHA(I).EQ.0.0) THEN
        T(I) = A(I)
    ELSE
        T(I) = A(I) * COS( ALPHA(I) )
  END IF
END DO
```

on obtient

```
TEMP = VCOS ( ALPHA ; TEMP )    % pas de condition %
T = A * TEMP
```

En conclusion, une vectorisation sera efficace dès l'instant où le nombre de constructions et rangements de vecteurs sera minime, le temps utilisé par ces opérations ne devra pas dépasser quelques pour cent du temps total. Ces opérations ne sont en fait nécessaires que pour préparer et récupérer les travaux effectifs de l'algorithme.

3.5 Conclusion sur la vectorisation et les machines vectorielles

La principale conclusion de ce travail est que la vectorisation de méthodes Montè Carlo comme GEANT3 est réalisable. Une première approche avec la vectorisation de la géométrie a donné des résultats meilleurs que ceux attendus. Les efforts nécessaires pour reconstruire les algorithmes généraux de simulation seront sans doute importants, mais l'espoir d'obtenir des facteurs de plus de 10 est un encouragement pour les supporter.

L'intérêt d'une telle réalisation est immédiat: de nouveaux objectifs peuvent etre maintenant définis.

- Les simulations pourront porter sur un plus grand nombre d'évènements pour un même temps CPU. Cela améliore les résultats de toutes les méthodes Montè Carlo. Pour GEANT3, cela permet de définir la géométrie pour un type d'évènement donné, d'en calculer la distribution précise d'énergie et de déceler les bruits de fond obtenus.

- Le gain en temps CPU peut aussi être utilisé pour améliorer la précision de la simulation d'un évènement. La géométrie et les processus physiques pourront être plus détaillés et les résultats de meilleure qualité.

- GEANT3 pourra être utilisé pour de nouvelles expériences qui jusqu'à présent ne pouvaient concevoir en un temps raisonnable l'obtention de résultats représentatifs.

- Le développement de nouvelles extensions de GEANT3 sera aussi plus facile à tester sur des jeux de données conséquents. Les temps nécessaires seront plus courts pour de meilleurs résultats.

CHAPITRE 4
APPROCHE DE ADA POUR GEANT3

Ada, [28] nouveau langage utilisé par le département de défense des Etats Unis, est dérivé de PASCAL et ALGOL 68. La plupart des concepts définis par Wirth ont été repris par Ichbiah et son équipe, d'autres ont été rajoutés.

- *Programmation structurée:* La programmation n'est pas seulement une technique mais beaucoup plus une mise en forme de la pensée aboutissant à une expression purement symbolique. Tout problème complexe peut être décomposé en sous – problèmes jusqu'à obtention de problèmes élémentaires directement solubles par la machine. Cette programmation descendante permet d'obtenir une clarification et une structuration des algorithmes. Le programme obtenu est ainsi plus facile à lire et aussi à comprendre, il est alors plus facile à modifier et à maintenir. Le partage du problème entre différentes équipes peut être réalisé. La plupart des programmeurs FORTRAN utilise déjà ce genre d'analyse sans le traduire de façon explicite par le langage : le passage à Ada est dans ce cas plus simple du point de vue algorithmique.

- *Langage universel:* Etant dérivé de langages scientifiques, il est évidemment aussi un langage scientifique. De plus, les possibilités de manipulation de chaînes de caractères et de fichiers font de lui un langage de gestion. La notion de tâche permet de réaliser des systèmes temps réel. La compilation séparée et la notion de modules indépendants permettent la définition de bibliothèques de sous – programmes (impossible en PASCAL).

- *Langage typé:* Comme pour pascal, tout objet doit être défini ainsi que son type avant d'être utilisé dans le programme. Les erreurs sémantiques sont détectables lors de la compilation, la cohérence des données doit être respectée pendant l'exécution. La notion de type permet de définir des objets dont les propriétés sont indépendantes de la machine et des procédures définies pour des objets de types caractérisés.

- *Langage modulaire:* Le corps d'un programme peut être décomposé en modules. Chaque module est composé de deux parties : une partie déclarative contenant les objets et les définitions des procédures visibles en dehors de ce module; et une partie facultative contenant les instructions et les objets intermédiaires de chacune des procédures visibles. Ces modules sont soit des procédures, soit des fonctions, soit des paquettages, soit des tâches. Ces définitions d'unités modulaires permettent une utilisation parfaite des concepts de programmation structurée, l'utilisateur peut construire ses programmes à partir de modules existants et déjà testés.

- *Temps réel:* Ada permet une programmation de processus parallèles associée à des possibilités de synchronisations et de communications entre ces processus. La définition de système temps réel peut être dorénavant supportée par un langage évolué et standard. Ada utilise le concept de rendez – vous permettant la synchronisation directe, la communication par message variable et l'exclusion mutuelle entre processus parallèles.

Ces propriétés devraient faire de Ada l'un des langages le plus utilisé dans les années à venir. Voyons comment un programme comme GEANT3 pourrait être reconstruit en Ada.

4.1 Les structures de données de GEANT3 en Ada

Les structures de données de GEANT3 sont gérées par le système ZEBRA(cf Chap 1). Celui – ci est utilisé pour les raisons suivantes :

- allocation dynamique de l'espace
- structuration modulaire de l'information (langage typé)
- portabilité entre plusieurs calculateurs de types différents.

La définition même de Ada devrait permettre une mise en place plus simple mais aussi plus claire que celle faite en FORTRAN. L'allocation dynamique et le type pointeur, la définition de types sont en fait des principes de base de Ada. [2] [25] Toutefois la gestion proprement dite de l'information et les algorithmes de manipulations seront les mêmes en FORTRAN qu'en Ada. Il convient de redéfinir un module équivalent à ZEBRA proposant les mêmes possibilités de gestion de structures de données complexes. Seules les idées essentielles pour gérer ces structures seront exposées, le reste n'étant plus ensuite qu'une question de programmation.

4.1.1 Notion de paquettage Ada

Pour assurer la modularité de l'information, Ada propose d'utiliser des "package". Il s'agit de réunir dans un même module, la définition de l'objet et les procédures qui le manipulent. Une interface est définie pour l'utilisateur du paquettage, seules les données et procédures présentes dans cette interface sont accessibles. Ainsi tous les mécanismes internes nécessaires à la gestion de l'objet sont invisibles en dehors du paquettage.

Souvent les mêmes algorithmes de manipulations de données sont utilisés sur des objets de types différents; telle la notion de pile, où l'on peut avoir à gérer simultanément une pile d'entiers, une pile de tâches et une pile d'adresses, les

mécanismes de dépiler et empiler sont les mêmes, seuls les objets manipulés différent. Afin d'avoir à éviter de réécrire plusieurs fois les mêmes paquettages sur des objets différents, Ada propose de générer plusieurs paquettages à partir d'un même paquettage défini pour un type formel, celui-ci étant remplacé lors de la génération par le type effectif.

4.1.2 Un paquettage par structure

La solution la plus simple et peut être aussi la plus efficace pour les structures de GEANT3, consiste à définir pour chacune d'entre elles, un paquettage contenant sa définition et les procédures qui la manipulent. L'intérêt d'une telle solution est que ces procédures peuvent être optimisées en fonction du type de structure choisie, la place mémoire utilisée peut être minimisée. En revanche il faudra répéter pour chaque structure un ensemble de mécanismes identiques. La portabilité de l'information doit être assurée par chaque paquettage, des procédures d'entrées/sorties spécifiques à chaque structure de données doivent être définiees.

Exemple de paquettage pour la structure des matériaux de GEANT3(fig 48, 6):

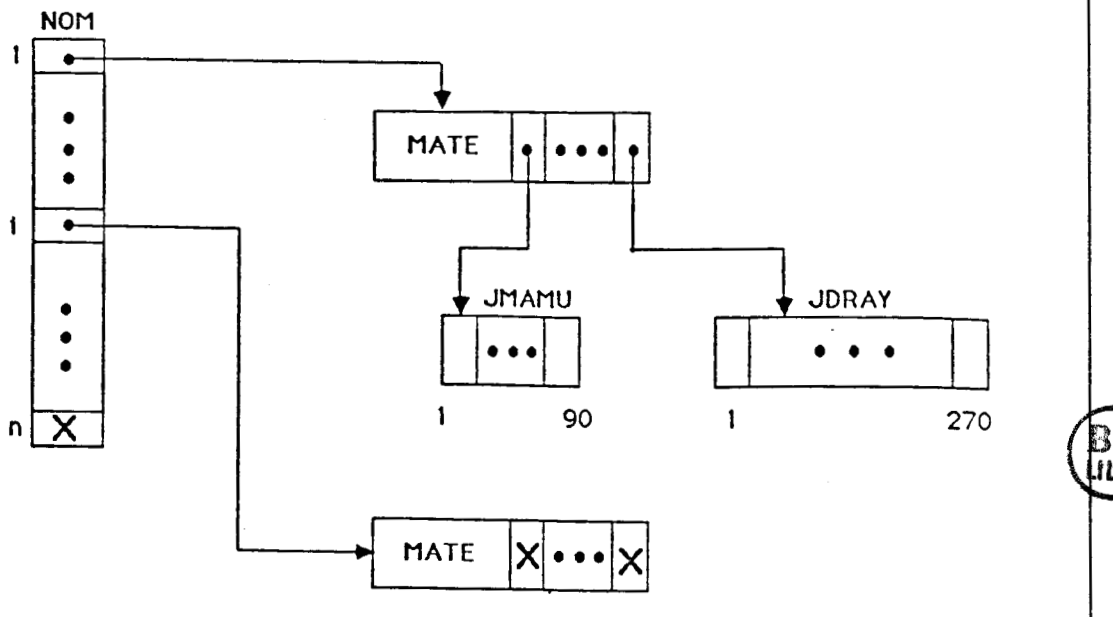


Figure 48: Structure des matériaux en Ada

BU LILLE

```
package MATE_PK is
    -- interface visible depuis l'exterieur du paquettage
    type TBFL is array ( INTEGER range <>) of FLOAT;
    type PTBFL is access TBFL;
    -- definition de l'objet
    type MATE is record
        NOM:STRING(1..20);
        A,Z,DENSITY,RADL,ABSL:FLOAT:=0.0;
    end record;
    -- construction de la structure
    type JMAT is record
        INFO:MATE;
        JMAEL,JMAMU,JPHOT,JDRAY:PTBFL:=null;   -- etc.
    end record;
    -- deux procedures de manipulation de la structure
    -- creation d'un materiau
    procedure GSMATE(N:in INTEGER;
                     MAT:in MATE);

    -- creation des proprietes physiques pour un muon
    procedure GSMAMU(N:in INTEGER;
                     PT:in PTBFL);

    -- autres procedures a definir pour cette structure
    -- procedure FZIN;FZOUT.....
end MATE_PK;

package body MATE_PK is
    TAILLE:INTEGER:=10;
    type ACCES is access JMAT;
    type T_NOM is array(INTEGER range<>) of ACCES;
    type P_NOM is access T_NOM;
    -- gestion dynamique d'une table de pointeurs vers les materiaux
    P:P_NOM:=new T_NOM'(1..TAILLE=>null);

    procedure GSMATE(N:in INTEGER;
                     MAT:in MATE) is
        Q:P_NOM:=null;
    begin
        if N>TAILLE
          then
            Q:=new T_NOM(1..TAILLE+10);
            Q.all(1..TAILLE):=P.all(1..TAILLE);
            TAILLE:=TAILLE+10;
            P:=Q;
        end if;
```

```
        -- creation et mise a jour d'un element
        P.all(N):=new JMAT;
        P.all(N).INFO:=MAT;
    end GSMATE;

    procedure GSMAMU(N:in INTEGER;
                     PT:in PTBFL) is
    begin
        -- lien avec l'information pour les muons
        P.all(N).JMAMU:=PT;
    end GSMAMU;

    -- procedure FZIN,FZOUT ...  a definir pour chaque paquettage
    -- definition des procedures presentes dans l'interface
end MATE_PK;
```

Utilisation du paquettage depuis une procédure

```
-- utilisation du paquettage MATE_PK
with MATE_PK;
use MATE_PK;
procedure GMATE is
--
--    ***********************************************************************************
--    *                                                                                 *
--    *  Definition des materiaux standards de GEANT3                                   *
--    *                                                                                 *
--    *                                                                                 *
--    ***********************************************************************************
--
    HELIUM:MATE;
    -- initialisation a la creation
    LITHIUM:MATE:=("LITHIUM              ",6.94,3.0,0.534,155.0,120.6);
    -- creation d'un tableau dynamique pour les muons
    JMAMU:PTBFL:=new TBFL(1..90);
begin
    -- acces a chaque composant d'un materiau
    HELIUM.NOM:="HELIUM              ";
    HELIUM.A:=4.0;
    HELIUM.Z:=2.0;
    HELIUM.DENSITY:=0.125;
    HELIUM.RADL:=755.0;
    HELIUM.ABSL:=478.0;
    -- creation de 2 materiaux
    GSMATE(3,HELIUM);
    GSMATE(4,LITHIUM);
```

```
        -- acces indirect au tableau dynamique pour initialisation
        for I in 1..90 loop
            JMAMU.all(I):=1.0;
        end loop;
        -- rangement des caracteristiques physiques (muon dans lithium)
        GSMAMU(4,JMAMU);
    end GMATE;
```

Cette construction n'est qu'une approche du paquettage de la structure des matériaux, beaucoup d'autres procédures devraient être écrites mais la structure de données elle-même est parfaitement définie. Cette structure permet de minimiser l'allocation d'espace mémoire, la solution choisie tient compte des propriétés de la structure. De plus une partie des mécanismes mis en place sont inaccessibles en dehors du paquettage : c'est le cas pour la table des noms, cette gestion ne présentant aucun intérêt pour l'utilisateur de ce paquettage. Les procédures d'entrées/sorties doivent être définies en fonction des caractéristiques de la structure, un indicateur devra être placé en début d'enregistrement pour permettre d'identifier le lien entre la structure lue et le paquettage la manipulant. Un système de reconnaissance de l'information devra être construit pour assurer la reconstruction de plusieurs structures stockées dans un même fichier.

Pour chaque structure de GEANT3 on peut définir un tel paquettage.

Bien que cette solution de paquettage par structure soit parfaitement réalisable et sous certains points intéressante (allocation minimum), la construction proprement dite demandera beaucoup de travail souvent répétitif. L'aspect statique des structures de données peut être un handicap pour la maintenabilité des logiciels. L'utilisation du paquettage générique devrait être mieux adaptée à la construction de toutes ces structures.

4.1.3 Un paquettage générique de gestion de structure de données

Le paquettage générique que nous voulons construire doit permettre de gérer n'importe quel type de structures relationelles, d'assurer la sauvegarde et la reconstruction d'informations et de manipuler un nombre indéfini d'objets.

4.1.3.1 Définition de l'information

Une structure de données est composée d'informations de deux types :

- *Informations qualitatives:* c'est l'ensemble des données sur lesquelles les calculs seront exécutés.

- *Informations structurelles:* C'est l'ensemble des liens existant entre une information qualitative donnée et un sous-ensemble de toutes les autres informations qualitatives.

Une structure est en fait un graphe où chaque noeud représente une information qualitative et où toutes les flèches partant de ce noeud représentent l'information structurelle associée.

Exemple d'information : définition d'un automate déterministe sur N

```
type NOEUD;
type P_ND is access NOEUD;
type PT_NOEUD is array ( INTEGER range <> ) of P_ND;
type NOEUD is record
        NOM : CHARACTER ;
        FLECHE : PT_NOEUD;
end record
```

L'information qualitative est la lettre associée au noeud, l'information structurelle est la liste des flèches partant vers d'autres lettres.

Une structure de données est dite close si elle est formée d'un ensemble d'informations qualitatives et d'un ensemble d'informations structurelles où chaque lien ne référence que des informations contenues dans cette structure.

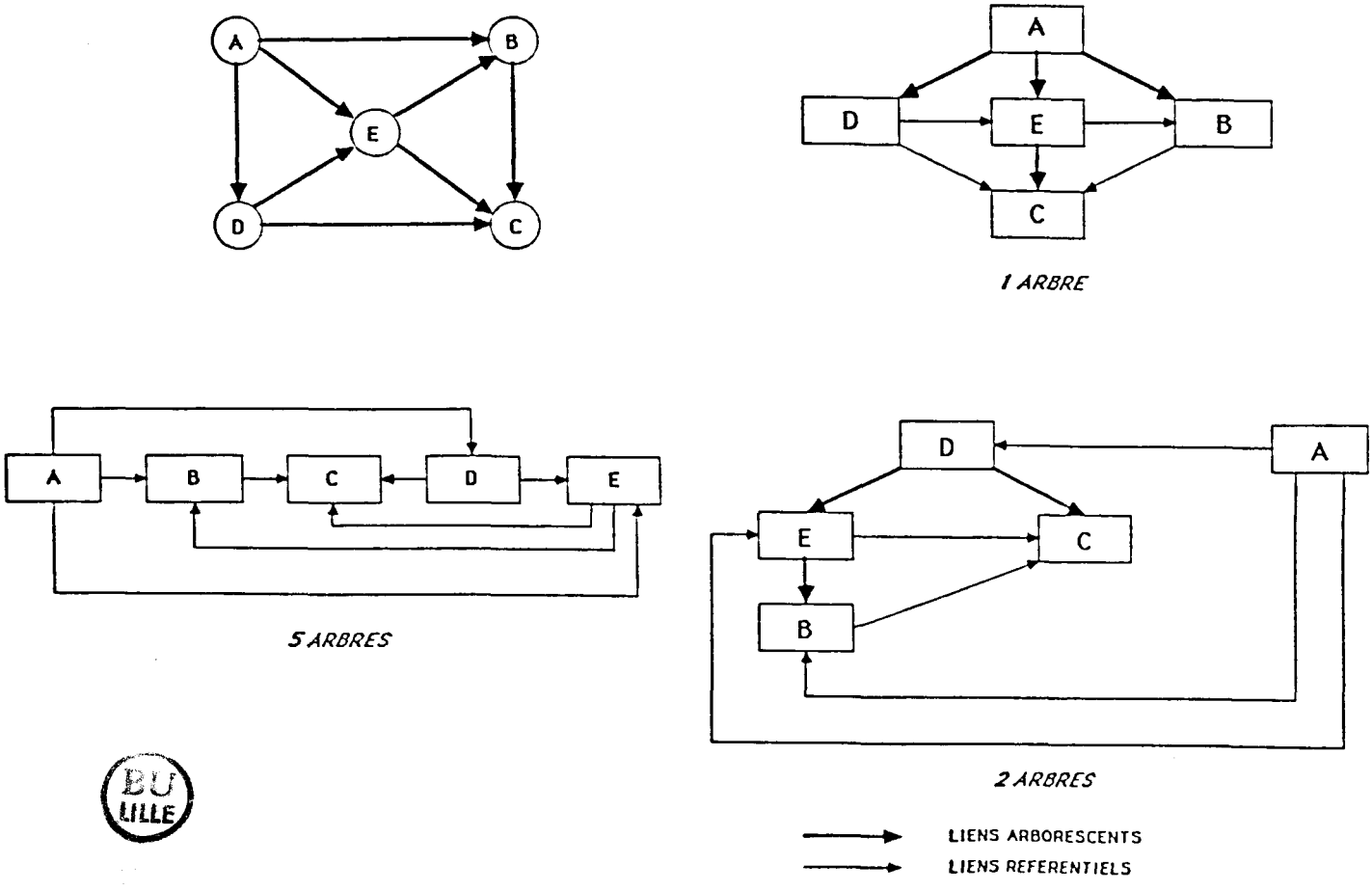
4.1.3.2 Construction structurelle du paquettage

Les structures de données manipulées par GEANT3 sont en fait des arbres où certains noeuds référencent un ou plusieurs noeuds non directement accessibles par la construction arborescente.

Toute structure de graphe peut être construite à partir d'une ou plusieurs structures arborescentes, sur lesquelles on peut rajouter d'autres liens entre deux noeuds quelconques de ces arbres. Plusieurs décompositions peuvent être choisies pour un même graphe, la solution la plus simple étant de définir un arbre pour chaque noeud du graphe, tous les liens sont alors des liens rajoutés (fig 49).

Pour permettre de définir de telles structures de données, quatre types d'information structurelle sont nécessaires (fig 50) :

- *PERE:* chaque noeud construit est soit une racine d'un arbre: alors PERE vaut "nul", soit il est positionné dans l'arbre et l'adresse de son père est alors conservée.
- *FILS:* si un noeud posséde des descendants, l'un d'entre eux est accessible directement par ce lien. L'arbre est représenté par l'intermédiaire de liens horizontaux et liens verticaux [24].
- *FRERE:* si un noeud posséde plusieurs descendants, l'accès au premier est direct, pour les autres il faut y accéder à partir du premier par le lien FRERE.



BU LILLE

Figure 49: Extraction des sous – arbres d'un graphe

- *REF:* Pour permettre d'autres références que celles proposées par la structure d'arbre, un ou plusieurs liens référentiels peuvent être définis. Ces liens peuvent pointer vers tout noeud de n'importe quel arbre.

4.1.3.3 Construction qualitative du paquettage

Le type d'information manipulée peut être très varié, cela peut aller des types simples comme les entiers, aux types composés. Comme la construction structurelle est la même pour tous les objets manipulés, les seuls paramètres génériques du paquettage ne seront donc dépendants que de ces objets.

Deux types de générations sont envisageables :

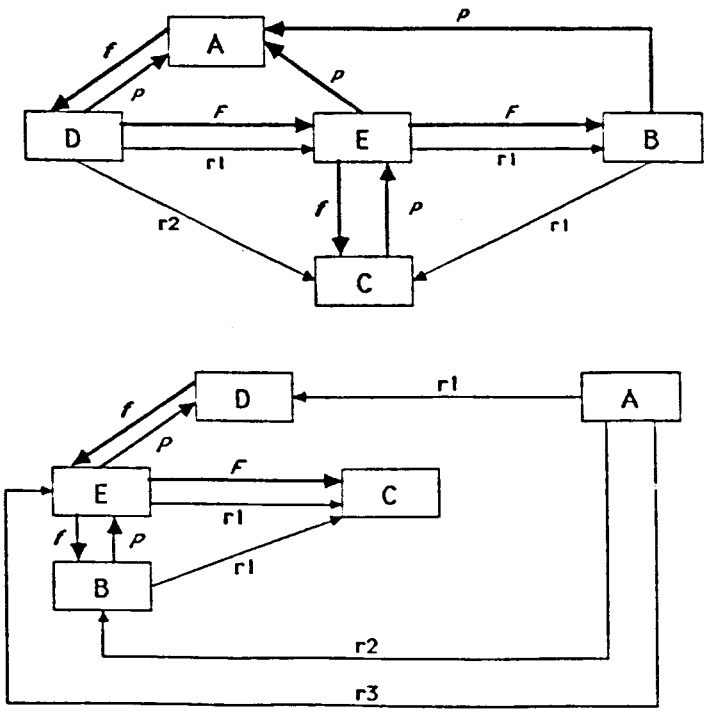


Figure 50: Exemple d'utilisation de l'information structurelle

BU LILLE

- *Par itération:*

Un paquettage est instantié pour chaque type d'objet manipulé. Dans ce cas le nombre d'objets différents peut être infini, il suffit de générer autant de paquettages qu'il y a de types. Lors de l'instantiation, le paquettage crée un nouveau type contenant l'information structurelle et qualitative. Pour cela, il faut connaître les types des objets pointés par cette information structurelle, qui ne sont autres que des types construits par d'autres paquettages. Or Ada ne permet pas les références croisées entre deux types générés par deux paquettages différents, puisqu'un type servant de paramètre effectif doit être complètement spécifié lors de l'instantiation. Les seules structures de données manipulables sont alors de type arborescent, où le type créé par l'instantiation du paquettage de niveau i sert de paramètre lors de l'instantiation du paquettage de niveau i – 1. Les liens de références doivent alors être indirects : au lieu d'utiliser l'adresse d'un objet, on utilise son nom pour le référencer.

Exemple de définition d'arbre :

```
-- creation d'un arbre de 4 niveaux
-- en commencant par les feuilles
package NIV_4 is new FEUILLE(TYP_4);
use NIV_4;
-- soit TYP_CREE le type cree par le package
-- il est utilise pour la generation du niveau 3
package NIV_3 is new NOEUD(TYP_3,NIV_4.TYP_CREE);
use NIV_3;
package NIV_2 is new NOEUD(TYP_2,NIV_3.TYP_CREE);
use NIV_2;
package NIV_1 is new NOEUD(TYP_1,NIV_2.TYP_CREE);
use NIV_1;
-- definition d'une variable representant une racine
RACINE:NIV_4.TYP_CREE; -- etc
```

- *Par unification:*

  Un seul paquettage est créé pour tous les types utilisés. Un macro – type est construit à partir des autres, cela correspond au type "uni" de ALGOL 68, on peut le simuler en Ada en utilisant un indicateur du type d'information. Une valeur différente pour cet indicateur est définie pour chaque type de données présent dans la structure. Puis il suffit de construire un type enregistrement ne comprenant qu'une partie variable sur un discriminant précisant cet indicateur. (cf 4.1.3.5) Les pointeurs sont ainsi banalisés, n'importe quel type de structure est alors constructible. La génération du macro – type peut être assurée par l'utilisateur du paquettage et dans ce cas le nombre de types peut être infini, ou il peut être construit par le paquettage lui même mais le nombre de types est alors limité au nombre fini de paramètres génériques du paquettage. Seule cette solution permet de construire toutes structures de données.

4.1.3.4 Les entrées/sorties

Les procédures de sauvegarde et de restructuration de l'information doivent etre assurées par le paquettage. Pour cela deux fichiers sont nécessaires :

- Le premier contient l'information qualitative. Il est créé par le paquettage en fonction du macro – type utilisé lors de l'instantiation. Ce type de données doit être monolithique, il ne peut y avoir de pointeurs vers d'autres données (définition de l'information qualitative). Un fichier séquentiel permet de sauver et de relire une ou plusieurs structures closes. Un fichier à accès direct

permet la même chose mais de plus, il permet de gérer l'information de façon virtuelle, soit elle est en mémoire et donc directement accessible à partir du noeud structurel, soit elle est sur disque et son numéro d'accès est contenu dans le noeud, une lecture sur disque fera remonter l'information en mémoire centrale quand elle sera référencée. Une procédure récursive permet de balayer toute une structure close à partir d'un noeud et de sauvegarder l'information qualitative de chaque noeud sur disque. Ce fichier ne pourra en aucun cas être dissocié de la définition du macro – type

- Le deuxième contient l'information structurelle. Il est construit lors de la sauvegarde d'une structure close et doit en permettre la reconstruction lors de sa lecture. Un fichier séquentiel d'entier est construit par balayage récursif de la structure. Une table intermédiaire est nécessaire pour la reconstruction des liens de références, une lecture séquentielle de ce fichier en parallèle avec un accès direct au fichier de données qualitatives permet, en un seul passage de reconstruire la structure close.

Pour construire une procédure d'impression de structure close, l'utilisateur doit définir la procédure d'impression d'un objet du macro – type, le paquettage ne peut en effet connaître le type des informations contenues dans ce macro – type. Cette procédure d'impression devient aussi un des paramètres du paquettage générique. L'impression est ensuite assurée par une procédure récursive qui balaye complètement la structure.

4.1.3.5 Réalisation en Ada

Voici une présentation de paquettage générique qui permet de manipuler ces types de structures de données. Seules quelques procédures ont été définies.

```
generic
-- paramatres generiques definissant le Macro-type
type DATA is private;
type PT_DATA is access DATA;
-- procedure d'impression d'un objet de type DATA
with procedure PUT(VX:in PT_DATA);
package ZEBRA is
-- definition de la structure de l'information graphique
    type NODE;
    type PT_NODE is access NODE;
    type T_REF is array(INTEGER range <>) of PT_NODE;
    type P_REF is access T_REF;
    type NODE is record
        DATA:PT_DATA:=null;   -- acces aux donnees qualitatives
        PERE,FRERE,FILS:PT_NODE:=null;
        REF:P_REF:=null;
        E_S:INTEGER:=0;       -- indicateur de balayage
        INDEX:INTEGER:=0;     -- index pour acces direct
```

```
    end record;
    -- procedure de creation d'un noeud
    procedure MZBOOK(PT:out PT_NODE;
                     PERE:in PT_NODE:=null;
                     NREF:in INTEGER:=0;
                     PDA:in PT_DATA:=null);
    -- lecture d'une structure close
    procedure FZIN(PT:out PT_NODE);
    -- ecriture d'une structure close
    procedure FZOUT(PT:in out PT_NODE);
    -- impression d'une structure close
    procedure PRINT(PT:in out PT_NODE);
    -- creation des fichiers d'entrees/sorties
    procedure CREATION;
end ZEBRA;

with DIRECT_IO,SEQUENTIAL_IO;
package body ZEBRA is
    -- instantiation pour fichier a acces direct
    package SGBD is new DIRECT_IO(DATA);
    -- instantiation pour fichier a acces sequentiel
    package TABLE is new SEQUENTIAL_IO(INTEGER);
    use TABLE;
    use SGBD;
    -- definition des deux fichiers
    TIND:TABLE.FILE_TYPE;
    BASE:SGBD.FILE_TYPE;

    procedure MZBOOK(PT:out PT_NODE;
                     PERE:in PT_NODE:=null;
                     NREF:in INTEGER:=0;
                     PDA:in PT_DATA:=null) is
        P:PT_NODE:=new NODE;
    begin
        if PERE/=null
          then
            -- insertion dans la structure de donnees existante
            P.PERE:=PERE;
            P.FRERE:=P.FILS;
            PERE.FILS:=P;
        end if;
        P.DATA:=PDA;
        if NREF>0
          then
            -- creation dynamique des liens de references
            P.REF:=new T_REF'(1..NREF=>null);
        end if;
```

```
    PT:=P;
end MZBOOK;

procedure PRINT( PT:in out PT_NODE) is
-- procedure recursive d'impression
begin
    -- E_S permet de savoir si le noeud a deja ete imprime
    if PT.E_S=0
      then
        PT.E_S:=1 ;
        if PT.DATA/=null
          then
            PUT(PT.DATA);
        end if;
        if PT.FILS/=null
          then
            PRINT(PT.FILS);
        end if;
        if PT.FRERE/=null
          then
            PRINT(PT.FRERE);
        end if;
        if PT.REF/=null
          then
            for I in 1..PT.REF.all'LAST loop
                if PT.REF.all(I)/=null
                  then
                    PRINT(PT.REF.all(I));
                end if;
            end loop;
        end if;
    end if;
end PRINT;

procedure FZDAT( P:in out PT_NODE) is
-- vidage des donnees qualitatives sur fichier
    PT:PT_NODE:=P;
begin
    if PT.INDEX=0
      then
        if PT.DATA/=null
          then
            -- mise a jour du numero d'enregistrement
            PT.INDEX:=INTEGER(SGBD.INDEX(BASE)) ;
            -- ecriture dans le fichier direct
            WRITE(BASE,PT.DATA.all);
            PT.DATA:=null;
```

```
          else
            -- pas d'information qualitative
            PT.INDEX:=-1;
        end if;
        -- meme recursivite que pour PRINT
        -- ..............
        -- ..............
    end if;
end FZDAT;

procedure FZTBL( PT:in out PT_NODE) is
-- vidage des donnees structurelles sur fichier
begin
   if PT/=null and E_S/=1
     then
        -- ecriture de l'index dans le fichier sequentiel
        WRITE(TIND,PT.INDEX);
        E_S:=1;
        -- recursivite sur la structure
        FZTBL(PT.FILS);
        FZTBL(PT.FRERE);
        if PT.REF/=null
          then
            -- ecriture du nombre de liens de references
            WRITE(TIND,PT.REF.ALL'LAST);
            for I in 1..PT.REF.ALL'LAST loop
                FZTBL(PT.REF.all(I));
            end loop;
          else
            -- pas de lien de reference
            WRITE(TIND,0);
        end if;
      else
        if PT=null
          then
            -- pas de noeud
            WRITE(TIND,0);
        end if;
    end if;
end FZTBL;

procedure FZOUT (PT:in out PT_NODE) is
begin
    FZDAT(PT);
    TABLE.OPEN(TIND,OUT_FILE,"index.dat");
    FZTBL(PT);
    CLOSE(TIND);
```

```
end FZOUT;

procedure FZIN(PT:out PT_NODE) is
-- reconstruction d'une structure de donnees close
    type ENRG is record
        NUM:INTEGER:=-1;
        ADR:PT_NODE:=null;
    end record;
    type TABL is array(1..100) of ENRG;
    T:TABL;
    P:PT_NODE:=PT;
    N:INTEGER:=0;
begin
    READ(TIND,N);
    if N=0
      then
        PT:=null;
      else
        P:=new NODE;
        if N/=-1
          then
            -- recherche dans table si l'enregistrement N a deja
            -- ete lu, sinon lecture, insertion dans la table, m.a.j.
          else
            P.DATA:= null;
        end if;
        PT:=P;
        -- recursivite
        FZTBL(PT.FILS);
        FZTBL(PT.FRERE);
        READ(TIND,N);
        if N/=0
          then
            PT.REF:=new T_REF'( 1..N=>null);
            -- lecture des liens de references
            for I in 1..PT.REF.ALL'LAST loop
                FZTBL(PT.REF.all(I));
            end loop;
        end if;
    end if;
end FZIN;

procedure CREATION is
-- creation des fichiers contenant les donnees
begin
    SGBD.CREATE(BASE,INOUT_FILE,"base.dat","size 360");
    TABLE.CREATE(TIND,OUT_FILE,"index.dat");
```

BU LILLE

```
            TABLE.CLOSE(TIND);
        end CREATION;
    end ZEBRA;
```

Exemple d'utilisation sur une structure de graphe(fig 51) avec trois types d'objets différents.

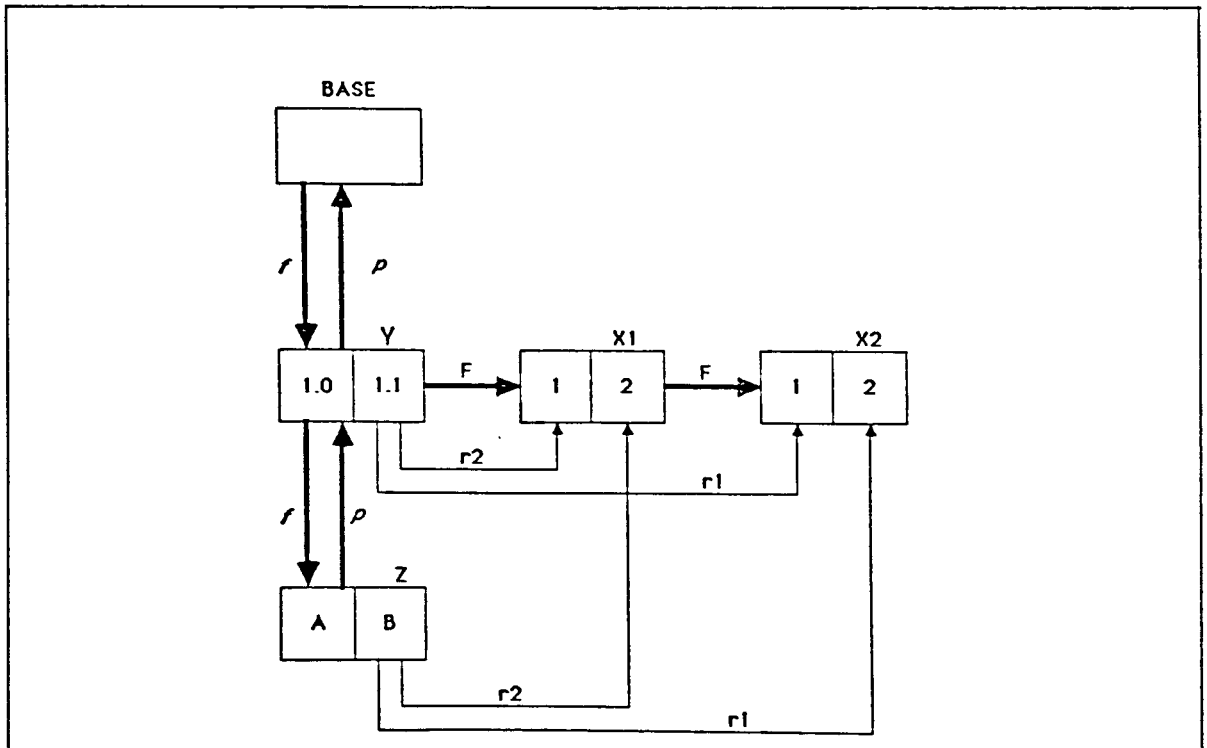


Figure 51: Exemple de graphe

BU LILLE

```
-- utilisation des paquettages ZEBRA et impression TEXT_IO
with ZEBRA,TEXT_IO;
use TEXT_IO;
procedure EXZEB is
    -- definition des trois types manipules
    -- peut etre fait dans un autre paquettage
    type X is record
```

```
    A,B:INTEGER;
end record;
type Y is record
    C,D:FLOAT;
end record;
type Z is record
    E,F:CHARACTER;
end record;
-- liste des indicateurs de type
type LISTE is (TX,TY,TZ);
-- definition du macro-type
type DATA(XYZ:LISTE) is record
    case
        XYZ is
        when TX => DX:X;
        when TY => DY:Y;
        when TZ => DZ:Z;
    end case;
end record;
-- pointeur vers le macro-type
type PT_DATA is access DATA;
P_D:PT_DATA:=null;
-- definition des procedures d'ecriture elementaire pour chaque type
-- utilisation de l'overloading

procedure PUT(V:in X) is
    package INT_IO is new INTEGER_IO(INTEGER);
    use INT_IO;
begin
    PUT(V.A);
    PUT(V.B);
end PUT;

procedure PUT (V:in Y) is
    package FL_IO is new FLOAT_IO(FLOAT);
    use FL_IO;
begin
    PUT(V.C);
    PUT(V.D);
end PUT;

procedure PUT (V:in Z) is
begin
    PUT(V.E);
    PUT(V.F);
end PUT;
-- procedure d'ecriture pour le macro type
```

```
    procedure PUT(VX:in PT_DATA) is
        package ENU_IO is new ENUMERATION_IO(LISTE);
        use ENU_IO;
    begin
        PUT(VX.XYZ);
        case
            VX.XYZ is
            when TX=>PUT(VX.DX);
            when TY=>PUT(VX.DY);
            when TZ=>PUT(VX.DZ);
        end case;
    end PUT;
    -- generation de la structure
    package STRUCTURE is new ZEBRA(DATA,PT_DATA,PUT);
    use STRUCTURE;

    procedure LEVEL is
    -- un niveau necessaire pour definir des variables apres
    -- la definition de procedures
        PX1,PX2,PY,PZ,BASE:PT_NODE:=null;
    begin
        MZBOOK(BASE);              --creation de la base sans donnees
        P_D:=new DATA'(TX,(1,2));--choix du type et initialisation
        MZBOOK(PX1,BASE,0,P_D);  --creation de 2 fils de BASE de type X
        MZBOOK(PX2,BASE,0,P_D);
        P_D:=new DATA'(TY,(1.0,1.1));
        MZBOOK(PY,BASE,2,P_D);   --creation d'un fils de BASE de type Y
        PY.REF.all:=(PX1,PX2);   --m.a.j des references
        P_D:=new DATA'(TZ,('A','B'));
        MZBOOK(PZ,PY,2,P_D);     --creation d'un fils de PY de type Z
        PZ.REF.all:=(PX1,PX2);   --m.a.j des references
        PRINT(BASE);             --impression de la structure BASE
    end LEVEL;
begin
    LEVEL;
end EXZEB;
```

Exemple d'utilisation avec la structure de matériaux de GEANT3(fig 52)

```
with ZEBRA,TEXT_IO;
use TEXT_IO;
procedure EXZEB is
    -- definition d'un type qualitative
    type MAT is record
        NOM:STRING(1..20);
        A,Z,DENSITY,RADL,ABSL:FLOAT:=0.0;
```

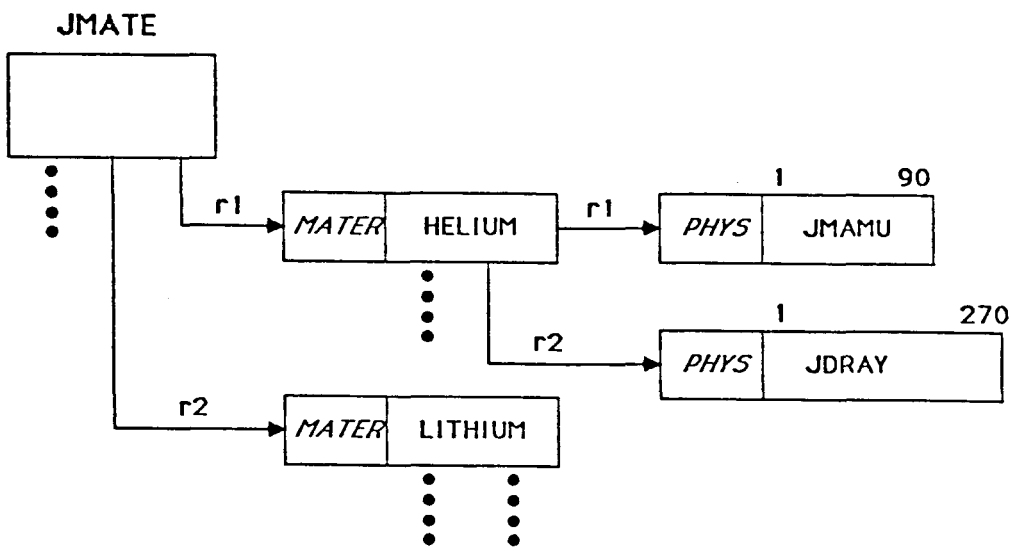


Figure 52: Construction structurelle des matériaux de GEANT3

BU LILLE

```
end record;
type TBFL is array(INTEGER range <>) of FLOAT;
type PTBFL is access TBFL;
-- liste  des types utilises pour definir le macro_type
type LISTE is (MATER,MAMU,DRAY); -- etc ...
-- definition du macro_type
type DATA(TYP:LISTE) is record
    case
        TYP is
        when MATER => DMAT:MAT;
        when MAMU => JMAMU:TBFL(1..90);
        when DRAY => JDRAY:TBFL(1..270);
    end case;
end record;
type PT_DATA is access DATA;
P_D:PT_DATA:=null;

procedure PUT(V:in MAT) is
    -- impression d'un materiau
    package FL_IO is new FLOAT_IO(FLOAT);
```

```
    use FL_IO;
begin
    PUT(V.NOM);
    PUT(V.A);
    PUT(V.Z);
    PUT(V.DENSITY);
    PUT(V.RADL);
    PUT(V.ABSL);
    PUT_LINE(" ");
end PUT;

procedure PUT (V:in TBFL) is
    -- impression d'un mecanisme physique
    package FL_IO is new FLOAT_IO(FLOAT);
    use FL_IO;
begin
    for I in 1..V'LAST loop
        PUT(V(I));
    end loop;
    PUT_LINE(" ");
end PUT;

procedure PUT(VX:in PT_DATA) is
    -- impression du macro-type
    package ENU_IO is new ENUMERATION_IO(LISTE);
    use ENU_IO;
begin
    PUT(VX.TYP);
    case
        VX.TYP is
        when MATER=>PUT(VX.DMAT);
        when MAMU=>PUT(VX.JMAMU);
        when DRAY=>PUT(VX.JDRAY);
    end case;
end PUT;
-- creation de la structure generale
package STRUCTURE is new ZEBRA(DATA,PT_DATA,PUT);
use STRUCTURE;

procedure LEVEL is
    JMATE:PT_NODE:=null;
    JMAMU:TBFL(1..90):=(1..90=>2.9);
    JDRAY:TBFL(1..270):=(1..100=>1.12,others=>0.0);
    HELIUM:MAT:=("helium      ",4.0,2.0,0.125,755.0,478.0);
    LITHIUM:MAT:=("lithium     ",6.94,3.0,0.534,155.0,120.6);
begin
    MZBOOK(JMATE,null,10);
```

```
            P_D:=new DATA'(MATER,HELIUM);
            MZBOOK(JMATE.REF.all(1),null,2,P_D);
            P_D:=new DATA'(MATER,LITHIUM);
            MZBOOK(JMATE.REF.all(2),null,0,P_D);
            P_D:=new DATA'(PHYS,JMAMU);
            MZBOOK(JMATE.REF.all(1).REF.all(1),null,0,P_D);
            P_D:=new DATA'(PHYS,JDRAY);
            MZBOOK(JMATE.REF.all(1).REF.all(2),null,0,P_D);
            PRINT(JMATE);
            CREATION;
            FZOUT(JMATE);
        end LEVEL;
    begin
        LEVEL;
    end EXZEB;
```

4.2 Multi – tâches et GEANT3

Ada est l'un des premiers langages à proposer une programmation parallèle explicite par processus parallèles et transportable d'une machine à une autre. Ceci est concrétisé en Ada par la notion de tâche, les mécanismes de communication entre ces tâches sont de type rendez – vous [19].

Les algorithmes de GEANT3 présentent des aspects intéressants pour le multi – tâches :

- Répétition des mêmes algorithmes sur plusieurs jeux de données
- Décomposition du programme en tâches successives

4.2.1 Génération arborescente de tâches

Pour chaque particule suivie, il peut y avoir à simuler d'autres particules secondaires. Celles – ci subiront ensuite les mêmes algorithmes, et pourront aussi générer d'autres particules et ainsi de suite. Les particules secondaires générées ne pouvant être simulées directement, elles sont empilées pour stockage temporaire. En Ada, il suffit de créer une nouvelle tâche pour chacune des particules générées, celle – ci pouvant elle aussi créer d'autres tâches secondaires. Une génération arborescente de tâches peut ainsi être élaborée, en fonction des résultats de simulation.(fig 53)

Cette génération de tâches peut être appliquée sur tous les évènements simultanément, ainsi si le nombre de processeurs Ada est assez grand, le parallélisme obtenu sera maximal.

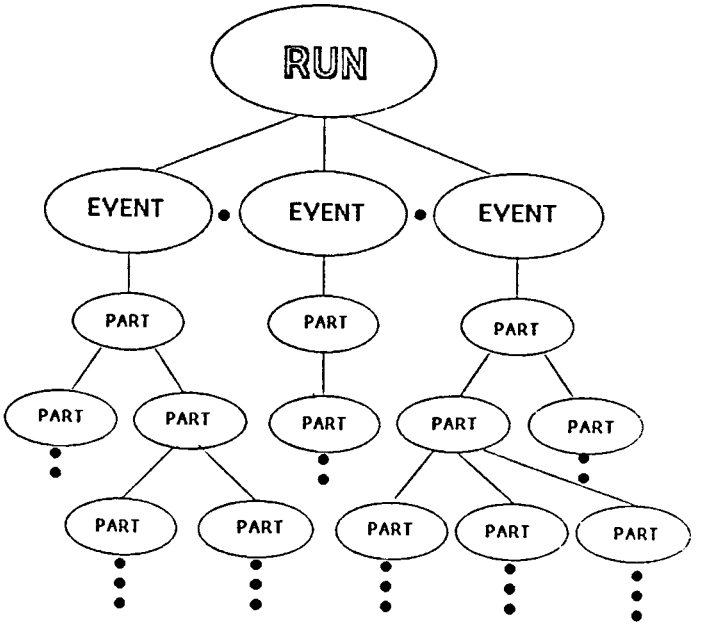


Figure 53: Génération arborescente de tâches

Les règles de visibilité Ada permettent aux tâches créées depuis une même tâche, de partager les données définies au niveau de cette tâche génératrice. Aussi peut-on définir deux types distincts d'informations :

- *A usage libre:* accessible seulement en lecture. Toutes les tâches l'utilisant peuvent alors y accéder à tout moment.
- *A usage contrôlé:* accessible aussi en écriture. La tâche génératrice doit en assurer la sécurite d'accès au travers d'une section critique définie à l'aide de rendez-vous. [21] Une seule tâche peut accéder à l'information à un instant donné.

Ainsi les structures définissant l'environnement seront communes à toutes les tâches présentes. Les structures liées à un évènement seront accessibles à toutes les tâches assurant le suivi d'une particule générée depuis cet évènement, elles pourront devenir des ressources critiques pour ces tâches secondaires. Les informations liées à une particule seront locales à la tâche chargée de son suivi.

Idée de définition des tâches de GEANT3

```
with TEXT_IO;
```

```
use TEXT_IO;
procedure GEANT is
    -- definition de l'environnement experimental
    pragma TIME_SLICE(0.01);
    type T_EVENT;
    type P_EVENT is access T_EVENT;

    task type T_EVENT is
        entry GO(PEV:in P_EVENT);
        entry SEC_CRITIQUE;
    end T_EVENT;
    P:P_EVENT:=null;
    NBEVENT : INTEGER :=10;

    task type T_PART is
        entry GO(PEVENT:in P_EVENT;
                 TYP_PART:in INTEGER;
                 X,Y,Z:in FLOAT);
    end T_PART;
    type P_PART is access T_PART;
    ILAST:INTEGER:=123;

    function RDM(N:in INTEGER)return INTEGER is
        -- fonction aleatoire pour test
    begin
        ILAST:=(ILAST*13456-34456) mod N;
        return ILAST;
    end RDM;

    procedure CREAT_TASK(PEV:in P_EVENT;
                         TP:in INTEGER;
                         X,Y,Z:in FLOAT)is
        P:P_PART:=new T_PART;
        -- creation d'une tache pour une particule secondaire
    begin
        P.all.GO(PEV,TP,X,Y,Z);
    end CREAT_TASK;

    task body T_PART is
        -- definition de l'information associee a une particule
        TYP_PART,TYP_2 : INTEGER:=0;
        X,Y,Z,X2,Y2,Z2 : FLOAT:=0.0;
        GENER,END_OF_LIFE : BOOLEAN:=FALSE;
        PEV : P_EVENT:=null;
    begin
        PUT("TPART GO");
        accept GO(PEVENT:in P_EVENT;
```

```
                TYP_PART:in INTEGER;
                X,Y,Z:in FLOAT) do
            PEV:=PEVENT;
        end GO;
        loop
            -- suivi de la particule
            GENER:=RDM(50)>45;  -- autre particule generee ?
            if GENER
              then
                CREAT_TASK(PEV,TYP_2,X2,Y2,Z2);
            end if;
            -- ecriture dans la zone commune a toutes les particules
            PEV.all.SEC_CRITIQUE;
            -- fin du suivi de la particule
            END_OF_LIFE:=RDM(100)<20; -- arret de la particule ?
            exit when END_OF_LIFE;
        end loop;
        PUT("PART FIN");
    end T_PART;

    task body T_EVENT is
        PPART : P_PART:=new T_PART;
        PEVENT : P_EVENT:=null;
        -- definition des structures liees a un evenement
    begin
        PUT_LINE("EVENT GO");
        accept GO(PEV:in P_EVENT) do
            -- stockage de l'adresse de la tache elle-meme, elle sera
            -- transmise aux taches secondaires pour permettre d'acceder
            -- aux donnees liees a l'evenement
            PEVENT:=PEV;
        end GO;
        -- creation d'une particule ( GUKINE )
        PPART.all.GO(PEVENT,1,0.0,1.0,10.0);
        loop
            select
                accept SEC_CRITIQUE do
                    PUT(" sec crit ");
                    -- section critique
                end SEC_CRITIQUE;
              else
                exit when PPART'TERMINATED;
            end select;
        end loop;
        PUT_LINE("EVENT FIN");
    end T_EVENT;
```

```
begin
    PUT("GRUN");
    -- creation de 100 evenements
    for I in 1..100 loop
        P:=new T_EVENT;
        P.all.GO(P);
    end loop;
    PUT("GRUN END");
end GEANT;
```

La gestion des tâches ainsi créées est transparente pour l'utilisateur. Ce type de programme peut être exécuté sur n'importe quel type de machine : séquentielle ou multi – processeurs. Le système d'exploitation se chargera du partage des processeurs entre les tâches, ainsi que des communications entre celles – ci. Sur une machine séquentielle, il se peut toutefois que l'exécution d'un grand nombre de tâches parallèles provoque un écroulement des performances à cause de changements de contextes trop fréquents. (un "time slice" assez grand devra être choisi)

4.2.2 Multi – tâches et pipe – line

Lors du suivi d'une particule, un certain nombre de travaux indépendants et consécutifs sont entrepris. Pour affiner l'utilisation du multi – tâches précédemment défini, on pourrait définir une tâche pour chacun de ces travaux. Chaque tâche reçoit un jeu de données, exécute la partie de travail qui lui est propre, puis envoie les résultats vers la tâche suivante. Ainsi un macro pipe – line peut être construit à partir de plusieurs tâches. Lorsqu'une tâche a envoyé les informations qu'elle a produites, elle peut directement passer au jeu de données suivant. Le débit de ce pipe – line dépendra du temps d'exécution de la tâche la plus longue.

Plusieurs tâches chargées du suivi de particules pourront utiliser le même pipe – line pour réaliser les mêmes travaux consécutifs sur les particules. (fig 54) Cette implémentation de tâches en macro pipe – line devrait permettre de définir une architecture multi – processeurs où chacun serait spécialisé pour un type de travail.

Exemple : calcul de $X^3 / 5 + 1$ en trois phases distinctes pour un vecteur de données ;

```
with TEXT_IO;
use TEXT_IO;
procedure PIPELI is
    pragma TIME_SLICE(0.2);
    package INT_IO is new INTEGER_IO(INTEGER);
    use INT_IO;

    task MASTER is
         entry RECUP(X:in INTEGER);
```

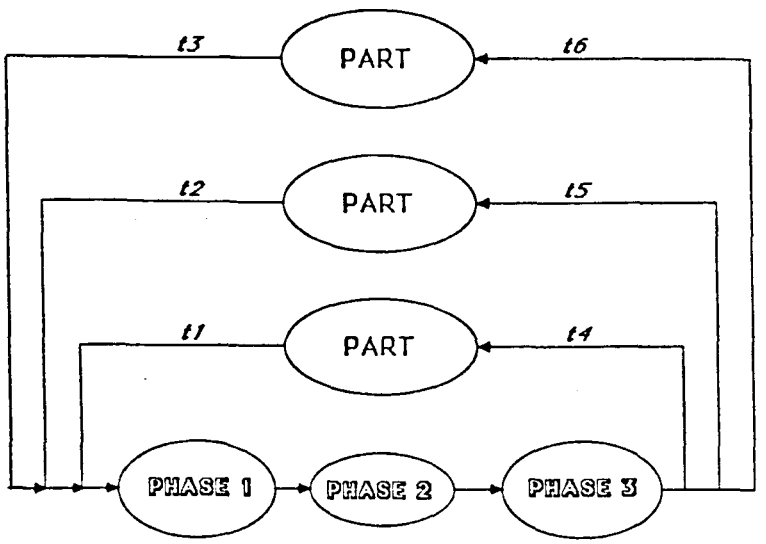


Figure 54: Partage du macro pipe – line de tâches

BU LILLE

```
end MASTER;

task body MASTER is
    TAILLE:constant INTEGER :=10;
    VECTEUR:array(1..64)of INTEGER;
    I,J:INTEGER:=1;
    -- tache controlant le macro pipe-line
    task T_CUBE is
        entry DEBUT(X:in INTEGER);
    end T_CUBE;

    task T_DIV is
        entry DEBUT(X:in INTEGER);
    end T_DIV;

    task T_PLUS is
        entry DEBUT(X:in INTEGER);
    end T_PLUS;

    task body T_CUBE is
        Y:INTEGER;
        -- calcul du cube de X
```

```
    begin
        loop
            accept DEBUT(X:in INTEGER) do
                Y:=X;
                PUT_LINE("AA");
            end;
            Y:=Y*Y*Y;
            -- rendez-vous avec la tache suivante
            T_DIV.DEBUT(Y);
        end loop;
    end T_CUBE;

    task body T_DIV is
        Y:INTEGER;
        -- division par 5 de X
    begin
        loop
            accept DEBUT(X:in INTEGER) do
                Y:=X;
                PUT_LINE("BBB");
            end;
            Y:=Y/5;
            -- rendez-vous avec la tache suivante
            T_PLUS.DEBUT(Y);
        end loop;
    end T_DIV;

    task body T_PLUS is
        Y:INTEGER;
        -- ajout de 1
    begin
        loop
            accept DEBUT(X:in INTEGER) do
                Y:=X;
                PUT_LINE("CCCC");
            end;
            Y:=Y+1;
            -- rendez-vous avec la tache controle
            MASTER.RECUP(Y);
        end loop;
    end T_PLUS;

begin
    -- initialisation du vecteur
    for K in 1..TAILLE loop
        VECTEUR(K):=K;
    end loop;
```

```
        -- la tache recupere d'abord les resultats, si il n'y en
        -- a pas, elle tente d'envoyer un jeu de donnees, si
        -- la tache T_CUBE n'est pas prete elle ne fait rien  etc..
        loop
        -- recuperation d'un resultat
            select
                accept RECUP(X:in INTEGER) do
                    VECTEUR(J):=X;
                    PUT_LINE("MASTER");
                end;
                J:=J+1;
              else
                -- pas de recuperation possible
                -- envoi d'une donnee
                if I<TAILLE
                  then
                    select
                        T_CUBE.DEBUT(VECTEUR(I));
                        I:=I+1;
                      else
                        null;
                    end if;
                end select;
            end select;
            exit when J>TAILLE;
        end loop;
        for K in 1..TAILLE loop
            INT_IO.PUT(VECTEUR(K));
            PUT_LINE("");
        end loop;
        abort T_CUBE,T_DIV,T_PLUS;
    end MASTER;

begin
    null;
end PIPELI;
```

4.2.3 Synchronisation indirecte

Le passage des données entre deux tâches est réalisé par une communication de type rendez-vous. En conséquence, la tâche émettrice doit attendre que la tâche réceptrice soit prête à recevoir le message. Pendant ce temps, elle aurait pu commencer à traiter un autre jeu de données. De plus les synchronisations étant de type direct, la tâche émettrice doit connaître le nom de la tâche réceptrice. Cela ne permet pas de multiplier le nombre de tâches exécutant le même travail, ce qui

pourtant, résoudrait un problème d'étranglement provoqué par le partage du travail entre plusieurs tâches, où l'une d'entre elles demanderait plus de temps CPU que les autres.

Ces deux problèmes de communication peuvent être résolus par l'utilisation de mécanismes de synchronisation indirecte de type sémaphores ou évènements. La publication présentée en appendice D propose une implémentation de ces types de synchronisation en Ada. On peut l'utiliser pour la réalisation du macro pipe-line. L'exemple précédent devient :

```
with TEXT_IO,SEMMES;
use TEXT_IO;
procedure PIPEL2 is
    pragma TIME_SLICE(0.2);
    -- definition d'un package de semaphore avec message
    -- pour des entiers : definition en appendice D
    package SEMAP is new SEMMES(INTEGER);
    use SEMAP;
    package INT_IO is new INTEGER_IO(INTEGER);
    use INT_IO;

    task MASTER is
        entry RECUP(X:in INTEGER);
    end MASTER;

    task body MASTER is
        TAILLE:constant INTEGER :=10;
        VECTEUR:array(1..64)of INTEGER;
        I,J:INTEGER:=1;
        SMASTER,SA,SB,SC:SEMAPHORE;

        task T_CUBE;

        task T_DIV;

        task T_PLUS;

        task body T_CUBE is
            Y,X:INTEGER;
            -- calcul du cube de X
        begin
            loop
                -- attente sur semaphore SA
                P(SA,X);
                Y:=X;
                PUT_LINE("AA");
                Y:=Y*Y*Y;
```

```
            -- liberation du semaphore SB : pas d'arret de T_CUBE
            V(SB,Y);
        end loop;
    end T_CUBE;

    task body T_DIV is
        Y,X:INTEGER;
        -- division de X par 5
    begin
        loop
            -- attente sur semaphore SB
            P(SB,X);
            Y:=X;
            PUT_LINE("BBB");
            Y:=Y/5;
            -- declenchement du semaphore SC
            V(SC,Y);
        end loop;
    end T_DIV;

    task body T_PLUS is
        Y,X:INTEGER;
        -- addition de 1 a X
    begin
        loop
            -- attente sur SC
            P(SC,X);
            Y:=X;
            PUT_LINE("CCCC");
            Y:=Y+1;
            -- liberation du semaphore SMASTER
            -- un resultat est produit
            -- la tache passe au jeu de donnees suivant
            V(SMASTER,Y);
        end loop;
    end T_PLUS;
    -- definition de la tache utilisant le pipe-line
begin
    -- declenchement direct pour toutes les donnees
    for K in 1..TAILLE loop
        VECTEUR(K):=K;
        V(SA,VECTEUR(K));
    end loop;
    -- attente pour chaque jeu de donnees
    for K in 1..TAILLE loop
        P(SMASTER,VECTEUR(K));
    end loop;
```

```
            for K in 1..TAILLE loop
                INT_IO.PUT(VECTEUR(K));
                PUT_LINE("");
            end loop;
            -- les taches T_CUBE, T_DIV, T_PLUS peuvent etre dupliquees
            -- sans modification des communications
            -- les memes semaphores seraient utilises
            abort T_CUBE,T_DIV,T_PLUS;
            PUT("fini");
        end MASTER;

begin
    null;
end PIPEL2;
```

4.3 Conclusion

Le multi – tâches disponible en Ada s'adapte parfaitement aux algorithmes de GEANT3. Cette programmation présente en plus l'intérêt d'être exécutable sur toute sorte d'architecture, qu'elle soit mono ou multi – processeurs. Les aspects communication et gestion des tâches sont en effet assurés par le compilateur Ada. Ainsi une seule version de GEANT3 est nécessaire se qui facilite grandement la mise au point et la maintenance.

Néanmoins, il ne faut se cacher la vérité : le travail de réécriture de GEANT3 en Ada est gigantesque. C'est sans doute l'inconvénient le plus important de Ada, c'est qu'il arrive bien longtemps après FORTRAN.

Partie 3

Conclusion

Nous avons présenté dans cette thèse quatre approches différentes permettant d'utiliser des architectures multi – processeurs, dans le but de réduire les temps d'exécution. Chaque solution proposée présente des aspects intéressants.

- *Multi – évènements:*

  Cette solution a comme avantage d'être simple à mettre en oeuvre et de ne pas demander une architecture très spéciale. Elle peut être appliquée sur un grand nombre de machines multi – processeurs. Seul le débit du réseau de communication impose une limite au nombre de processeurs actifs simultanément.

- *Pile partagée ou répartie:*

  Ces algorithmes ont l'avantage de proposer un gain aussi bien pour le traitement interactif que pour le traitement différé. Le nombre de processeurs utilisés peut être plus important que précédemment. Les architectures nécessaires sont beaucoup plus spécifiques qu'auparavant.

- *Vectorisation:*

  La vectorisation de GEANT3 demande beaucoup de modifications des algorithmes. Néanmoins, cette solution est très intéressante puisqu'elle permet d'utiliser les super – computers comme CRAY ou Cyber. La fiabilité et la puissance de ces machines sont en effet très attirantes pour les utilisateurs.

- *Multi – tâches:*

  Ada nous a charmé, malheureusement la modularité de programmation, la lisibilité des programmes, la structuration des données et le multi – tâches indépendant de l'architecture ne pourront provoquer la réécriture complète de GEANT3.

Beaucoup de travaux restent à faire, surtout sur la vectorisation. Enfin, la solution la meilleure ne serait – elle pas de connecter ensemble plusieurs émulateurs de CRAY par un FASTBUS et d'appliquer un algorithme multi – évènements où chaque évènement serait simulé de façon vectorielle ? (introduction d'une solution mixte multi – tâches vectorielle)

APPENDICE A

VECTEUR DE SÉMAPHORES

Lorsque plusieurs calculateurs sont en concurrence sur une même ressource critique, le partage de l'accès peut être géré par un système de synchronisation de type sémaphore. Un tel système n'empêche pas une utilisation abusive par un calculateur consistant à forcer l'accès par libération de la ressource réservée par un autre calculateur.

Exemple :

```
        cal 1                          cal 2
          .                              .
          .                          P ( mutex )
          .                              .
    V ( mutex )                          .
    P ( mutex )                          .
          .                              .
          .                          V ( mutex )
```

Une solution consiste à utiliser des " vecteurs sémaphores ". Au lieu d'utiliser 1 bit de sémaphore, on utilise N bits pour N calculateurs, soit :

$$Vsem = (b_1 , b_2 , b_3 , b_4) \quad \text{pour } N = 4$$

A un instant donné, seul 1 bit du vecteur peut avoir la valeur 1, seules les configurations suivantes sont donc possibles :

(0 , 0 , 0 , 0) ressource libre
(1 , 0 , 0 , 0) ressource réservée par le calculateur 1
(0 , 1 , 0 , 0) ressource réservée par le calculateur 2
(0 , 0 , 1 , 0) ressource réservée par le calculateur 3
(0 , 0 , 0 , 1) ressource réservée par le calculateur 4

(1 , 0 , 1 , 0) est impossible

Les opérations sur les vecteurs sémaphores sont les suivantes :

- *Opération MAS (reg.val , reg.sem):*

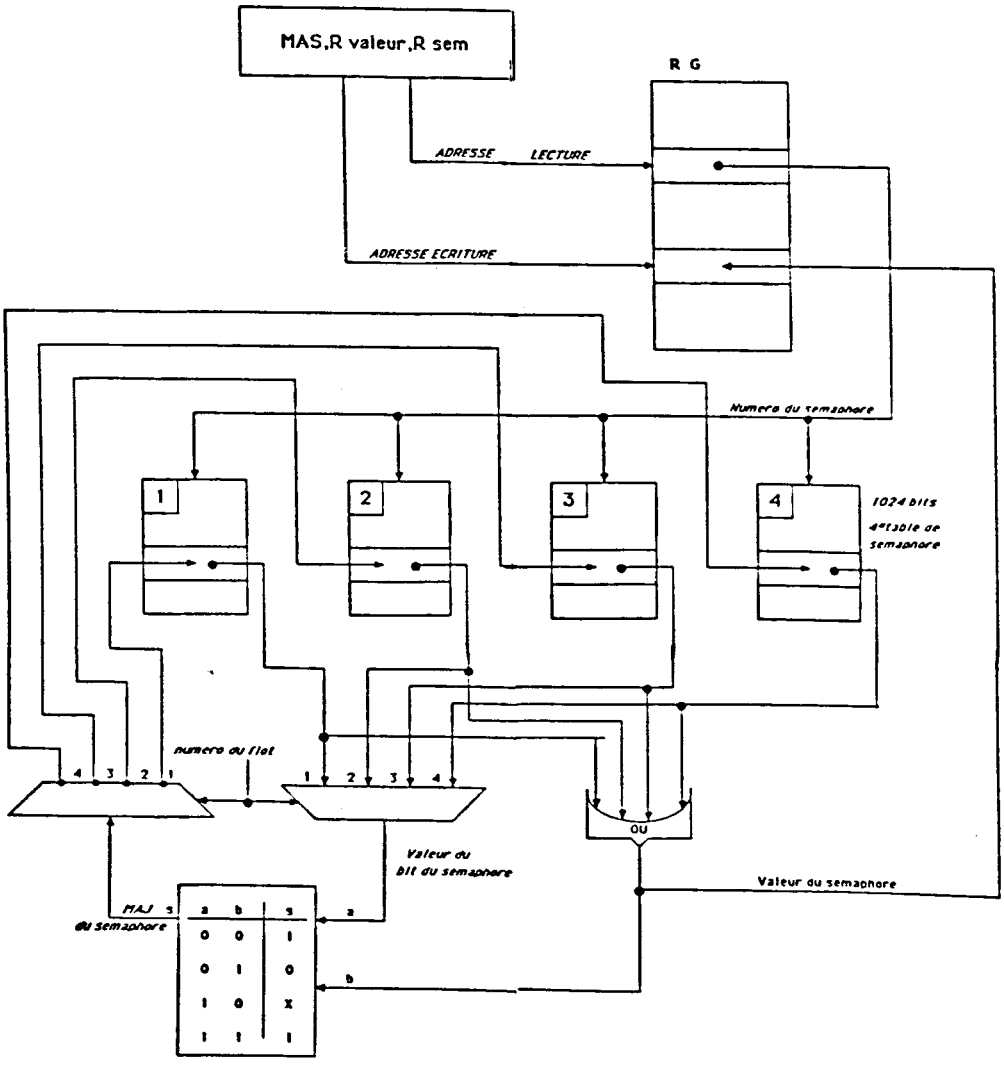


Figure 55: Réalisation matérielle pour MAS

elle charge dans le registre reg.val le résultat d'un OU effectué sur les N bits du vecteur sémaphore de numéro donné par reg.sem. Ensuite le bit associé au calculateur est mis à jour en respectant la table suivante

bit i	resul OU	bit i
0	0	1
0	1	0
1	0	X
1	1	1

le tout étant fait de façon indivisible.

- *Opération RES (reg.sem):*

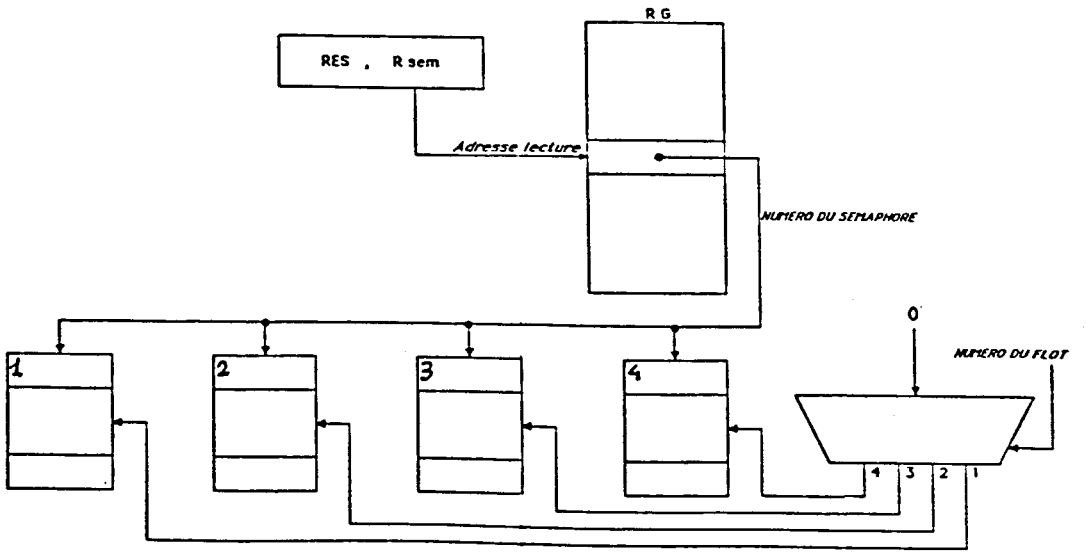


Figure 56: Réalisation matérielle pour RES

BU LILLE

met un zéro dans le bit du vecteur sémaphores associé au calculateur exécutant cette instruction.

- *Opération RESM (reg.sem):*

  met à zéro tous les bits du vecteur sémaphores, à utiliser seulement en mode privilégié pour débloquer une ressource.

Cette solution autorise l'exécution d'instructions sur sémaphores sans passer par un mode privilégié. La garantie que seul le calculateur possédant la ressource peut la libérer permet d'assurer une meilleure sécurité.

Une solution semblable existe, au lieu d'un vecteur de sémaphores, on utilise un entier représentant le numéro du processeur utilisant la ressource. Seul le processeur ayant réservé cette ressource peut alors la libérer.

La solution du vecteur sémaphores s'adapte parfaitement à la gestion de pile partagée proposée en 2.4. Une bascule peut être associée à chaque processeur, les valeurs de toutes ces bascules donnent l'état de la pile, et un processeur ne peut modifier que sa propre bascule.

APPENDICE B

UTILISATION DES ÉMULATEURS 3081E

Voici l'implémentation des procédures de communications définies dans le chapitre 2. Elles sont écrites à l'aide d'un gestionnaire de programme : PATCHY.

```
+PATCH,GEMUL
+KEEP,BUFOUT.
      PARAMETER (NBUFO=4999)
      COMMON/BUFOUT/LU,IPT,IPF,IPFC,IOBUF(NBUFO),IPREM
+KEEP,BUFIN.
      PARAMETER (NBUFI=5000)
      COMMON/BUFIN/LI,IPC,ILAS,IIBUF(NBUFI)
+KEEP,GCPARA
      COMMON/GCPARA/NPROC,NPRND,NPR1,LSCT(9)
+KEEP,MESSAG
      COMMON/MESSAG/IBRK,IMESS(88),NRANDO
+DECK,TIMEX,IF=-EMUL
      SUBROUTINE TIMEX(T)
      T=ITOD()/1000000.0
      END
+DECK,RND
      FUNCTION NGRND(NX0,I)
C     ***********************************************************************
C     *                                                                     *
C     *     FUNCTION NGNRD :PROCESS THE NTH   RANDOM NUMBER     *
C     *     called by GRUN                                                  *
C     *           J-L DEKEYSER              7.2.86                  *
C     *                                                                     *
C     ***********************************************************************
      DIMENSION IT(29)
      DATA IT/0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     +0,0,0/
      IF (IT(1).NE.0) GO TO 6
        IA=69069
        DO 5 I=1,29
          IF (IA.LT.0) IA=IA+2147483647+1
          IT(I)=IA
          IA=IA*IA
   5    CONTINUE
   6  N=I
      NB=1
      DO 100 J=1,32
        IF (N.EQ.0) GO TO 110
```

```
        NR=(N/2)*2
        IF (NR.NE.N) NB=NB*IT(J)
        N=N/2
  100 CONTINUE
  110 NB=NB*NX0
      IF (NB.LT.0)NB=NB+2147483647+1
      NGRND=NB
      END
      FUNCTION RNDM(DUMMY)
      RNDM=RN32(DUMMY)
      END
      SUBROUTINE RDMIN(INS)
      DUMMY=RN32IN(INS)
      END
      SUBROUTINE RDMOUT(IOUT)
      DUMMY=RN32OT(IOUT)
      END
+DECK,EFFGO,IF=-EMUL
      SUBROUTINE EFFGO
C.
C.    ************************************************************************************
C.    *                                                                                  *
C.    *       ROUTINE TO DEFINE  GEANT/FFREAD EMUL DATA CARD                             *
C.    *                                                                                  *
C.    *    ==>Called by : <USER>, UGINIT                                                 *
C.    *       AUTHOR    JL.DEKEYSER   ***                                                *
C.    *                                                                                  *
C.    ************************************************************************************
C.
+SEQ,GCPARA
C
C       DATA CARD FOR USING 3081E
C
      CALL FFKEY ('EMUL',NPROC ,3 ,'INTEGER')
C.
C.    ------------------------------------------------------------------
C.
C
 999  END
+DECK,EGINIT,IF=-EMUL
      SUBROUTINE EGINIT
C.
C.    ************************************************************************************
C.    *                                                                                  *
C.    *         3081E INITIALIZATION ROUTINE                                             *
C.    *                                                                                  *
C.    *                                                                                  *
```

```
C.    *    ==>Called by : <USER>, UGINIT                                    *
C.    *       AUTHOR    JL DEKEYSER   ***                                   *
C.    *                                                                     *
C.    **********************************************************************
C.
+SEQ,GCPARA
+SEQ,BUFIN
+SEQ,BUFOUT
C
C
C     INITIALIZATION OF VARIABLES FOR EMULATORS 3081E
C
      NPROC=1
      NPRND=10000
      NPR1=0
      IPF=0
      IPFC=0
      LI=-1
      LU=-1
      IPT=1
      IPC=1
      IPREM=0
C
      END
+DECK,GRUN,IF=-EMUL
      SUBROUTINE GRUN
C.
C.    **********************************************************************
C.    *                                                                     *
C.    *       STEERING ROUTINE TO PROCESS ALL THE EVENTS                    *
C.    *       ON EMULATORS 3081E                                            *
C.    *    ==>Called by : <USER>, main program                              *
C.    *       AUTHOR    J-L DEKEYSER  ***                                   *
C.    *                                                                     *
C.    **********************************************************************
C.
+SEQ,MESSAG
+SEQ,GCBANK
+SEQ,GCFLAG
+SEQ,GCNUM
+SEQ,GCUNIT
+SEQ,GCTIME
+SEQ,GCPARA
C.
      REAL*4 LDTIME
      INTEGER*4 HMBYTE/ Z 0018 0000 /, INIPAR / Z 8181 8181 /
C     SIZE FOR GOPAL
```

```
      INTEGER*4 ILPUT/Z 0017 FE00/
C     SIZE FOR GEXAM1
C     INTEGER*4 ILPUT/Z 0004 DD68/
C     SIZE FOR GEXAM3
C     INTEGER*4 ILPUT/Z 000B 46F0/
C.    ------------------------------------------------------------------
C.
C     INITIALIZATION OF SET-UP FOR 3081E
C
      CALL SYEOPT('CAMAC',0,1,20,5)
      NXTIME=300*NEVENT/NPROC
      CALL SYEOPT('TIMEOUT',NXTIME)
      CALL SYEOPT('DATAMEM',HMBYTE,INIPAT)
      CALL SYEOPT('LOADCHK',2)
C
C     RESERVATION OF EMULATORS
C
      DO 5 I=NPR1,NPROC-1+NPR1
        CALL SYEOPT('GIMEP',I,IRET,IPSIZE,IDSIZE)
   5  CONTINUE
C
C     DOWNLOAD OF PROGRAM ON ALL THE EMULATORS
C
      CALL SYEDWN(-2,10,'CLEA',IRET)
      PRINT *,IRET
      CALL TIMEX(TIMINT)
C
C     PROCESS INIT RANDOM NUMBER AND NUMBER OF EVENTS
C     EACH EMULATOR
C
      CALL RDMOUT(NRANDO)
      NR=NRANDO
      NVT=NEVENT/NPROC
      NVT1=NEVENT-(NVT*NPROC)
C
C     LOAD ALL COMMONS FROM HOST TO 3081E
C
      CALL SYEPUT(-2,'MESSAG ',IBRK    ,'    ',IRET,0,ILPUT)
      DO 10 I=NPR1,NPROC-1+NPR1
        NEVENT=NVT
        IF (I.EQ.NPROC-1+NPR1) NEVENT=NEVENT+NVT1
        IF (I.NE.NPR1)NRANDO=NGRND(NR,NVT*NPRND*(I-NPR1))
        WRITE(LOUT,3000)NEVENT,NRANDO,I
        IMESS(1)=NRANDO
        IMESS(2)=NEVENT
C
C     LOAD RANDOM NUMBER AND NUMBER OF EVENT AND START
```

```
C
        CALL SYEPUT(I,'MESSAG  ',IBRK    ,'GO  ',IRET,0,12   )
        PRINT *,IRET
   10 CONTINUE
      NEVENT=NVT*NPROC+NVT1
      IEVENT=NEVENT
C
C     WAIT FOR ONE EMULATOR IS STOPPED
C
   11 CALL SYEGET(-1,'MESSAG  ',IBRK    ,'    ',IRET,0,24,IDGET)
      PRINT *,IRET,IMESS(1),IMESS(2)
      CALL SYEOPT('PRTIME',IDGET,ILOAD,IEXEC,IFETCH)
      IF (IMESS(1).EQ.9999) GO TO 99
      IF (NPROC.EQ.1) IRET=1
      LDTIME=ILOAD/1 000 000.
      EXTIME=IEXEC/1 000 000.
      FETIME=IFETCH/1 000 000.
      WRITE(6,1234)LDTIME,EXTIME,FETIME,IDGET
C
C     CASE OF STOP
C
      IF (IMESS(1).EQ.8) THEN
          CALL GHBGSC(IDGET)
          ELSE
          IF (IMESS(1).EQ.18) THEN
              CALL GHNDSC(IDGET)
              ELSE
              IF (IMESS(1).EQ.9) THEN
                  CALL GHEMPT(IDGET)
                  ELSE
                  IF (IMESS(1).EQ.10) THEN
                      CALL GHFILL(IDGET)
                      ELSE
                        IF (IMESS(1).EQ.918) THEN
                        CALL GHEMPT(IDGET)
                        CALL GHNDSC(IDGET)
                        ELSE
                      GO TO 30
                      END IF
                  END IF
              END IF
          END IF
      END IF
      GO TO 11
C
C     GO AHEAD IF 1 EMULATEUR IS ACTIF YET
C
```

```
 30   IF (IRET.EQ.0) GO TO 11
 1234 FORMAT(' *** TIMING SECONDE *** LOAD EVENT ',F6.2,' EXECUTION '
     *      ,F10.2,' FETCH RESULT ',F6.2,' PROCESSEUR ',I3)
C
 3000 FORMAT('*** NUMBER OF EVENTS ',I8,' RANDOM NUMBER ',I12,
     +' EMULATEUR NUMBER ',I3,' ***',//)
  99  END
+DECK,GRUNE,IF=EMUL
      SUBROUTINE GRUN
C.
C.    ************************************************************************************
C.    *                                                                                  *
C.    *       STEERING ROUTINE TO PROCESS ALL THE EVENTS                                 *
C.    *       ON ONE EMULATORS                                                           *
C.    *    ==>CALLED BY : <USER>, MAIN PROGRAM ON 3081E                                  *
C.    *       Author    R.Brun  **********                                               *
C.    *                                                                                  *
C.    ************************************************************************************
+SEQ,MESSAG
+SEQ,GCBANK
+SEQ,GCFLAG
+SEQ,GCNUM
+SEQ,GCUNIT
+SEQ,GCTIME
C.
C.    ------------------------------------------------------------------
C.
C             Keep starting time
C
C
C             Initialises event partition
  10  CALL GTRIGI
C             Process one event (trigger)
      CALL GTRIG
C             Clear event partition
      CALL GTRIGC
      IF(IEORUN.NE.0)GO TO 99
      IEVENT = IEVENT + 1
      IF(IEVENT.GT.NEVENT)GO TO 90
      GO TO 10
C
  90  IEVENT=IEVENT-1
  99  END
+DECK,GUHHIS,IF=-EMUL
      SUBROUTINE GUHHIS
C     ********************************************************************
C     *                                                                  *
```

```
C     *      RECONSTRUCTION OF HISTO ON HOST                          *
C     *                                                               *
C     *      CALLED BY UGLAST                                         *
C     *            J-L DEKEYSER              7.2.86                   *
C     *                                                               *
C     ****************************************************************
+SEQ,GCPARA
      COMMON/HFLAG/IDH1(16),NEWHIS,IDH2(14)
      COMMON//IQ(10)
C     CALL HISTGO(0)
      DO 10 I=NPR1,NPROC-1+NPR1
        CALL SYEGET(I,'BLANKCOM',IQ,'RDM ',IRET,0,NEWHIS*4)
        PRINT *,IRET
        IF (IRET.GE.2) GO TO 10
        IF (I.NE.NPR1) THEN
          CALL HFETCH(0,35)
          REWIND 35
        END IF
        CALL HSTORE(0,35)
  10  CONTINUE
      END
+DECK,FZEMU,IF=EMUL
       SUBROUTINE ZEBEMU(LV)
C     ****************************************************************
C     *                                                               *
C     *    PROCESS ABSOLUTE ADDRESS OF ZEBRA ON 3081E                 *
C     *                                                               *
C     ****************************************************************
       DIMENSION LV(9)
       COMMON/MZCA/NN(33),LQATAB,LQASTO,NNN(54)
       COMMON/MZCB/JQSTOR,KQT,NNB(8),LQSTOR,NNNB(15)
       COMMON/MZCC/IQTABV(16),NNNC(203)
       COMMON/ZEBQ/IDFENC(4),LQ(24)
       LQATAB=LOCF(IQTABV(1))-1
       LQASTO=LOCF(LQ(1))-1
       LQSTOR=LOCF(LV(1))-1
       IQTABV(KQT+1)=LQSTOR
       END
      SUBROUTINE FZOX1 (V1,N1)

      PARAMETER      (IQDROP=25, IQMARK=26, IQCRIT=27, IQSYSX=28)
      COMMON /QUEST/ IQUEST(100)
      COMMON /ZEBQ/  IQFENC(4), LQ(24)
                                              DIMENSION IQ(19), Q(19)
                                          EQUIVALENCE (LQ(9),IQ(1),Q(1))
      COMMON /MZCA/  NQSTOR, NQOFFT(16), NQOFFS(16)
     +,              LQATAB,LQASTO,LQBTIS
```

```
     +,              MQKEYS(3), NQINIT, NQTSYS, NQM99
     +,              NQFATA,NQCASE, NQTRAC,MQTRAC(44)
                                    EQUIVALENCE (KQSP,NQOFFS(1))
                                    DIMENSION    IQCUR(16)
                                    EQUIVALENCE (IQCUR(1),LQSTOR)
      COMMON /MZCB/  JQSTOR,KQT,KQS
     +,              JQDIVI,JQDIVR,JQKIND,JQMODE,JQDIVN,JQSHAR,NQRESV
     +,              LQSTOR,NQFEND,NQSTRU,NQREF,NQLINK,NQMINR,LQ2END
     +,              JQDVLL,JQDVSY,NQLOGL,NQSNAM(6)
                                    DIMENSION    IQTABV(16)
                                    EQUIVALENCE (IQTABV(1),LQPSTO)
      COMMON /MZCC/  LQPSTO,NQPFEN,NQPSTR,NQPREF,NQPLK,NQPMIN,LQP2E
     +,              JQPDVL,JQPDVS,NQPLOG,NQPNAM(6)
     +,              LQSYSS(10), LQSYSR(10), IQTDUM(22)
     +,              LQSTA(21), LQEND(20), NQDMAX(20),IQMODE(20)
     +,              IQKIND(20),IQRTO(20), IQRNO(20), NQDWIP(20)
     +,              NQDGAU(20),NQDGAF(20),NQDPSH(20),NQDSIZ(20)
     +,              IQDN1(20), IQDN2(20)
                   EQUIVALENCE (LQFS,LQSYSS(4)), (LQFF,LQSYSR(4))
     +,                        (LQFI,LQSYSR(5)), (LQFX,LQSYSR(6))
      COMMON /FZCX/  LUNX,LUNNX,IXDIVX,LTEMPX,IEVFLX
     +,              MEDIUX,IEXCHX,IACTVX,MAXREX,LOGLVX
     +,              IOPTXD,IOPTXI,IOPTXS,IOPTXZ
     +,              IOPTXL,IOPTXN,IOPTXP,IOPTXK
     +,              IDX(2),IPILX(4),NWTXX,NWSEGX,NWTABX,NWBKX,LENTRX
     +,              NWUHCX,IOCHX(16),NWUHX,NWIOX
C--------------    END CDE                                  --------------
      DIMENSION V1(N1)

      IDX(1) = N1
      CALL GEPBUF(LUNX,IDX,2)
C     WRITE (LUNX) IDX,V1
      CALL GEPBUF(LUNX,V1,N1)
      CALL GENDRC
      IQ(KQSP+LQFX+19) = IQ(KQSP+LQFX+19) + IDX(1) + 3
      IQ(KQSP+LQFX+20) = IQ(KQSP+LQFX+20) + 1
      RETURN
      END
      SUBROUTINE FZOX2 (V1,N1,V2,N2)

      PARAMETER      (IQDROP=25, IQMARK=26, IQCRIT=27, IQSYSX=28)
      COMMON /QUEST/ IQUEST(100)
      COMMON /ZEBQ/  IQFENC(4), LQ(24)
                                          DIMENSION IQ(19), Q(19)
                                        EQUIVALENCE (LQ(9),IQ(1),Q(1))
      COMMON /MZCA/  NQSTOR, NQOFFT(16), NQOFFS(16)
     +,              LQATAB,LQASTO,LQBTIS
```

```
     +,              MQKEYS(3), NQINIT, NQTSYS, NQM99
     +,              NQFATA,NQCASE, NQTRAC,MQTRAC(44)
                                    EQUIVALENCE (KQSP,NQOFFS(1))
                                    DIMENSION    IQCUR(16)
                                    EQUIVALENCE (IQCUR(1),LQSTOR)
      COMMON /MZCB/  JQSTOR,KQT,KQS
     +,              JQDIVI,JQDIVR,JQKIND,JQMODE,JQDIVN,JQSHAR,NQRESV
     +,              LQSTOR,NQFEND,NQSTRU,NQREF,NQLINK,NQMINR,LQ2END
     +,              JQDVLL,JQDVSY,NQLOGL,NQSNAM(6)
                                    DIMENSION    IQTABV(16)
                                    EQUIVALENCE (IQTABV(1),LQPSTO)
      COMMON /MZCC/  LQPSTO,NQPFEN,NQPSTR,NQPREF,NQPLK,NQPMIN,LQP2E
     +,              JQPDVL,JQPDVS,NQPLOG,NQPNAM(6)
     +,              LQSYSS(10), LQSYSR(10), IQTDUM(22)
     +,              LQSTA(21), LQEND(20), NQDMAX(20),IQMODE(20)
     +,              IQKIND(20),IQRTO(20), IQRNO(20), NQDWIP(20)
     +,              NQDGAU(20),NQDGAF(20),NQDPSH(20),NQDSIZ(20)
     +,              IQDN1(20), IQDN2(20)
                   EQUIVALENCE (LQFS,LQSYSS(4)), (LQFF,LQSYSR(4))
     +,                        (LQFI,LQSYSR(5)), (LQFX,LQSYSR(6))
      COMMON /FZCX/  LUNX,LUNNX,IXDIVX,LTEMPX,IEVFLX
     +,              MEDIUX,IEXCHX,IACTVX,MAXREX,LOGLVX
     +,              IOPTXD,IOPTXI,IOPTXS,IOPTXZ
     +,              IOPTXL,IOPTXN,IOPTXP,IOPTXK
     +,              IDX(2),IPILX(4),NWTXX,NWSEGX,NWTABX,NWBKX,LENTRX
     +,              NWUHCX,IOCHX(16),NWUHX,NWIOX
C--------------     END CDE                                  --------------
      DIMENSION V1(N1),V2(N2)

      IDX(1) = N1 + N2
      CALL GEPBUF(LUNX,IDX,2)
C     WRITE (LUNX) IDX,V1,V2
      CALL GEPBUF(LUNX,V1,N1)
      CALL GEPBUF(LUNX,V2,N2)
      CALL GENDRC
      IQ(KQSP+LQFX+19) = IQ(KQSP+LQFX+19) + IDX(1) + 3
      IQ(KQSP+LQFX+20) = IQ(KQSP+LQFX+20) + 1
      RETURN
      END
      SUBROUTINE FZOX3 (V1,N1,V2,N2,V3,N3)

      PARAMETER      (IQDROP=25, IQMARK=26, IQCRIT=27, IQSYSX=28)
      COMMON /QUEST/ IQUEST(100)
      COMMON /ZEBQ/  IQFENC(4), LQ(24)
                                          DIMENSION IQ(19), Q(19)
                                       EQUIVALENCE (LQ(9),IQ(1),Q(1))
      COMMON /MZCA/  NQSTOR, NQOFFT(16), NQOFFS(16)
```

```
     +,              LQATAB,LQASTO,LQBTIS
     +,              MQKEYS(3), NQINIT, NQTSYS, NQM99
     +,              NQFATA,NQCASE, NQTRAC,MQTRAC(44)
                                    EQUIVALENCE (KQSP,NQOFFS(1))
                                    DIMENSION    IQCUR(16)
                                    EQUIVALENCE (IQCUR(1),LQSTOR)
      COMMON /MZCB/  JQSTOR,KQT,KQS
     +,              JQDIVI,JQDIVR,JQKIND,JQMODE,JQDIVN,JQSHAR,NQRESV
     +,              LQSTOR,NQFEND,NQSTRU,NQREF,NQLINK,NQMINR,LQ2END
     +,              JQDVLL,JQDVSY,NQLOGL,NQSNAM(6)
                                    DIMENSION    IQTABV(16)
                                    EQUIVALENCE (IQTABV(1),LQPSTO)
      COMMON /MZCC/  LQPSTO,NQPFEN,NQPSTR,NQPREF,NQPLK,NQPMIN,LQP2E
     +,              JQPDVL,JQPDVS,NQPLOG,NQPNAM(6)
     +,              LQSYSS(10), LQSYSR(10), IQTDUM(22)
     +,              LQSTA(21), LQEND(20), NQDMAX(20),IQMODE(20)
     +,              IQKIND(20),IQRTO(20), IQRNO(20), NQDWIP(20)
     +,              NQDGAU(20),NQDGAF(20),NQDPSH(20),NQDSIZ(20)
     +,              IQDN1(20), IQDN2(20)
                   EQUIVALENCE (LQFS,LQSYSS(4)), (LQFF,LQSYSR(4))
     +,                        (LQFI,LQSYSR(5)), (LQFX,LQSYSR(6))
      COMMON /FZCX/  LUNX,LUNNX,IXDIVX,LTEMPX,IEVFLX
     +,              MEDIUX,IEXCHX,IACTVX,MAXREX,LOGLVX
     +,              IOPTXD,IOPTXI,IOPTXS,IOPTXZ
     +,              IOPTXL,IOPTXN,IOPTXP,IOPTXK
     +,              IDX(2),IPILX(4),NWTXX,NWSEGX,NWTABX,NWBKX,LENTRX
     +,              NWUHCX,IOCHX(16),NWUHX,NWIOX
C--------------    END CDE                                  --------------
      DIMENSION V1(N1),V2(N2),V3(N3)

      IDX(1) = N1 + N2 + N3
      CALL GEPBUF(LUNX,IDX,2)
C     WRITE (LUNX) IDX,V1,V2,V3
      CALL GEPBUF(LUNX,V1,N1)
      CALL GEPBUF(LUNX,V2,N2)
      CALL GEPBUF(LUNX,V3,N3)
      CALL GENDRC
      IQ(KQSP+LQFX+19) = IQ(KQSP+LQFX+19) + IDX(1) + 3
      IQ(KQSP+LQFX+20) = IQ(KQSP+LQFX+20) + 1
      RETURN
      END
      SUBROUTINE FZOX4 (V1,N1,V2,N2,V3,N3,V4,N4)

      PARAMETER      (IQDROP=25, IQMARK=26, IQCRIT=27, IQSYSX=28)
      COMMON /QUEST/ IQUEST(100)
      COMMON /ZEBQ/  IQFENC(4), LQ(24)
                                              DIMENSION IQ(19), Q(19)
```

```
                                        EQUIVALENCE (LQ(9),IQ(1),Q(1))
      COMMON /MZCA/  NQSTOR, NQOFFT(16), NQOFFS(16)
     +,              LQATAB,LQASTO,LQBTIS
     +,              MQKEYS(3), NQINIT, NQTSYS, NQM99
     +,              NQFATA,NQCASE, NQTRAC,MQTRAC(44)
                                        EQUIVALENCE (KQSP,NQOFFS(1))
                                        DIMENSION    IQCUR(16)
                                        EQUIVALENCE (IQCUR(1),LQSTOR)
      COMMON /MZCB/  JQSTOR,KQT,KQS
     +,              JQDIVI,JQDIVR,JQKIND,JQMODE,JQDIVN,JQSHAR,NQRESV
     +,              LQSTOR,NQFEND,NQSTRU,NQREF,NQLINK,NQMINR,LQ2END
     +,              JQDVLL,JQDVSY,NQLOGL,NQSNAM(6)
                                        DIMENSION    IQTABV(16)
                                        EQUIVALENCE (IQTABV(1),LQPSTO)
      COMMON /MZCC/  LQPSTO,NQPFEN,NQPSTR,NQPREF,NQPLK,NQPMIN,LQP2E
     +,              JQPDVL,JQPDVS,NQPLOG,NQPNAM(6)
     +,              LQSYSS(10), LQSYSR(10), IQTDUM(22)
     +,              LQSTA(21), LQEND(20), NQDMAX(20),IQMODE(20)
     +,              IQKIND(20),IQRTO(20), IQRNO(20), NQDWIP(20)
     +,              NQDGAU(20),NQDGAF(20),NQDPSH(20),NQDSIZ(20)
     +,              IQDN1(20), IQDN2(20)
                   EQUIVALENCE (LQFS,LQSYSS(4)), (LQFF,LQSYSR(4))
     +,                        (LQFI,LQSYSR(5)), (LQFX,LQSYSR(6))
      COMMON /FZCX/  LUNX,LUNNX,IXDIVX,LTEMPX,IEVFLX
     +,              MEDIUX,IEXCHX,IACTVX,MAXREX,LOGLVX
     +,              IOPTXD,IOPTXI,IOPTXS,IOPTXZ
     +,              IOPTXL,IOPTXN,IOPTXP,IOPTXK
     +,              IDX(2),IPILX(4),NWTXX,NWSEGX,NWTABX,NWBKX,LENTRX
     +,              NWUHCX,IOCHX(16),NWUHX,NWIOX
C--------------     END CDE                                  --------------
      DIMENSION V1(N1),V2(N2),V3(N3),V4(N4)

      IDX(1) = N1 + N2 + N3 + N4
      CALL GEPBUF(LUNX,IDX,2)
C     WRITE (LUNX) IDX,V1,V2,V3,V4
      CALL GEPBUF(LUNX,V1,N1)
      CALL GEPBUF(LUNX,V2,N2)
      CALL GEPBUF(LUNX,V3,N3)
      CALL GEPBUF(LUNX,V4,N4)
      CALL GENDRC
      IQ(KQSP+LQFX+19) = IQ(KQSP+LQFX+19) + IDX(1) + 3
      IQ(KQSP+LQFX+20) = IQ(KQSP+LQFX+20) + 1
      RETURN
      END
      SUBROUTINE FZOX5 (V1,N1,V2,N2,V3,N3,V4,N4,V5,N5)

      PARAMETER      (IQDROP=25, IQMARK=26, IQCRIT=27, IQSYSX=28)
```

```
      COMMON /QUEST/ IQUEST(100)
      COMMON /ZEBQ/  IQFENC(4), LQ(24)
                                              DIMENSION IQ(19), Q(19)
                                          EQUIVALENCE (LQ(9),IQ(1),Q(1))
      COMMON /MZCA/  NQSTOR, NQOFFT(16), NQOFFS(16)
     +,              LQATAB,LQASTO,LQBTIS
     +,              MQKEYS(3), NQINIT, NQTSYS, NQM99
     +,              NQFATA,NQCASE, NQTRAC,MQTRAC(44)
                                          EQUIVALENCE (KQSP,NQOFFS(1))
                                          DIMENSION    IQCUR(16)
                                          EQUIVALENCE (IQCUR(1),LQSTOR)
      COMMON /MZCB/  JQSTOR,KQT,KQS
     +,              JQDIVI,JQDIVR,JQKIND,JQMODE,JQDIVN,JQSHAR,NQRESV
     +,              LQSTOR,NQFEND,NQSTRU,NQREF,NQLINK,NQMINR,LQ2END
     +,              JQDVLL,JQDVSY,NQLOGL,NQSNAM(6)
                                          DIMENSION    IQTABV(16)
                                          EQUIVALENCE (IQTABV(1),LQPSTO)
      COMMON /MZCC/  LQPSTO,NQPFEN,NQPSTR,NQPREF,NQPLK,NQPMIN,LQP2E
     +,              JQPDVL,JQPDVS,NQPLOG,NQPNAM(6)
     +,              LQSYSS(10), LQSYSR(10), IQTDUM(22)
     +,              LQSTA(21), LQEND(20), NQDMAX(20),IQMODE(20)
     +,              IQKIND(20),IQRTO(20), IQRNO(20), NQDWIP(20)
     +,              NQDGAU(20),NQDGAF(20),NQDPSH(20),NQDSIZ(20)
     +,              IQDN1(20), IQDN2(20)
                   EQUIVALENCE (LQFS,LQSYSS(4)), (LQFF,LQSYSR(4))
     +,                        (LQFI,LQSYSR(5)), (LQFX,LQSYSR(6))
      COMMON /FZCX/  LUNX,LUNNX,IXDIVX,LTEMPX,IEVFLX
     +,              MEDIUX,IEXCHX,IACTVX,MAXREX,LOGLVX
     +,              IOPTXD,IOPTXI,IOPTXS,IOPTXZ
     +,              IOPTXL,IOPTXN,IOPTXP,IOPTXK
     +,              IDX(2),IPILX(4),NWTXX,NWSEGX,NWTABX,NWBKX,LENTRX
     +,              NWUHCX,IOCHX(16),NWUHX,NWIOX
C-------------     END CDE                                 -------------
      DIMENSION V1(N1),V2(N2),V3(N3),V4(N4),V5(N5)

      IDX(1) = N1 + N2 + N3 + N4 + N5
      CALL GEPBUF(LUNX,IDX,2)
C     WRITE (LUNX) IDX,V1,V2,V3,V4,V5
      CALL GEPBUF(LUNX,V1,N1)
      CALL GEPBUF(LUNX,V2,N2)
      CALL GEPBUF(LUNX,V3,N3)
      CALL GEPBUF(LUNX,V4,N4)
      CALL GEPBUF(LUNX,V5,N5)
      CALL GENDRC
      IQ(KQSP+LQFX+19) = IQ(KQSP+LQFX+19) + IDX(1) + 3
      IQ(KQSP+LQFX+20) = IQ(KQSP+LQFX+20) + 1
      RETURN
```

```
      END
      SUBROUTINE FZOX6 (V1,N1,V2,N2,V3,N3,V4,N4,V5,N5,V6,N6)

      PARAMETER      (IQDROP=25, IQMARK=26, IQCRIT=27, IQSYSX=28)
      COMMON /QUEST/ IQUEST(100)
      COMMON /ZEBQ/  IQFENC(4), LQ(24)
                                              DIMENSION IQ(19), Q(19)
                                    EQUIVALENCE (LQ(9),IQ(1),Q(1))
      COMMON /MZCA/  NQSTOR, NQOFFT(16), NQOFFS(16)
     +,              LQATAB,LQASTO,LQBTIS
     +,              MQKEYS(3), NQINIT, NQTSYS, NQM99
     +,              NQFATA,NQCASE, NQTRAC,MQTRAC(44)
                                    EQUIVALENCE (KQSP,NQOFFS(1))
                                    DIMENSION    IQCUR(16)
                                    EQUIVALENCE (IQCUR(1),LQSTOR)
      COMMON /MZCB/  JQSTOR,KQT,KQS
     +,              JQDIVI,JQDIVR,JQKIND,JQMODE,JQDIVN,JQSHAR,NQRESV
     +,              LQSTOR,NQFEND,NQSTRU,NQREF,NQLINK,NQMINR,LQ2END
     +,              JQDVLL,JQDVSY,NQLOGL,NQSNAM(6)
                                    DIMENSION    IQTABV(16)
                                    EQUIVALENCE (IQTABV(1),LQPSTO)
      COMMON /MZCC/  LQPSTO,NQPFEN,NQPSTR,NQPREF,NQPLK,NQPMIN,LQP2E
     +,              JQPDVL,JQPDVS,NQPLOG,NQPNAM(6)
     +,              LQSYSS(10), LQSYSR(10), IQTDUM(22)
     +,              LQSTA(21), LQEND(20), NQDMAX(20),IQMODE(20)
     +,              IQKIND(20),IQRTO(20), IQRNO(20), NQDWIP(20)
     +,              NQDGAU(20),NQDGAF(20),NQDPSH(20),NQDSIZ(20)
     +,              IQDN1(20), IQDN2(20)
                  EQUIVALENCE (LQFS,LQSYSS(4)), (LQFF,LQSYSR(4))
     +,                       (LQFI,LQSYSR(5)), (LQFX,LQSYSR(6))
      COMMON /FZCX/  LUNX,LUNNX,IXDIVX,LTEMPX,IEVFLX
     +,              MEDIUX,IEXCHX,IACTVX,MAXREX,LOGLVX
     +,              IOPTXD,IOPTXI,IOPTXS,IOPTXZ
     +,              IOPTXL,IOPTXN,IOPTXP,IOPTXK
     +,              IDX(2),IPILX(4),NWTXX,NWSEGX,NWTABX,NWBKX,LENTRX
     +,              NWUHCX,IOCHX(16),NWUHX,NWIOX
C-------------     END CDE                                  -------------
      DIMENSION V1(N1),V2(N2),V3(N3),V4(N4),V5(N5),V6(N6)

      IDX(1) = N1 + N2 + N3 + N4 + N5 + N6
      CALL GEPBUF(LUNX,IDX,2)
C     WRITE (LUNX) IDX,V1,V2,V3,V4,V5,V6
      CALL GEPBUF(LUNX,V1,N1)
      CALL GEPBUF(LUNX,V2,N2)
      CALL GEPBUF(LUNX,V3,N3)
      CALL GEPBUF(LUNX,V4,N4)
      CALL GEPBUF(LUNX,V5,N5)
```

```
      CALL GEPBUF(LUNX,V6,N6)
      CALL GENDRC
      IQ(KQSP+LQFX+19) = IQ(KQSP+LQFX+19) + IDX(1) + 3
      IQ(KQSP+LQFX+20) = IQ(KQSP+LQFX+20) + 1
      RETURN
      END
      SUBROUTINE GEPBUF(LUN,V,N)
C     ************************************************************************
C     *                                                                      *
C     *    PUT IN BUFFER FOR OUTPUT                                          *
C     *                                                                      *
C     *    CALLED BY   FZOX1,FZOX2,....FZOX6                                 *
C     *           J-L DEKEYSER              7.2.86                           *
C     *                                                                      *
C     ************************************************************************
+SEQ,BUFOUT.
+SEQ,MESSAG
+SEQ,GCPARA
      DIMENSION V(N)
      IF (N.LE.0) GO TO 99
C
C     EMPTIES  BUFFER IF IT IS NOT THE SAME FILE
C
      IF (LU.NE.LUN.AND.LU.NE.-1) THEN
C
C     BEGIN OF CRITICAL SECTION IF MORE THAN ONE EMULATOR
C
            IMESS(1)=8
            IMESS(2)=LU
            CALL BREAK
            IMESS(1)=0
            CALL GEMPSC
      END IF
      IF (N.GT.NBUFO-IPT) THEN
        IF (IPF.EQ.0) THEN
C
C     BUFFER TOO SMALL
C
          IMESS(1)=9999
          IMESS(2)=1
          CALL BREAK
        ELSE
          IF (IPREM.EQ.0.AND.NPROC.GT.1) THEN
C
C     BEGIN OF CRITICAL SECTION IF MORE THAN ONE EMULATOR
C
            IMESS(1)=8
```

```
               IMESS(2)=LUN
               CALL BREAK
               IMESS(1)=0
               IPREM=1
             END IF
             CALL GEMPTI
           END IF
         END IF
         LU=LUN
         CALL UCOPY(V,IOBUF(IPT),N)
         IPT=IPT+N
   99 END
      SUBROUTINE GENDRC
C    ************************************************************************
C    *                                                                      *
C    *    PUT A END OF RECORD IN THE BUFFER OF OUTPUT                       *
C    *                                                                      *
C    *     CALLED BY  FZOX1,FZOX2,....FZOX6                                 *
C    *           J-L DEKEYSER              7.2.86                           *
C    *                                                                      *
C    ************************************************************************
+SEQ,BUFOUT
      IF (IPFC.NE.0) THEN
          IOBUF(IPFC)=IPT
        ELSE
          IPF=IPT
      END IF
      IOBUF(IPT)=-1
      IPFC=IPT
      IPT=IPT+1
   99 END
      SUBROUTINE GEMPSC
C    ************************************************************************
C    *                                                                      *
C    *    INTERRUPTION FOR EMPTIING BUFFER                                  *
C    *                 AND END OF CRITICAL SECTION                          *
C    *     CALLED BY  GSAVE                                                 *
C    *           J-L DEKEYSER              4.3.86                           *
C    *                                                                      *
C    ************************************************************************
+SEQ,BUFOUT
+SEQ,MESSAG
      IMESS(1)=918
      IMESS(2)=IPFC*4+16
      CALL BREAK
      IMESS(1)=0
      IPREM=0
```

```
      IPF=0
      IPT=1
      IPFC=0
  999 END
      SUBROUTINE GEMPTI
C     ************************************************************************
C     *                                                                      *
C     *    INTERRUPTION FOR EMPTIING BUFFER                                  *
C     *                                                                      *
C     *     CALLED BY  GEPBUF ET GENDRC                                      *
C     *           J-L DEKEYSER              7.2.86                           *
C     *                                                                      *
C     ************************************************************************
+SEQ,BUFOUT
+SEQ,MESSAG
      IF (IPF.EQ.0) GO TO 999
      IMESS(1)=9
      IMESS(2)=IPFC*4+16
      CALL BREAK
      IMESS(1)=0
      IF (IPT.EQ.IPFC+1) GO TO 200
C
C     PUT THE CURRENT RECORD AT HEAD
C
      CALL UCOPY(IOBUF(IPFC+1),IOBUF(1),IPT-IPFC-1)
      IPF=0
      IPT=IPT-IPFC
      IPFC=0
      GO TO 999
  200 IPT=1
      IPFC=0
      IPF=0
  999 END
      SUBROUTINE XINBS(LUN,XAV,NA,XV,NX)
C     ************************************************************************
C     *                                                                      *
C     *    INTERRUPTION FOR FILLING BUFFER IN INPUT BY XINBS                 *
C     *                                                                      *
C     *     CALLED BY  FZINA,....                                            *
C     *           J-L DEKEYSER              7.2.86                           *
C     *                                                                      *
C     ************************************************************************
+SEQ,BUFIN.
+SEQ,MESSAG
      DIMENSION XV(NX),XAV(NA)
      IMESS(1)=10
      IMESS(2)=LUN
```

```
      IMESS(3)=NA
      IMESS(4)=NX
      CALL BREAK
      IMESS(1)=0
      IF (ILAS.LE.0) GO TO 100
      CALL UCOPY(IIBUF,XAV,NA)
      CALL UCOPY(IIBUF(1+NA),XV,ILAS)
      N=MIN0(ILAS,NX)
      NX=MAX0(N,1)
      GO TO 999
  100 NX=ILAS
  999 END
      SUBROUTINE XINBF(LUN,XV,NX)
C     ***************************************************************************
C     *                                                                         *
C     *     INTERRUPTION FOR FILLING BUFFER IN INPUT BY XINBF                   *
C     *                                                                         *
C     *      CALLED BY  FZINA,....                                              *
C     *            J-L DEKEYSER              7.2.86                             *
C     *                                                                         *
C     ***************************************************************************
+SEQ,BUFIN.
+SEQ,MESSAG
      DIMENSION XV(NX)
      IMESS(1)=10
      IMESS(2)=LUN
      IMESS(3)=0
      IMESS(4)=NX
      CALL BREAK
      IMESS(1)=0
      IF (ILAS.LE.0) GO TO 100
      CALL UCOPY(IIBUF,XV,ILAS)
  100 NX=ILAS
  999 END
+DECK,BUFEMU,IF=-EMUL
      SUBROUTINE GHBGSC(IPROC)
C     ***************************************************************************
C     *                                                                         *
C     * BEGIN OF  MUTUAL EXCLUSION FOR ACCESS AT THE SAME FILE                  *
C     *     FROM 2 EMULATORS = P(SEMAPHORE)                                     *
C     *      CALLED BY  GSAVE,GGET                                              *
C     *            J-L DEKEYSER              7.2.86                             *
C     *                                                                         *
C     ***************************************************************************
+SEQ,MESSAG
+SEQ,GCPARA
      TROU=0
```

```
C
C     RESEARCH IF FILE FREE?
C
      DO 10 I=1,NPROC
        IF (LSCT(I).EQ.IMESS(2)) TROU=1
   10 CONTINUE
      IF (TROU.EQ.1) THEN
          LSCT(IPROC)=-IMESS(2)
        ELSE
          LSCT(IPROC)=IMESS(2)
          CALL SYEPUT(IPROC,'MESSAG   ',IBRK  ,'CONT',IRET,0,4)
      END IF
      END
      SUBROUTINE GHNDSC(IPROC)
C     ***************************************************************************
C     *                                                                         *
C     *   END OF  MUTUAL EXCLUSION FOR ACCESS AT THE SAME FILE *
C     *    FROM 2 EMULATORS = V(SEMAPHORE)                                      *
C     *     CALLED BY  GSAVE,GGET                                               *
C     *           J-L DEKEYSER              7.2.86                              *
C     *                                                                         *
C     ***************************************************************************
+SEQ,MESSAG.
+SEQ,GCPARA
      TROU=LSCT(IPROC)
C
C     RELEASE THE FILE
C
      LSCT(IPROC)=0
      NP=0
C
C     UNBLOCK ONE EMULATOR WAITING FOR THIS FILE
C
      DO 10 I=1,NPROC
        IF (LSCT(I).EQ.-TROU) NP=I
   10 CONTINUE
      IF (NP.EQ.0) GO TO 99
      CALL SYEPUT(NP,'MESSAG   ',IBRK  ,'CONT',IRET,0,4)
      LSCT(NP)=TROU
   99 END
      SUBROUTINE GHEMPT(IPROC)
C     ***************************************************************************
C     *                                                                         *
C     *    WRITE THE BUFFER ON DISK                                             *
C     *                                                                         *
C     *     CALLED BY  GRUN ON 4361                                             *
C     *           J-L DEKEYSER              7.2.86                              *
```

```
C    *                                                                    *
C    **********************************************************************
+SEQ,BUFOUT
+SEQ,MESSAG
C
C     SEND THE BUFFER
C
      CALL SYEGET(IPROC,'BUFOUT  ',LU,'RDM ',IRET,0,IMESS(2))
C
C     RESTART THE EMULATOR
C
      CALL SYEPUT(IPROC,'MESSAG   ',IBRK  ,'CONT',IRET,0,4)
      IF (IPF.LE.0) GO TO 99
      N=IPF-1
      NA=1
   10 WRITE(LU) IOBUF(NA),(IOBUF(I),I=NA+1,N)
        NA=N+2
        N=IOBUF(N+1)-1
      IF (IOBUF(NA-1).NE.-1) GO TO 10
  99  END
      SUBROUTINE GHFILL(IPROC)
C    **********************************************************************
C    *                                                                    *
C    *    READ THE DISK AND FILL THE BUFFER IN INPUT                      *
C    *                                                                    *
C    *     CALLED BY  GRUN ON 4361                                        *
C    *           J-L DEKEYSER              7.2.86                         *
C    *                                                                    *
C    **********************************************************************
+SEQ,MESSAG
+SEQ,BUFIN
      ILAS=IMESS(4)
      LI=IMESS(2)
      NA=IMESS(3)
      IF (NA.EQ.0) THEN
          CALL XINBF(LI,IIBUF,ILAS)
        ELSE
          CALL XINBS(LI,IIBUF(1),NA,IIBUF(1+NA),ILAS)
      END IF
      IPC=1
C
C     SEND OF  BUFFER
C
      ISIZ=12
      IF (ILAS.GT.0) ISIZ=(NA+ILAS)*4 +12
      CALL SYEPUT(IPROC,'BUFIN   ',LI,'CONT',IRET,0,ISIZ)
      END
```

```
+DECK,GGET,IF=EMUL
      SUBROUTINE GGET (LUN,KEYSU,NUKEYS,IDENT,IER)
C.
C.    ************************************************************************
C.    *                                                                      *
C.    *       Routine to read in data structures                             *
C.    *                                                                      *
C.    *       LUN      Logical unit number                                   *
C.    *       KEYSU    Keywords to select data structures                    *
C.    *       NKEYS    Number of keywords                                    *
C.    *       IER      Error flag                                            *
C.    *                                                                      *
C.    *    ==>Called by : <USER>, UGINIT,GUKINE                              *
C.    *       Author    R.Brun  **********                                   *
C.    *                                                                      *
C.    ************************************************************************
C.
+SEQ,GCBANK
+SEQ,GCFLAG
+SEQ,GCNUM
+SEQ,GCVOLU
+SEQ,GCDRAW
+SEQ,GCCUTS
+SEQ,GCPHYS
+SEQ,MESSAG
+SEQ,GCPARA
      COMMON/QUEST/IQUEST(100)
C
      CHARACTER*4 KLEY(22)
      DIMENSION KEYSU(1),KEYS(22),IUHEAD(2)
      DIMENSION KSEL(14),LKEY(22),LKNUM(22),LINK(14),JLINK(17)
      EQUIVALENCE (JLINK(1),JDIGI)
      DATA LINK/7,6,13,16,8,10,2,9,3,15,5,17,4,1/
      DATA KLEY/'PART','MATE','TMED','VOLU','ROTM','SETS','DRAW','RUNG'
     +         ,'INIT','INIT','INIT','INIT','INIT','INIT','INIT','INIT'
     +         ,'HEAD','KINE','KINE','JXYZ','HITS','DIGI'/
      DATA LKNUM/1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,9,10,11,12,13,14/
      DATA IFIRST/0/
C.
C.    ------------------------------------------------------------------
C.
      IF(IFIRST.EQ.0)THEN
         IFIRST=1
         CALL UCTOH(KLEY,LKEY,4,88)
      ENDIF
C
      IDENT=-1
```

```
      IER    = 0
      NKEYS=IABS(NUKEYS)
      IF (NKEYS.LE.0)                                    GO TO 999
      CALL UCTOH(KEYSU,KEYS,4,4*NKEYS)
C
      IF(NUKEYS.LT.0)THEN
        I1=1
        I2=15
        K1=1
        K2=7
      ELSE
        I1=18
        I2=22
        K1=10
        K2=14
      ENDIF
C
      NKT=0
      DO 10 K=K1,K2
   10 KSEL(K)=0
      DO 20 I=I1,I2
      N=LKNUM(I)
      DO 20 IK=1,NKEYS
      IF(KEYS(IK).EQ.LKEY(I))THEN
         KSEL(N)=1
         NKT=NKT+1
      ENDIF
   20 CONTINUE
      IF(NKT.EQ.0)GO TO 999
      NUH=2
C
C     BEGIN OF CRITICAL SECTION IF MORE THAN ONE EMULATORS
C
      IF (NPROC.NE.1) THEN
        IMESS(1)=8
        IMESS(2)=LUN
        CALL BREAK
        IMESS(1)=0
      END IF
C
C               Go for next start of event data structure
C
      IF(NUKEYS.LT.0)THEN
         IF(JRUNG.NE.0)CALL MZDROP(IXCONS,JRUNG,' ')
         CALL FZIN(LUN,IXCONS,JRUNG,1,'E',NUH,IUHEAD)
         IF(IQUEST(1).GT.2)GO TO 90
         IDIV=IXCONS
```

```
      ELSE
         IF(JHEAD.NE.0)CALL MZDROP(IXDIV,JHEAD,' ')
         CALL FZIN(LUN,IXDIV,JHEAD,1,'E',NUH,IUHEAD)
         IF(IQUEST(1).GT.2)GO TO 90
         IDIV=IXDIV
      ENDIF
C
      IDENT= IUHEAD(1)
      NK   = IUHEAD(2)
      IF(NK.LE.0)GO TO 99
      IF(NK.GT.10)GO TO 99
      DO 30 I=1,NK
C
C              Read next header
C
         NUH=2
         CALL FZIN(LUN,IDIV,0,0,'S',NUH,IUHEAD)
         IF(IQUEST(1).GT.2)GO TO 90
         KS=IUHEAD(1)
         IF(KS.LE.0)GO TO 30
         IF(KS.GT.14)GO TO 30
         IF(KSEL(KS).EQ.0)GO TO 30
         IL=LINK(KS)
         IF(JLINK(IL).NE.0)CALL MZDROP(IDIV,JLINK(IL),' ')
C
C              Read pending data structure
C
         CALL FZIN(LUN,IDIV,JLINK(IL),1,'A',NUH,IUHEAD)
         IF(IQUEST(1).GT.2)GO TO 90
   30 CONTINUE
C
C             Fill header bank
C             Reconstruct NKVIEW,NVOLUM,NVERTX,NTRACK
C
      IF(NUKEYS.LT.0)THEN
         IF(KSEL(3).NE.0.AND.JTMED.NE.0)CALL UCOPY(Q(JTMED+1),CUTGAM,9)
         IF(KSEL(4).NE.0.AND.JVOLUM.GT.0)NVOLUM=IQ(JVOLUM-1)
         IF(KSEL(7).NE.0.AND.JDRAW.GT.0) NKVIEW=IQ(JDRAW-2)
      ENDIF
C
      IF(JHEAD.GT.0)THEN
         IDRUN=IQ(JHEAD+1)
         IDEVT=IQ(JHEAD+2)
      ENDIF
C
      IF(KSEL(10).GT.0)THEN
         NVERTX=0
```

```
          NTRACK=0
          IF(JVERTX.GT.0)NVERTX=IQ(JVERTX+1)
          IF(JKINE .GT.0)NTRACK=IQ(JKINE +1)
        ENDIF
        GO TO 99
C
C               Error, EOF,etc
C
  90    IER=IQUEST(1)
C
C
C
C       END OF CRITICAL SECTION IF MORE THAN ONE EMULATORS
C
  99    IF (NPROC.NE.1) THEN
          IMESS(1)=18
          CALL BREAK
          IMESS(1)=0
        END IF
  999   END
+DECK,GSAVE,IF=EMUL
        SUBROUTINE GSAVE(LUN,KEYSU,NUKEYS,IDENT,IER)
C.
C.      ************************************************************************************
C.      *                                                                                  *
C.      *       Routine to write out data structures                                       *
C.      *                                                                                  *
C.      *       LUN      Logical unit number                                               *
C.      *       KEYSU    Keywords to select data structures                                *
C.      *       NKEYS    Number of keywords                                                *
C.      *       IER      Error flag                                                        *
C.      *                                                                                  *
C.      *    ==>Called by : <USER>, UGINIT,GUOUT                                           *
C.      *       Author    R.Brun  **********                                              *
C.      *                                                                                  *
C.      ************************************************************************************
C.
+SEQ,GCBANK
+SEQ,MESSAG
+SEQ,GCPARA
+SEQ,BUFOUT
        CHARACTER*4 KLEY(22)
        DIMENSION KEYSU(1),KEYS(22),IUHEAD(2)
        DIMENSION KSEL(14),LKEY(22),LKNUM(22),LINK(14),JLINK(17)
        EQUIVALENCE (JLINK(1),JDIGI)
        DATA LINK/7,6,13,16,8,10,2,9,3,15,5,17,4,1/
        DATA KLEY/'PART','MATE','TMED','VOLU','ROTM','SETS','DRAW','RUNG'
```

```
     +         ,'INIT','INIT','INIT','INIT','INIT','INIT','INIT','INIT'
     +         ,'HEAD','KINE','KINE','JXYZ','HITS','DIGI'/
      DATA LKNUM/1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,9,10,11,12,13,14/
      DATA IFIRST/0/
C.
C.    ------------------------------------------------------------------
C.
      IF(IFIRST.EQ.0)THEN
         IFIRST=1
         CALL UCTOH(KLEY,LKEY,4,88)
      ENDIF
C
C
      IER    = 0
      NKEYS=IABS(NUKEYS)
      IF (NKEYS.LE.0)                                    GO TO 99
      CALL UCTOH(KEYSU,KEYS,4,4*NKEYS)
C
      IF(NUKEYS.LT.0)THEN
        I1=1
        I2=15
        K1=1
        K2=7
      ELSE
        I1=18
        I2=22
        K1=10
        K2=14
      ENDIF
C
      DO 10 K=K1,K2
   10 KSEL(K)=0
      NK=0
      DO 25 I=I1,I2
         N=LKNUM(I)
         DO 20 IK=1,NKEYS
            IF(KEYS(IK).EQ.LKEY(I))THEN
               IL=LINK(N)
               IF(JLINK(IL).NE.0)THEN
                  KSEL(N)=1
                  NK=NK+1
               ENDIF
            ENDIF
   20    CONTINUE
   25 CONTINUE
C
      IUHEAD(1)=IDENT
```

```
      IUHEAD(2)=NK
      IF(NUKEYS.LT.0)THEN
C
C======>      Write RUN header and constants
C
         CALL FZOUT(LUN,IXCONS,JRUNG,1,' ',2,2,IUHEAD)
         DO 30 I=1,7
            IF(KSEL(I).NE.0)THEN
               IL=LINK(I)
               CALL FZOUT(LUN,IXCONS,JLINK(IL),0,' ',2,1,I)
            ENDIF
  30     CONTINUE
C
      ELSE
C
C======>      Write event header and data structures
C             Released unused space in JHITS and JDIGI
C
         IF(KSEL(13).NE.0)CALL GRLEAS(JHITS)
         IF(KSEL(14).NE.0)CALL GRLEAS(JDIGI)
C
         CALL FZOUT(LUN,IXDIV,JHEAD,1,' ',2,2,IUHEAD)
         DO 40 I=10,14
            IF(KSEL(I).NE.0)THEN
               IL=LINK(I)
               CALL FZOUT(LUN,IXDIV ,JLINK(IL),0,' ',2,1,I)
            ENDIF
  40     CONTINUE
C
      ENDIF
C
C
C     END OF CRITICAL SECTION IF MORE THAN ONE EMULATORS
C
      IF (IPREM.NE.0) THEN
        CALL GEMPSC
      END IF
  99  END
+DECK,DUMMY,IF=EMUL
      SUBROUTINE HMACHI
      END
      SUBROUTINE REENT
      END
      SUBROUTINE ERRTRA
      END
      SUBROUTINE READMS
      END
```

```
      SUBROUTINE WRITMS
      END
+PATCH,DOCEMU
C        DOCUMENTATION OF PATCH GEMUL OF GEANT3 FOR EMULATORS 3081/E
C
C
C      RULES FOR IMPLEMENTING AN APPLICATION OF GEANT3 ON 3081/E
C      ==========================================================
C
C     PHASE 1
C     =======
C     MODIFICATION OF YOUR APPLICATION.
C
C           1. BEGIN OF PROGRAM
C                +SELF,IF=EMUL
C                       SUBROUTINE EMUSER
C                +SELF,IF=-EMUL
C                       PROGRAM MAIN
C                +SELF
C
C           2. MODIFY UGINIT
C                AFTER CALL GINIT , INSERT
C                +SELF,IF=EMUL
C                       CALL EGINIT
C                +SELF
C                BEFORE CALL FFGO , INSERT
C                +SELF,IF=EMUL
C                       CALL EFFGO
C                +SELF
C
C           3. MODIFY UGLAST
C                AFTER CALL GLAST , INSERT
C                +SELF,IF=EMUL
C                       CALL GUHHIS
C                +SELF
C
C           4. WIPPE ALL YOUR CALLS TO THE DRAWING PACKAGE
C                +SELF,IF=-EMUL
C                       CALL TVBGN(3)
C                       ............
C                +SELF
C                OR DON'T USE OPTION GKS,PIG,..
C
C           5. LOCAL VARIABLES
C                IF YOU USE A LOT OF LOCAL VARIBLES IN A ROUTINE
C                YOU HAVE TO DEFINE A COMMON BLOCK FOR THESE VARIABLES
C
```

```
C       PHASE 2
C       =======
C       CREATE THE LIBRARY GGEN TXTLIB
C             CREATION OF GEANT TEXT
C
TRACE       C
/*             Patchy step                                          */
LOADMOD     YPATCH$M
'FILEDEF     12 DISK GEANT PAM *'
'FILEDEF      5 DISK GEANT CRADLE *'
FILEDEF     21 DISK GEANT FORTRAN A '('RECFM F BLKSIZE 80
START       YPATCHY
/*             Compile                                              */
'FORTVS      GEANT ( NOPRINT IL(DIM)'
ERASE       GEANT FORTRAN A
CHOP        GEANT TEXT
EXIT
C
C          FILE GEANT CRADLE
C
+EXE.
+USE, IBM.
+USE,P=GCDES
+USE,P=GBASE
+USE,P=GCONS
+USE,P=GHITS
+USE,P=GIOPA
+USE,P=GKINE
+USE,P=GPHYS
+USE,P=GTRAK
+USE,P=GUSER
+PAM,12
+QUIT.
C
C            CREATION OF GEANG TEXT
C
TRACE       C
/*             Patchy step                                          */
LOADMOD     YPATCH$M
'FILEDEF     11 DISK GEANT PAM *'
'FILEDEF     12 DISK GEANZ PAM *'
'FILEDEF      5 DISK GEANG CRADLE *'
FILEDEF     21 DISK GEANG FORTRAN A '('RECFM F BLKSIZE 80
START       YPATCHY
/*             Compile                                              */
'FORTVS      GEANG ( NOPRINT IL(DIM)'
ERASE       GEANG FORTRAN A
```

```
/*              Create the library                                    */
CHOP         GEANG TEXT
EXIT
C
C         FILE GEANG CRADLE
C
+EXE.
+USE, IBM.
+USE,P=GCDES
+USE,P=GDRAW.
+USE,P=GGEOM.
+PAM,11,R=GCDES.
+PAM,12.
+QUIT.
+DECK, DUMMY.
      SUBROUTINE TVBGN(A)
      END
      SUBROUTINE TVNEXT(A)
      END
      SUBROUTINE TVSHOW(A)
      END
      SUBROUTINE TVCAP(A)
      END
      SUBROUTINE TVRNG(A)
      END
      SUBROUTINE TVHRNG(A)
      END
      SUBROUTINE HPLDRW(A)
      END
      SUBROUTINE HPLSOF(A)
      END
      SUBROUTINE TVPLOT(A)
      END
+QUIT.
C
C             CREATION OF GGEN TXTLIB
C
TRACE        OFF
'COPYFILE GEALIB TXTLIB A GGEN TXTLIB A'
TXTLIB       ADD GGEN    GEANGZ GEANTZ
EXIT
C
C       PHASE 3
C       =======
C       CREATION OF 2 LIBRARIES GHOST AND GEMULS
C
C             1. CREATE THE FORTRAN FILE WITH YOUR APPLICATION
```

```
C               USING THE OPTION EMUL AND COMPILE IT ( OPTION IL(DIM) )
C               YOU OBTAIN THE FILE GXXXX TEXT
C
/*             TO MAKE GEXAM1 LIBRARIES EXEC                          */
TRACE       C
/*             PATCHY STEP                                            */
LOADMOD     YPATCH$M
'FILEDEF * CLEAR'
'FILEDEF     11 DISK GEANT PAM *'
'FILEDEF     12 DISK GEANX PAM *'
'FILEDEF     13 DISK GEMUL PAM *'
'FILEDEF     5  DISK GEXAM1 CRADLE *'
FILEDEF     21 DISK GEXAM1 FORTRAN A '('RECFM F BLKSIZE 80
START       YPATCHY
/*             COMPILE                                                */
'FORTVS      GEXAM1 (NOPRINT FLAG(E) IL(DIM)'
ERASE       GEXAM1 FORTRAN A
/*             CREATE THE LIBRARY                                     */
CHOP        GEXAM1 TEXT
EXIT
C
C            FILE GEXAM1 CRADLE
C
+EXE.
+OPT,MAP.
+USE,GHEISHA,EMUL.
+USE,GCDES.
+USE,GECDES.
+USE,GEXAM1.
+use,GEX1DATA,T=EXE
+ASM, 23.
+PAM,11.
+PAM,13.
+PAM,12.
+QUIT.
C
C            2. CREATE THE FORTRAN FILE WITH PATCH GEMUL USING THE
C               OPTION EMUL AND COMPILE IT.
C               AND AFTER YOU CAN CREATE GSLAVE TXTLIB
C
TRACE       C
LOADMOD     YPATCH$M
'FILEDEF * CLEAR'
'FILEDEF     12 DISK GEMUL PAM *'
'FILEDEF     11 DISK GEANT PAM *'
QUEUE "+EXE."
QUEUE "+USE, IBM, EMUL."
```

```
QUEUE "+USE,P=GECDES"
QUEUE "+USE,P=GEMUL"
QUEUE "+USE,P=GCDES"
QUEUE "+PAM,11."
QUEUE "+PAM,12."
QUEUE "+QUIT."
FILEDEF     21 DISK GEMUL FORTRAN A '('RECFM F BLKSIZE 80
START       YPATCHY
/*             Compile                                          */
'FORTVS      GEMUL ( NOPRINT IL(DIM)'
ERASE       GEMUL FORTRAN A
/*             Create the library                               */
CHOP        GEMUL TEXT
/*             Patchy step                                      */
TXTLIB      GEN GSLAVE GEMUL GEXAM1 WRT
EXIT
C
C            3. CREATE THE FORTRAN FILE WITH PATCH GEMUL WITHOUT
C               OPTION EMUL AND COMPILE IT.
C               AND AFTER YOU CAN CREATE GHOST TXTLIB
C
'FILEDEF * CLEAR'
TRACE       C
LOADMOD     YPATCH$M
'FILEDEF     12 DISK GEMUL PAM *'
'FILEDEF     11 DISK GEANT PAM *'
QUEUE "+EXE."
QUEUE "+USE, IBM."
QUEUE "+USE,P=GECDES"
QUEUE "+USE,P=GEMUL"
QUEUE "+USE,P=GCDES"
QUEUE "+PAM,11."
QUEUE "+PAM,12."
QUEUE "+QUIT."
FILEDEF     21 DISK GEMUL FORTRAN A '('RECFM F BLKSIZE 80
START       YPATCHY
/*             COMPILE                                          */
'FORTVS      GEMUL ( NOPRINT IL(DIM)'
ERASE       GEMUL FORTRAN A
/*             Create the library                               */
CHOP        GEMUL TEXT
/*             Patchy step                                      */
TXTLIB      GEN GHOST  GEMUL GEXAM1
EXIT
C
C      PHASE 4
C      =======
```

```
C       CREATION OF MAIN PROGRAM FOR EMULATORS
C
C             1. INITIALIZE THE SIZE OF COMMONS GCBANK AND BLANKCOM
C
C             2. CHOOSE AN ORDER FOR ALL YOUR COMMON PRESENT IN
C                EMULATORS AND HOST. THE 2 COMMONS BUFFIN AND BUFFOUT
C                WILL BE PUT AT THE END .
C
+PATCH,EMUL
+DECK,MAIN
+SEQ,MESSAG
+SEQ,GCPARA
+SEQ,GCBANK
+SEQ,GCCUTS
  ..........
  ..........
+SEQ,GCLUND
C
C
      NRANDO=IMESS(1)
      NEVENT=IMESS(2)
      IMESS(1)=0
      CALL USER
      CALL RDMIN(NRANDO)
      CALL ZEBEMU(LQ)
      CALL GRUN
      CALL RDMOUT(NRANDO)
      IF (NSAVE.GT.0.AND.NPROC.EQ.1) CALL GEMPTI
  99  END
      SUBROUTINE USER
      COMMON//HBOOK(2000)
      COMMON/GCBANK/GEANTS(50000)
      END
C
C             3. COMPILE IT AND OBTAIN THE FILE EMUL TEXT
C
C       PHASE 5
C       =======
C       CREATION OF MAIN PROGRAM FOR HOST
C
C             1. DEFINE THE SAME ORDER FOR THE COMMONS AND CALL EMUSER
C
+PATCH,S4361
+DECK,MAIN
+SEQ,MESSAG
+SEQ,GCPARA
+SEQ,GCBANK
```

```
  .........
  .........
+SEQ,GCLUND
C
       COMMON/      / BID1
       COMMON/MZCA  / BID3
       COMMON/MZCB  / BID4
       COMMON/MZCC  / BID5
       ..................
       COMMON/SLATE / BID20
       COMMON/ZTELLC/ BID21
C
      CALL EMUSER
      END
C
C             2. COMPILE IT AND OBTAIN S4361 TEXT
C
C      PHASE 6
C      =======
C      TRANSLATION
C
C             1. PROCEDURE FOR TRANSLATION
C
TRACE       OFF
GLOBAL TXTLIB GSLAVE GGEN NFORTLIB EKERNLIB,
ZEBRA PACKLIB GENLIB
LOADE EMUL
TRANSL EMUL LOADE '(' NOPIBM NOPPROG NOPSTATS PIPELN MARPIP
EXIT
C
C      PHASE 7
C      =======
C      EXECUTION
C
C             1. PROCEDURE FOR EXECUTION
C
'FILEDEF * CLEAR'
'FILEDEF 10 DISK EMUL LINKOUT *'
'FILEDEF 11 DISK DMINIT LINKOUT *'
'FILEDEF 5  DISK GEXAM1  DATA *'
'FILEDEF 6 DISK RESEXAM1 DATA A'
'FILEDEF 35 DISK EMUL FWORK A'
'FILEDEF 7 TERMINAL'
TRACE OFF
GLOBAL TXTLIB GHOST GGEN ZEBRA PACKLIB,
CAMLIB CMSLIB VFORTLIB genlib
'GLOBAL LOADLIB VFLODLIB'
```

```
'LOAD S4361 SYEPAK M233 (START'
C
C      RESTRICTION
C      ===========
C
C            1. ALL THE STATEMENTS OF INPUT/OUPUT ARE WIPPED.
C               ONLY GGET AND GSAVE CAN READ OR WRITE THE DATA STRUCTURES
C
C            2. THE SIZE OF BUFFER OF OUPUT CAN BE INCREAZED
C               IF THE MEMORY OF EMULATORS IS NOT COMPLETLY USED.
C               BUT THE MINIMAL SIZE IS THE SAME SIZE THAN LOGICAL
C               RECORD THE DEFAULTS IS 4096 WORDS AND CAN BE MODIFIED
C               WHEN YOU CALL GOPEN. IF THE BUFFER IS SMALL THE
C               EMULATOR WILL BE STOPPED FEW TIMES BY EVENT, IF IT IS
C               BIG IT WILL BE STOPPED ONE TIME FOR FEW EVENT.
```

APPENDICE C

VECTORISATION DE LA GÉOMÉTRIE

Présentation de quelques procédures de la géométrie de GEANT3 et de leurs transformations en procédures vectorielles sur Cyber 205 .

- Vectorisation de GNEXT

Algorithme séquentiel

```
      DO 120 IN=1,NIN
         JIN=LQ(JVO-IN)
         IROT=Q(JIN+4)
         IF(IROT.EQ.0)THEN
            DO 90 I=1,3
               XT(I)=XC(I)-Q(JIN+4+I)
 90         XT(I+3)=XC(I+3)
         ELSE
            CALL GITRAN(XC,Q(JIN+5),IROT,XT)
            CALL GRMTD(XC(4),IROT,XT(4))
         ENDIF
         IVOT=Q(JIN+2)
         JVOT=LQ(JVOLUM-IVOT)
C
         NPAR=Q(JVOT+5)
         IF(NPAR.GT.0)THEN
            DO 100 I=1,NPAR
100         SPAR(I)=Q(JVOT+6+I)
         ELSE
            NPAR=Q(JIN+9)
            DO 110 I=1,NPAR
110         SPAR(I)=Q(JIN+9+I)
         ENDIF
         PN=NPAR
C
         CALL GPCHK(Q(JVOT+2),PN,SPAR,Q(JIN+5),IROT, Q(JVO+2),NSPAR,
     +   SPAR,0,0.0)
C
C                THIS SECTION CONTINUES ON THE ASSUMPTION THAT WE
C                ARE OUTSIDE THE OTHER SHAPES.
C
         ISHT=Q(JVOT+2)
         CALL GNSOUT(XT,ISHT,SPAR IDIST,SNEXT,SAFE)
         IF(SAFE.LT.SAFETY) SAFETY=SAFE
C
120   CONTINUE
```

Algorithme vectoriel

```
      IF (JVO.NE.JVOLD) THEN
      JVECT=LV(LIMITV-IVO+1)
      IFL=0
        IF (JVECT.LT.0) THEN
           JVECT=-JVECT+1
           IFL=1
        ELSE
           JVECT=JVECT+1
        END IF
      NIN2=2*NIN
      NIN5=5*NIN
      NIN6=6*NIN
      NIN15=15*NIN
      JVOLD=JVO
      IDIF=1
      END IF
      CALL GVTRAN(XC,V(JVECT+NIN2),V(JVECT+NIN6),XTV,NIN,IFL)
      CALL GVRMTD(XC(4),V(JVECT+NIN6),XTV(1,4),NIN,IFL)
      IF (ISEARC.NE.2) GO TO 81
C     PRINT *,V(JVECT+NIN15),V(JVECT+NIN15+NIN)
      CALL GVSOUT(XTV,LV(JVECT),V(JVECT+NIN15),
     +       IDIST,SNEXT,SAFE,NIN)
      IF (SAFE.LT.SAFETY) SAFETY=SAFE

      GO TO 210
```

- Vectorisation de GITRAN

Algorithme séquentiel

```
      SUBROUTINE GITRAN(X,DX,IROT,XNEW)
C.
C.    ************************************************************************
C     *                                                                      *
C     *        ROUTINE TO APPLY THE TRANSFORMATION GIVEN BY                  *
C     *        THE POINT DX AND THE ROTATION MATRIX IN THE GEANT             *
C     *        ROTATION MATRIX BANKS WITH INDEX NUMBER IROT TO THE           *
C     *        POINT X, STORING THE RESULT AS THE POINT XNEW.                *
C     *                                                                      *
C.    *    ==>Called by : GHDIV,GMTST,GNEXT,GSEAR                           *
C.    *         Author  A.McPherson  **********                              *
C.    *                                                                      *
C.    ************************************************************************
C.
+SEQ,GCBANK
      DIMENSION X(3),DX(3),XNEW(3),XL(3)
C.
C.    ------------------------------------------------------------------
C.
      DO 10 I=1,3
      XNEW(I)=X(I)-DX(I)
   10 CONTINUE
      IF(IROT.EQ.0)GO TO 99
      DO 20 I=1,3
   20 XL(I)=XNEW(I)
      JR=LQ(JROTM-IROT)
      DO 30 I=1,3
   30 XNEW(I)=XL(1)*Q(JR+3*I-2)+XL(2)*Q(JR+3*I-1)+XL(3)*Q(JR+3*I)
C
   99 END
```

Algorithme vectoriel

```
      SUBROUTINE GVTRAN(X,DX,Q,XNEW,NIN,IFL)
C
C
C
C
      PARAMETER (MAXIN=300)
C
C
      DIMENSION X(3),DX(3*NIN),Q(*),XNEW(*),XL(3*MAXIN)
C
C
      XNEW(1;NIN)=X(1)-DX(1;NIN)
      M1=MAXIN+1
      XNEW(M1;NIN)=X(2)-DX(NIN+1;NIN)
      M2=MAXIN*2+1
      XNEW(M2;NIN)=X(3)-DX(2*NIN+1;NIN)
C
      IF (IFL.EQ.0) GO TO 999
      XL(1;3*MAXIN)=XNEW(1;3*MAXIN)
      XNEW(1;NIN)=XL(1;NIN)*Q(1;NIN)+XL(M1;NIN)*Q(NIN+1;NIN)
     +    +XL(M2;NIN)*Q(2*NIN+1;NIN)
      XNEW(M1;NIN)=XL(1;NIN)*Q(3*NIN+1;NIN)+XL(M1;NIN)*Q(4*NIN+1;NIN)
     +    +XL(M2;NIN)*Q(5*NIN+1;NIN)
      XNEW(M2;NIN)=XL(1;NIN)*Q(6*NIN+1;NIN)+XL(M1;NIN)*Q(7*NIN+1;NIN)
     +    +XL(M2;NIN)*Q(8*NIN+1;NIN)
  999 END
```

- Vectorisation de GRMTD

Algorithme séquentiel

```
      SUBROUTINE GRMTD(XMAS,IROT,XNEW)
C.
C.    ************************************************************************
C.    *                                                                      *
C.    *    ROUTINE ROTATES A POINT IN XMAS FROM THE MASTER SYSTEM            *
C.    *    TO A NEW SYSTEM USING MATRIX # IROT.                              *
C.    *    RESULT IS RETURNED IN VECTOR XNEW.                                *
C.    *                                                                      *
C.    *    ==>Called by : GNEXT                                              *
C.    *         Author  A.McPherson  **********                              *
C.    *                                                                      *
C.    ************************************************************************
C.
+SEQ,GCBANK
      DIMENSION XMAS(3),XNEW(3)
C.
C.    ------------------------------------------------------------------
C.
      IF (IROT.LE.0) GO TO 10
      JR = LQ(JROTM- IROT)
C
      DO 5 I=1,3
    5 XNEW(I)=XMAS(1)*Q(JR+3*I-2)+XMAS(2)*Q(JR+3*I-1)+XMAS(3)*Q(JR+3*I)
C
      RETURN
C
   10 CONTINUE
      DO 20 I=1,3
   20 XNEW(I)=XMAS(I)
C
      END
```

Algorithme vectoriel

```
      SUBROUTINE GVRMTD(XMAS,Q,XNEW,NIN,IFL)
C
C
C
C
      PARAMETER (MAXIN=300)
C
C
      DIMENSION XMAS(3),Q(*),XNEW(*)
C
      M1=MAXIN+1
      M2=MAXIN*2+1
      IF (IFL.NE.0) THEN
      XNEW(1;NIN)=XMAS(1)*Q(1;NIN)+XMAS(2)*Q(NIN+1;NIN)
     +     +XMAS(3)*Q(2*NIN+1;NIN)
      XNEW(M1;NIN)=XMAS(1)*Q(3*NIN+1;NIN)+XMAS(2)*Q(4*NIN+1;NIN)
     +     +XMAS(3)*Q(5*NIN+1;NIN)
      XNEW(M2;NIN)=XMAS(1)*Q(6*NIN+1;NIN)+XMAS(2)*Q(7*NIN+1;NIN)
     +     +XMAS(3)*Q(8*NIN+1;NIN)
      ELSE
      XNEW(1;NIN)=XMAS(1)
      XNEW(M1;NIN)=XMAS(2)
      XNEW(M2;NIN)=XMAS(3)
      END IF
C
      END
```

- Vectorisation de GNOBOX

Algorithme séquentiel

```
      SUBROUTINE GNOBOX(X,PAR,IDIST,SNEXT,SAFE)
C.
C.                    ************************************************************
C.                    *                                                          *
C.                    *   ROUTINE TO COMPUTE DISTANCE FROM THE POINT             *
C.                    *   X(1-3) ALONG THE VECTOR X(4-6) TO THE BOX              *
C.                    *   WITH PARAMETERS PAR WHEN THE POINT IS                  *
C.                    *   OUTSIDE THE BOX. IF IDIST EQ 0 OR THE NEW              *
C.                    *   DISTANCE IS LESS THAN THE CURRENT VALUE OF             *
C.                    *   SNEXT THEN SNEXT IS SET TO THE NEW                     *
C.                    *   DISTANCE AND IDIST IS SET TO 1.                        *
C.                    *                                                          *
C.                    *    ==>Called by : GNSOUT                                 *
C.                    *         Author  A.McPherson  **********                  *
C.                    ************************************************************
C.
+SEQ,GCONST
      DIMENSION X(6),PAR(3),SAF(3)
      EQUIVALENCE (SAF(1),SAF1),(SAF(2),SAF2),(SAF(3),SAF3)
C.
C.    ------------------------------------------------------------
C.
      SAFE=BIG
      DO 5 I=1,3
      SAF(I)=ABS(X(I))-PAR(I)
```

```
      IF(SAF(I).GT.0.)THEN
         IF(SAF(I).LT.SAFE)SAFE=SAF(I)
      ENDIF
    5 CONTINUE
C
C             SAFE REGION COMPUTED NOW DO SNEXT.
C
      IF(X(1)*X(4).GE.0.0.AND.SAF1.GT.0.) GO TO 999
      IF(X(2)*X(5).GE.0.0.AND.SAF2.GT.0.) GO TO 999
      IF(X(3)*X(6).GE.0.0.AND.SAF3.GT.0.) GO TO 999
C
      SMAX1=BIG
      SMAX2=BIG
      SMAX3=BIG
      SMIN1=0.0
C
      IF(SAF1.LT.0.) GO TO 10
      IF(X(4).EQ.0.0) GO TO 999
      S4=ABS(X(4))
      SMIN1=SAF1/S4
      SMAX1=(PAR(1)+ABS(X(1)))/S4
      GO TO 20
C
   10 CONTINUE
      IF(X(4).NE.0.)THEN
         SMAX1=PAR(1)/ABS(X(4))-X(1)/X(4)
      ENDIF
   20 CONTINUE
C
      SMIN2=0.0
      IF(SAF2.LT.0.) GO TO 30
      IF(X(5).EQ.0.0) GO TO 999
      S5=ABS(X(5))
      SMIN2=SAF2/S5
      SMAX2=(PAR(2)+ABS(X(2)))/S5
      GO TO 40
C
   30 CONTINUE
      IF(X(5).NE.0.)THEN
         SMAX2=PAR(2)/ABS(X(5))-X(2)/X(5)
      ENDIF
   40 CONTINUE
C
      SMIN3=0.0
      IF(SAF3.LT.0.) GO TO 50
      IF(X(6).EQ.0.0) GO TO 999
      S6=ABS(X(6))
      SMIN3=SAF3/S6
      SMAX3=(PAR(3)+ABS(X(3)))/S6
      GO TO 60
C
   50 CONTINUE
      IF(X(6).NE.0.)THEN
         SMAX3=PAR(3)/ABS(X(6))-X(3)/X(6)
      ENDIF
   60 CONTINUE
```

```
C
      SMIN=AMAX1(SMIN1,SMIN2,SMIN3)
      SMAX=AMIN1(SMAX1,SMAX2,SMAX3)
      IF(SMAX.LT.SMIN) GO TO 999
C
      IF(SMIN.LE.0.)GO TO 999
      IF(IDIST.EQ.0.OR.SMIN.LT.SNEXT) SNEXT=SMIN
      IDIST=1
C
  999 CONTINUE
      END
```

Algorithme vectoriel

```
      SUBROUTINE GVOBOX(X,PAR,IDIST,SNEXT,SAFE,B,NV)
C
C*************************************************************
C*                                                           *
C*     CALCUL VECTORIEL DU PAS SNEXT POUR DES BOX  CYBER 205 *
C*                                                           *
C*            J-L DEKEYSER              17.4.86  F.S.U       *
C*                                                           *
C*************************************************************
C
      COMMON/GCONST/PI,TWOPI ,PIBY2,DEGRAD,RADDEG,CLIGHT ,BIG,EMASS
C
      PARAMETER (MAXIN=300)
C
      DIMENSION PAR(*),X(*)
C
      BIT B(MAXIN),BEND(MAXIN),BS(MAXIN)
C
      REAL VTEMP,VX,VPAR,VSAF,VSMAX,VSMIN,VMAX,VMIN,VS,VBS
C
      DESCRIPTOR VTEMP,VX(6),VPAR(3),VSAF(3),VSMAX,VSMIN,
     +           VMAX,VMIN,VS,VBS
C
      ASSIGN VX(1),X(1;NV)
      ASSIGN VX(2),X(MAXIN+1;NV)
      ASSIGN VX(3),X(MAXIN*2+1;NV)
      ASSIGN VX(4),X(MAXIN*3+1;NV)
      ASSIGN VX(5),X(MAXIN*4+1;NV)
      ASSIGN VX(6),X(MAXIN*5+1;NV)
      ASSIGN VPAR(1),PAR(1;NV)
      ASSIGN VPAR(2),PAR(1+NV;NV)
      ASSIGN VPAR(3),PAR(1+2*NV;NV)
      ASSIGN VBS,.DYN.NV
      ASSIGN VS,.DYN.NV
      ASSIGN VTEMP,.DYN.NV
      ASSIGN VMAX,.DYN.NV
      ASSIGN VMIN,.DYN.NV
      DO 1 J=1,3
          ASSIGN VSAF(J),.DYN.NV
    1 CONTINUE
          ASSIGN VSMAX,.DYN.NV
          ASSIGN VSMIN,.DYN.NV
C           CONSTRUCTION OF PATTERNS
      SAFE=BIG
      DO 5 I=1,3
      VBS=VABS(VX(I);VBS)
      VSAF(I)=VBS-VPAR(I)
      BS(1;NV)=VSAF(I).GT.0.0.AND.B(1;NV)
      IF (Q8SCNT(BS(1;NV)).GT.0) THEN
         SAFM=Q8SMIN(VSAF(I),BS(1;NV))
         IF (SAFM.LT.SAFE) SAFE=SAFM
      END IF
    5 CONTINUE
C
```

```
      BEND(1;NV)=.NOT.((VX(1)*VX(4).GE.0.0.AND.VSAF(1).GT.0.0).OR.
     +            (VX(2)*VX(5).GE.0.0.AND.VSAF(2).GT.0.0).OR.
     +            (VX(3)*VX(6).GE.0.0.AND.VSAF(3).GT.0.0).OR.
     +            (VX(4).EQ.0.0.AND.VSAF(1).GE.0.0).OR.
     +            (VX(5).EQ.0.0.AND.VSAF(2).GE.0.0).OR.
     +            (VX(6).EQ.0.0.AND.VSAF(3).GE.0.0)).AND.B(1;NV)
C
      IF (Q8SCNT(BEND(1;NV)).LE.0) GO TO 999
      VMIN=0.0
      VMAX=BIG
      DO 20 J=1,3
         VSMAX=BIG
         VSMIN=0.0
         VBS=VABS(VX(J+3);VBS)
         VS=VABS(VX(J);VS)
         WHERE ((VSAF(J).LT.0.0).AND.BEND(1;NV).AND.VX(J+3).NE.0.)
               VSMAX=VPAR(J)/VBS-VX(J)/VX(J+3)
         END WHERE
         WHERE ((VSAF(J).GE.0.0).AND.BEND(1;NV))
               VSMIN=VSAF(J)/VBS
               VSMAX=(VPAR(J)+VS)/VBS
         END WHERE
         WHERE (VMIN.LT.VSMIN) VMIN=VSMIN
         WHERE (VMAX.GT.VSMAX) VMAX=VSMAX
20    CONTINUE
      BS(1;NV)=(BEND(1;NV).AND.VMAX.GE.VMIN.AND.VMIN.GT.0.0)
      IF (Q8SCNT(BS(1;NV)).LE.0) GO TO 999
      SMIN=Q8SMIN(VMIN,BS(1;NV))
      IF (IDIST.EQ.0.OR.SMIN.LT.SNEXT) SNEXT=SMIN
      IDIST=1
999   END
```

- Vectorisation de GSEAR

Algorithme séquentiel

```
C                 Start loop over contents in order specified by the
C                 JNEXT sequence.
C
   10 CONTINUE
C
      IF(IFLG.EQ.0.AND.INXT.EQ.NNXT)GO TO 300
      IF (INXT.EQ.NNXT) INXT=0
      INXT = INXT +1
      IF(INXT.EQ.-IFLG)GO TO 300
      J = JNXT +INXT
      IN = Q(J)
C
C                 Inspect current content
C
      CALL GMTST(XC,IN,JVO,IYES,NP,P)
C
      IF (IYES.NE.0) GO TO 200
      GO  TO  10
```

Algorithme vectoriel

```
C
      IF (JVOLD.NE.JVO) THEN
        CALL GNJTOI(JVO,JVECT)
      IFL=0
        IF (JVECT.LT.0) THEN
           JVECT=-JVECT+1
           IFL=1
        ELSE
           JVECT=JVECT+1
        END IF
```

```
        NIN2=2*NIN
        NIN6=6*NIN
        NIN5=5*NIN
        NIN15=15*NIN
        JVOLD=JVO
      END IF
C
      CALL GVTRAN(XC,V(JVECT+NIN2),V(JVECT+NIN6),XT,NIN,IFL)
      IF (ISEARC.NE.2) GO TO 80
      CALL GVINME(XT,LV(JVECT),V(JVECT+NIN15),IYES,NIN)
      IF (IYES.NE.0) THEN
        IN=IYES
        NP=LV(JVECT+NIN+IN-1)
        P(1;NP)=Q8VGATHP(V(JVECT+NIN15+IN-1;NIN*NP),NIN,NP;P(1;NP))
C       DO 60 I=1,NP
C60     P(I)=V(JVECT+NIN15+(I-1)*NIN-1+IN)
        IYES=1
      ELSE
        GO TO 300
      END IF
      GO TO 200
```

- Vectorisation de GINME pour des boites

Algorithme séquentiel

```
      IF(ISH.NE.1) GO TO 40
C
C             RECTILINEAR BOX.
C
      DX      =P(1)
      IF (X(1).LT. - DX.OR.X(1).GT.DX) GO TO 999
      DY      =P(2)
      IF (X(2).LT. - DY.OR.X(2).GT.DY) GO TO 999
      DZ      =P(3)
      IF (X(3).LT. - DZ.OR.X(3).GT.DZ) GO TO 999
      IYES=1
      GO TO 999
C
   40 CONTINUE
```

Algorithme vectoriel

```
      SUBROUTINE GVINBO(X,PAR,IYES,B,NV)
C
C                 CYBER 205
C                          J-L DEKEYSER   F.S.U.
C
C
      COMMON/GCONST/PI,TWOPI ,PIBY2,DEGRAD,RADDEG,CLIGHT ,BIG,EMASS
C
      PARAMETER (MAXIN=300)
C
      DIMENSION  PAR(*),X(*)
C
      BIT B(MAXIN)
C
      INTEGER IYV,IVP
C
      REAL VTEMP,VX,VPAR
C
      DESCRIPTOR VTEMP,VX(3),VPAR(3),IYV,IVP
C
      ASSIGN IVP,.DYN.NV
      ASSIGN IYV,.DYN.NV
      ASSIGN VTEMP,.DYN.NV
      ASSIGN VX(1),X(1;NV)
      ASSIGN VX(2),X(MAXIN+1;NV)
      ASSIGN VX(3),X(MAXIN*2+1;NV)
      ASSIGN VPAR(1),PAR(1;NV)
      ASSIGN VPAR(2),PAR(1+NV;NV)
      ASSIGN VPAR(3),PAR(1+2*NV;NV)
```

```
      IYV=0
      WHERE(VX(1).LE.VPAR(1).AND.VX(1).GE.-VPAR(1).AND.
     +      VX(2).LE.VPAR(2).AND.VX(2).GE.-VPAR(2).AND.
     +      VX(3).LE.VPAR(3).AND.VX(3).GE.-VPAR(3).AND.B(1;NV))IYV=1
      IVP=1
      IYES=Q8SEQ(IVP,IYV)
      IF (IYES.EQ.NV) THEN
          IYES=0
        ELSE
          IYES=IYES+1
      END IF
  999 END
```

- Vectorisation de GINME pour des tubes

Algorithme séquentiel

```
      IF(ISH.NE.5) GO TO 20
C
C          TUBE AXIS = Z AXIS.
C
      DZ     = P(3)
      IF (X(3).GT.DZ) GO TO 999
      IF (X(3).LT. - DZ) GO TO 999
      R2=X(1)*X(1)+X(2)*X(2)
      IF(R2.LT.P(1)*P(1)) GO TO 999
      IF(R2.GT.P(2)*P(2)) GO TO 999
      IYES=1
      GO TO 999
C
   20 CONTINUE
```

Algorithme vectoriel

```
      SUBROUTINE GVINTU(X,PAR,IYES,B,NV)
C
C                CYBER 205
C                         J-L DEKEYSER   F.S.U.
C
C
      COMMON/GCONST/PI,TWOPI ,PIBY2,DEGRAD,RADDEG,CLIGHT ,BIG,EMASS
C
      PARAMETER (MAXIN=300)
C
      DIMENSION  PAR(*),X(*)
C
      BIT B(MAXIN)
C
      INTEGER IYV,IVP
C
      REAL VTEMP,VX,VPAR,R2
C
      DESCRIPTOR VTEMP,VX(3),VPAR(3),IYV,IVP,R2
C
      ASSIGN IVP,.DYN.NV
      ASSIGN IYV,.DYN.NV
      ASSIGN VTEMP,.DYN.NV
      ASSIGN VX(1),X(1;NV)
      ASSIGN VX(2),X(MAXIN+1;NV)
      ASSIGN VX(3),X(MAXIN*2+1;NV)
      ASSIGN VPAR(1),PAR(1;NV)
      ASSIGN VPAR(2),PAR(1+NV;NV)
      ASSIGN VPAR(3),PAR(1+2*NV;NV)
      ASSIGN R2,.DYN.NV
```

```
      R2=VX(1)*VX(1)+VX(2)*VX(2)
      IYV=0
      WHERE(R2.GE.VPAR(1)*VPAR(1).AND.
     +      R2.LE.VPAR(2)*VPAR(2).AND.
     +      VX(3).LE.VPAR(3).AND.VX(3).GE.-VPAR(3).AND.B(1;NV))IYV=1
      IVP=1
      IYES=Q8SEQ(IVP,IYV)
      IF (IYES.EQ.NV) THEN
          IYES=0
        ELSE
          IYES=IYES+1
      END IF
999   END
```

- Vectorisation de GINME pour des sections de tubes

Algorithme séquentiel

```
      IF(ISH.NE.6) GO TO 30
C
C          TUBE SEGMENT.
C
      DZ      = P(3)
      IF (X(3).GT.DZ) GO TO 999
      IF (X(3).LT. - DZ) GO TO 999
      R2=X(1)*X(1)+X(2)*X(2)
      IF(R2.LT.P(1)*P(1)) GO TO 999
      IF(R2.GT.P(2)*P(2)) GO TO 999
      IYES=1
      IF (R2.LT.1.0E - 6)                          GO TO 999
      IYES=0
      PHI     = ATAN2(X(2),X(1)) * RADDEG
      DP=P(5)-P(4)
      IF(DP.LT.0.0) DP=DP+360.0
      DDP=PHI-P(4)
      IF(DDP.LT.0.0) DDP=DDP+360.0
      IF(DDP.LT.0.0) DDP=DDP+360.0
      IF(DDP.GT.360.0) DDP=DDP-360.0
      IF(DDP.GT.DP) GO TO 999
      IYES=1
      GO TO 999
C
   30 CONTINUE
```

Algorithme vectoriel

```
      SUBROUTINE GVINTX(X,PAR,IYES,B,NV)
C
C                CYBER 205
C                          J-L DEKEYSER   F.S.U.
C
C
      COMMON/GCONST/PI,TWOPI ,PIBY2,DEGRAD,RADDEG,CLIGHT ,BIG,EMASS
C
      PARAMETER (MAXIN=300)
C
      DIMENSION  PAR(*),X(*)
C
      BIT B(MAXIN)
C
      INTEGER IYV,IVP
C
      REAL VTEMP,VX,VPAR,R2,PHI,DP,DDP
C
      DESCRIPTOR VTEMP,VX(3),VPAR(5),IYV,IVP,R2,PHI,DP,DDP
C
```

```
ASSIGN IVP,.DYN.NV
ASSIGN IYV,.DYN.NV
ASSIGN VTEMP,.DYN.NV
ASSIGN VX(1),X(1;NV)
ASSIGN VX(2),X(MAXIN+1;NV)
ASSIGN VX(3),X(MAXIN*2+1;NV)
ASSIGN VPAR(1),PAR(1;NV)
ASSIGN VPAR(2),PAR(1+NV;NV)
ASSIGN VPAR(3),PAR(1+2*NV;NV)
ASSIGN VPAR(4),PAR(1+3*NV;NV)
ASSIGN VPAR(5),PAR(1+4*NV;NV)
ASSIGN PHI,.DYN.NV
ASSIGN DDP,.DYN.NV
ASSIGN DP,.DYN.NV
R2=VX(1)*VX(1)+VX(2)*VX(2)
PHI=VATAN2(VX(2),VX(1);PHI)
DP=VPAR(5)-VPAR(4)
WHERE (DP.LT.0.0) DP=DP+360.0
DDP=PHI*RADDEG-VPAR(4)
WHERE (DDP.LT.0.0) DDP=DDP+360.0
WHERE (DDP.LT.0.0) DDP=DDP+360.0
WHERE (DDP.GT.360.0) DDP=DDP-360.0
IYV=0
WHERE(R2.GE.VPAR(1)*VPAR(1).AND.R2.GE.1.0E-6.AND.
+       R2.LE.VPAR(2)*VPAR(2).AND.DDP.LE.DP.AND.
+       VX(3).LE.VPAR(3).AND.VX(3).GE.-VPAR(3).AND.B(1;NV))IYV=1
 IVP=1
 IYES=Q8SEQ(IVP,IYV)
 IF (IYES.EQ.NV) THEN
     IYES=0
   ELSE
     IYES=IYES+1
 END IF
```

APPENDICE D

SYNCHRONISATION INDIRECTE EN ADA

Publié dans "Journal Pascal Ada and Modula 2" de novembre 1986

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

SYNCHRONIZATION BY INDIRECT ACTION MECHANISM IN ADA

J – L.Dekeyser[1)]

Geneva, 2 September 1986

Abstract: In Ada the synchronization of tasks is based on the mechanism of the "rendez – vous". In the present paper an Ada implementation of synchronization by events and semaphores is presented. With these mechanisms, the communication between two tasks can be performed without knowing the name of the other task. One task waits for one or more signals and may stop if they do not arrive, the other task sends one or more signals and does not have to stop. The two packages use the rendez – vous technique and communicate via a data structure. Events and semaphores are implemented as two private type objects and they respect the same rules as other Ada objects.

1) CERN,Division DD/EE,Geneve,Suisse

1. Introduction

The rendez-vous in Ada [3] is characterized by the following rules :

- If the calling task issues an entry call statement before a corresponding accept statement, its execution is suspended.

- If a task reaches an accept statement prior to any call, execution is suspended until such a call is received.

- The calling task has to know the name of the entry and the name of the task before the execution of an entry call statement. The server task on the other hand does not know the name of the caller; in this sense the rendez-vous is asymmetric.

Therefore it would be advantageous to synchronize task by an indirect action mechanism. In this scheme the task which releases a signal can continue its execution even if the task which waits for the signal has not yet reached the suitable statement. Another advantage is that the calling task does not have to know the name of the other task but only one or more intermediate objects common to several tasks.

2. NOTION OF EVENTS.

2.1 Definition of the synchronization by events.

An event [4] is defined as an object of the language and it can be manipulated by specific operations. It can be waited for by one or more tasks, or it can be released. Its visibility respects the rules of the language.

An event can be released with or without memorization.

A memorized event is represented by a boolean variable. At a given moment its value is TRUE if the event has been released. The procedure WAIT permits to wait one or more events are released. When a task reaches a "wait" call statement before the release of one of the events present in its parameters list, its execution is suspended. A procedure can reset the event to FALSE.

If it is not memorized, only the tasks waiting for the event at a given moment will be concerned.

2.1.1 Example :

Consider the two following procedures :

```
RELEASE(EVT) -- releases the event EVT

WAIT(EVT1,EVT2,....EVTn)
                    -- wait until all events
                    -- EVT1,EVT2,..EVTn are released
WAIT(FOR_AT_LEAST,EVT1,EVT2,....EVTn) --
                    -- wait until FOR_AT_LEAST events among
                    -- EVT1,EVT2,..EVTn are released
```

and the 3 tasks t1,t2 and t3. The following three examples show the development in 3 different execution patterns. ("." = active and "*" = inactive.)

- *Example 1:* An event evt1 is released before the "wait" call statement is reached and none of the tasks is stopped. (events are memorized, the time arrow runs from top to bottom)

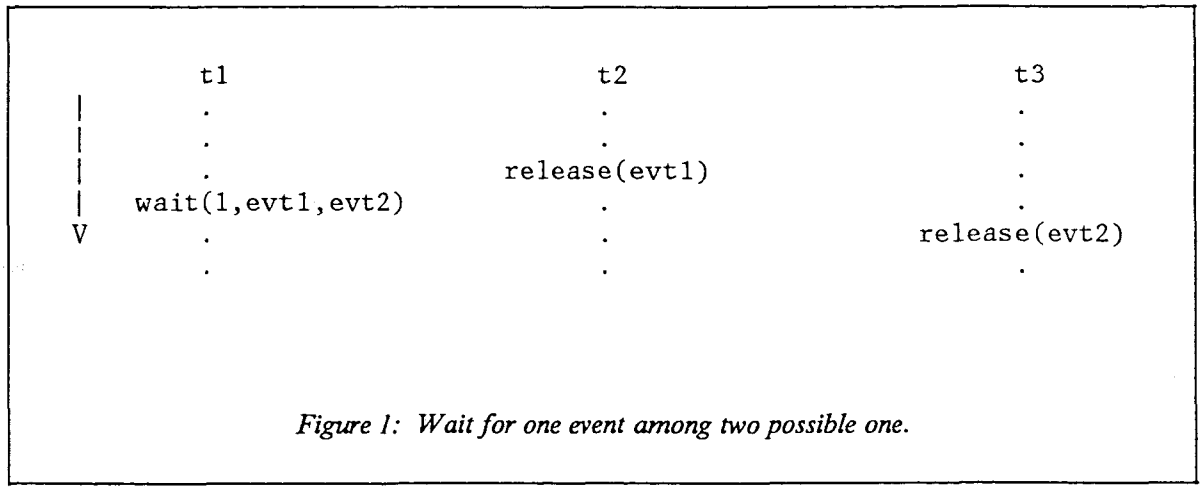


Figure 1: Wait for one event among two possible one.

- *Example 2:* Task t1 is stopped until one of the two events is released.

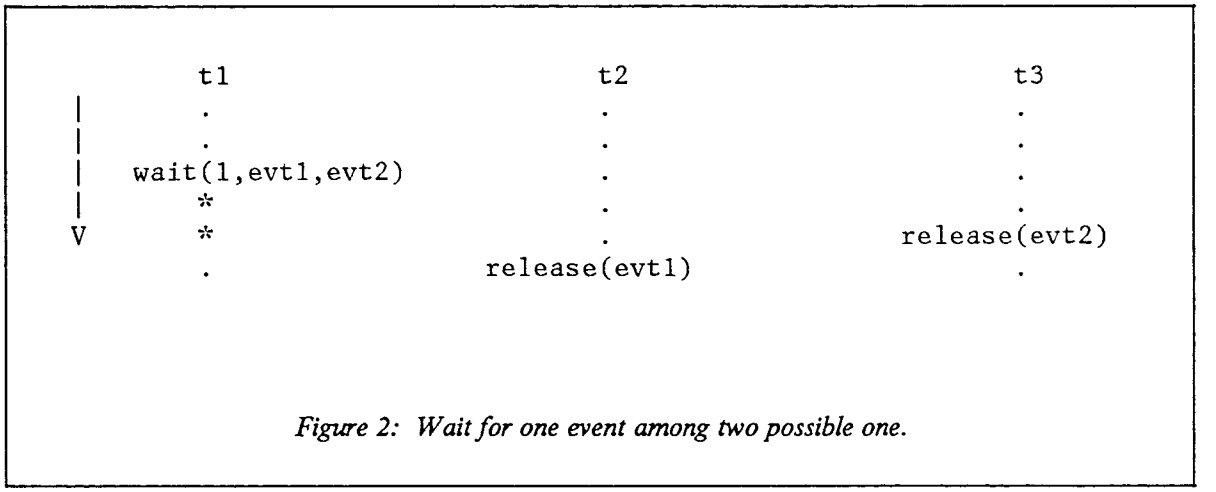


Figure 2: Wait for one event among two possible one.

- *Example 3:* Task t1 is stopped until the two events are released.

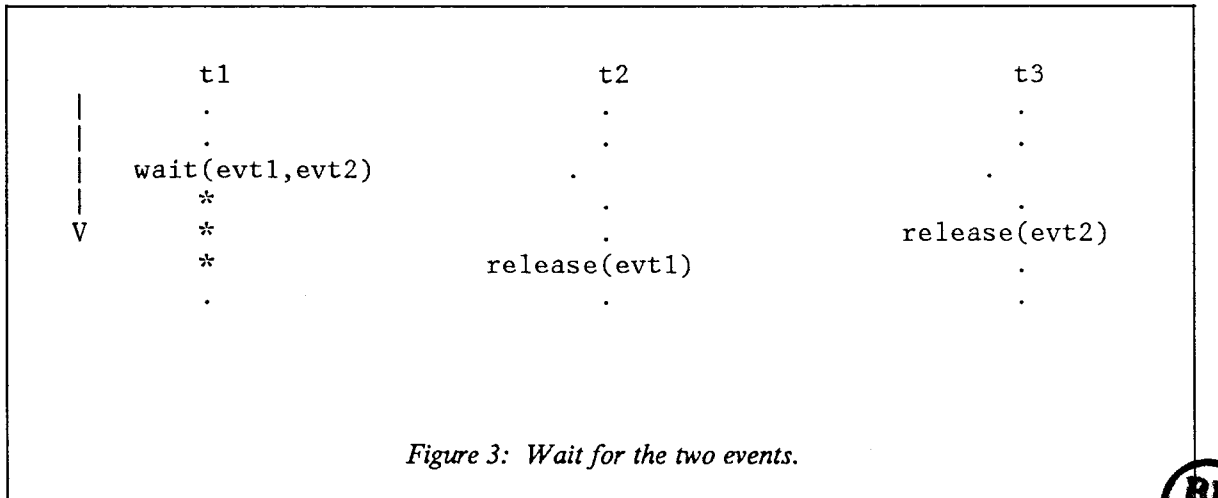


Figure 3: Wait for the two events.

BU LILLE

2.2 Implementation of events by means of the Ada rendez−vous.

In order to realize blocking and release of tasks according to the rules described above, a data structure as describes below is defined :

Let us first consider the following program :

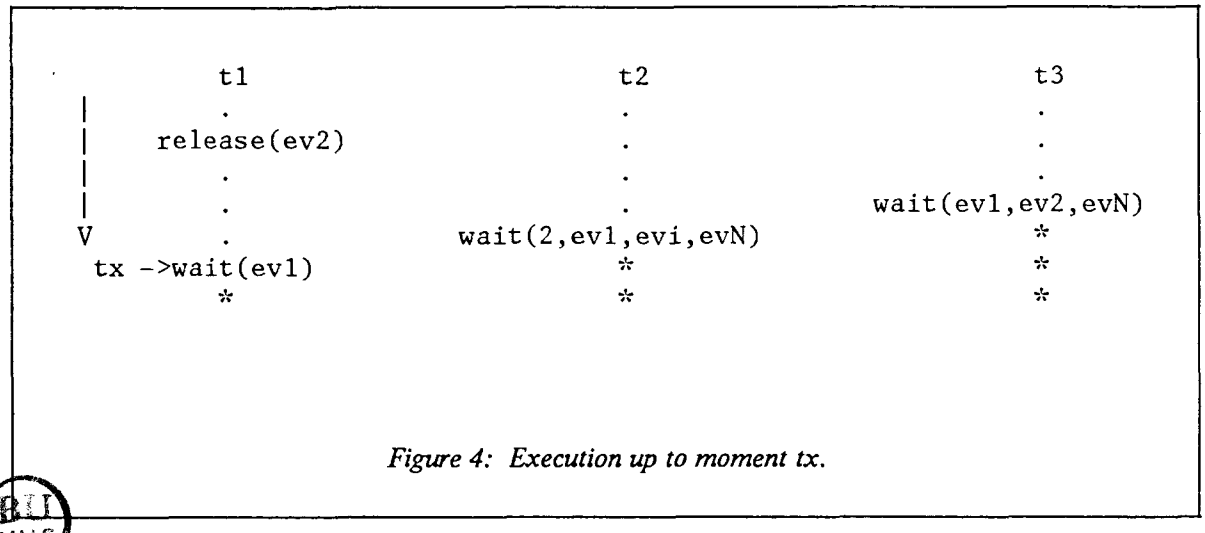


Figure 4: Execution up to moment tx.

BU LILLE

- *Task t1:* releases event ev2 and waits for event ev1.
- *Task t2:* waits for 2 events among the events ev1, evi and evN.
- *Task t3:* waits for all the events ev1, ev2 and evN. The event ev2 has already been released.

The associated data structure at moment tx is :

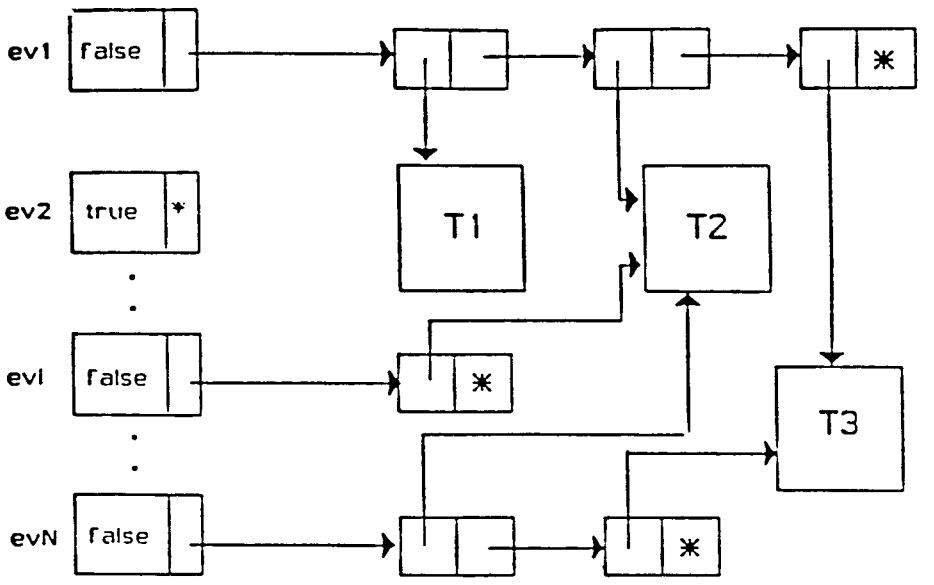


Figure 5: The data structure at moment tx.

- *Task t1:* has created task T1.
- *Task t2:* has created task T2.
- *Task t3:* has created task T3.

Each event is associated with boolean variable, whose value is TRUE if the event is released and FALSE otherwise, and with a list of Ada access types designating a task object.

A designated task has 3 entries :

- SET_WAIT(WAKE_BEFORE,WAKE_AFTER:integer) : This entry is called once by procedure WAIT, it permits to initialize the task at its creation. WAKE_BEFORE and WAKE_AFTER specify the number of "accept" of WAKE_UP before and after the "accept" of GO. These values depend on the value of FOR_AT_LEAST in the parameter list of the WAIT call and on the status of the events in the parameter list at the moment of the call.

  In the previous example :
  The call WAIT(ev1,ev2,evN) for task t3 produces an entry call SET_WAIT(2,0). Task t1 has already released event ev2.
  The call WAIT(2,ev1,evi,evN) for task t2 produces an entry call SET_WAIT(2,1).

- WAKE_UP : This entry is called by procedure RELEASE(evt) if the task is designated by an element of the list of access types designating a task object associated at event evt.

- GO : This entry is called by procedure WAIT which created this task. The accept statement will be executed only after ***WAKE_BEFORE*** entry calls.

2.2.1 Algorithms.

Procedure WAIT creates, if at least one event is not released, a task of type T. This task is stored in the data structure in such a way that it can be referenced from each event of the parameter list that has not yet been released when the WAIT call statement is executed.
Procedure WAIT calls the entry GO of this task. Yet the latter accepts the entry only after having accepted ***WAKE_BEFORE*** WAKE_UP entries called by the procedure RELEASE. ***WAKE_AFTER*** events have not been released yet when the entry GO is accepted.

Procedure RELEASE calls for each task accessible from the referenced event the WAKE_UP entry. After that, it restores the space used by all the access types associated at this event. The links between the event and the tasks are definitively cut.

Procedure RESET initializes the boolean variable associated to the event to FALSE if its list of access types is empty.

2.3 Realization in Ada

Listing of the package EVENT with an example of its use.

```
package EVENTS is
    type EVENT is limited private;

    procedure RELEASE(EVT: in out EVENT);

    procedure RESET(EVT: in out EVENT);

    --
    -- wait among 1,2,3.....9,10 events
    --
    procedure WAIT(FOR_AT_LEAST:in INTEGER:=1;
    EVT1:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=2;
    EVT1,EVT2:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=3;
    EVT1,EVT2,EVT3:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=4;
    EVT1,EVT2,EVT3,EVT4:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=5;
    EVT1,EVT2,EVT3,EVT4,EVT5:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=6;
    EVT1,EVT2,EVT3,EVT4,EVT5,EVT6:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=7;
    EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=8;
    EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8:in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=9;
    EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9 :in out EVENT);

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=10;
    EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0:in out EVENT);
    private
    --
    -- definition of type task T
    --

    task type T is
        entry SET_WAIT(WAKE_BEFORE,WAKE_AFTER:in INTEGER);
        entry WAKE_UP;
        entry GO;
    end T;
    --
```

```
    -- access type designating a task of type T
    --
    type PT is access T;
    type LINK;
    type LINK_PTR is access LINK;
    type LINK is record
        TASK_PTR:PT:=null;              -- access task
        NEXT:LINK_PTR:=null;             -- access link
    end record;
    type EVENT is record
        GO:BOOLEAN:=FALSE;       -- with memorization
        ROOT:LINK_PTR:=null;
    end record;
end EVENTS;
with UNCHECKED_DEALLOCATION;

package body EVENTS is
    --
    -- semaphore of mutual exclusion for the data structure
    --

task SEM is
    entry P;
    entry V;
end SEM;

task body SEM is

begin
    loop
        accept P;
        accept V;
    end loop;
end SEM;

task body T is
WAKE_UP_BEFORE,WAKE_UP_AFTER:INTEGER;
I :INTEGER;

begin
    --
    -- initialization of the task T
    --
    -- number of wake_up before accepting the entry debloc
    -- number of wake_up after accepting the entry debloc
    --
    accept SET_WAIT(WAKE_BEFORE,WAKE_AFTER:in INTEGER) do
        WAKE_UP_BEFORE:=WAKE_BEFORE;
        WAKE_UP_AFTER:=WAKE_AFTER;
    end;
    for I in 1..WAKE_UP_BEFORE loop
        accept WAKE_UP;
    end loop;
    accept GO;
    if WAKE_UP_AFTER>0
      then
        for I in 1..WAKE_UP_AFTER loop
```

```
            accept WAKE_UP;
        end loop;
    end if;
end T;

procedure FREE is
new UNCHECKED_DEALLOCATION(LINK,LINK_PTR);

procedure RELEASE(EVT:in out EVENT) is
PC, PTC:LINK_PTR:=null;

begin
    --
    -- start of critical section
    --
    SEM.P;
    --
    -- memorization of the release
    --
    EVT.GO:=TRUE;
    PTC:=EVT.ROOT;
    --
    -- scan the list of tasks associated at this event
    --
    while PTC/=null loop
        if PTC.TASK_PTR/=null
          then
            PTC.TASK_PTR.all.WAKE_UP;
        end if;
        PC:=PTC;
        PTC:=PTC.NEXT;
        FREE(PC);
    end loop;
    EVT.ROOT:=null;
    SEM.V;
    --
    -- end of critical section
    --
end RELEASE;

procedure RESET(EVT:in out EVENT) is

begin
    --
    -- start of critical section
    --
    SEM.P;
    if EVT.ROOT=null
      then
        EVT.GO:=FALSE;
    end if;
    SEM.V;
    --
    -- end of critical section
    --
end RESET;
```

```
procedure WAIT(FOR_AT_LEAST:in INTEGER:=10;
EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9 ,EVT0:in out EVENT )is
PTC:LINK_PTR:=null;
NOT_RELEASED_EVENT,NB_WAKE_UP:INTEGER:=0;
PTACH:PT;

procedure TRAITER(EVT:in out EVENT;
NOT_RELEASED_EVENT:in out INTEGER)is
PTC:LINK_PTR:=null;

begin
    if not EVT.GO
      then
        --
        -- insert at head of list
        --
        PTC:=new LINK;
        PTC.NEXT:=EVT.ROOT;
        EVT.ROOT:=PTC;
        NOT_RELEASED_EVENT:=NOT_RELEASED_EVENT+1;
    end if;
end TRAITER;

procedure MAJEVT (EVT:in out EVENT)is

begin
    if not EVT.GO
      then
        EVT.ROOT.all.TASK_PTR:=PTACH;
    end if;
end MAJEVT;

begin
    --
    -- start of critical section
    --
    SEM.P;
    TRAITER(EVT1,NOT_RELEASED_EVENT);
    TRAITER(EVT2,NOT_RELEASED_EVENT);
    TRAITER(EVT3,NOT_RELEASED_EVENT);
    TRAITER(EVT4,NOT_RELEASED_EVENT);
    TRAITER(EVT5,NOT_RELEASED_EVENT);
    TRAITER(EVT6,NOT_RELEASED_EVENT);
    TRAITER(EVT7,NOT_RELEASED_EVENT);
    TRAITER(EVT8,NOT_RELEASED_EVENT);
    TRAITER(EVT9,NOT_RELEASED_EVENT);
    TRAITER(EVT0,NOT_RELEASED_EVENT);
    NB_WAKE_UP:=FOR_AT_LEAST-(10-NOT_RELEASED_EVENT);
    if NB_WAKE_UP>0
      then
        --
        -- creation of the task T
        --
        PTACH:=new T;
        PTACH.all.SET_WAIT(NB_WAKE_UP,NOT_RELEASED_EVENT-NB_WAKE_UP);
        --
        -- update the data structure
```

```
            --
            MAJEVT(EVT1);
            MAJEVT(EVT2);
            MAJEVT(EVT3);
            MAJEVT(EVT4);
            MAJEVT(EVT5);
            MAJEVT(EVT6);
            MAJEVT(EVT7);
            MAJEVT(EVT8);
            MAJEVT(EVT9);
            MAJEVT(EVT0);
        end if;
        SEM.V;
        --
        -- end of critical section
        --
        -- call the entry  debloc of the new task
        --
        if NB_WAKE_UP>0 then
            PTACH.all.GO;
        end if;
    end WAIT;

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=1;
    EVT1:in out EVENT)is
    EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0:EVENT:=(TRUE,null);
    TP:INTEGER;

    begin
        TP:=FOR_AT_LEAST+9;
        WAIT(TP,EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0);
    end WAIT;

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=2;
    EVT1,EVT2:in out EVENT)is
    EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0:EVENT:=(TRUE,null);
    TP:INTEGER;

    begin
        TP:=FOR_AT_LEAST+8;
        WAIT(TP,EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0);
    end WAIT;

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=3;
    EVT1,EVT2,EVT3:in out EVENT)is
    EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0:EVENT:=(TRUE,null);
    TP:INTEGER;

    begin
        TP:=FOR_AT_LEAST+7;
        WAIT(TP,EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0);
    end WAIT;

. . . . . .

    procedure WAIT(FOR_AT_LEAST:in INTEGER:=9;
    EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9:in out EVENT)is
```

```
EVT0:EVENT:=(TRUE,null);
TP:INTEGER;

begin
    TP:=FOR_AT_LEAST+1;
    WAIT(TP,EVT1,EVT2,EVT3,EVT4,EVT5,EVT6,EVT7,EVT8,EVT9,EVT0);
end WAIT;
end EVENTS;
```

2.3.1 Example of use.

Control of a boiler by a multiprocessor.

There are 5 tasks :

- *TEMP:* supervises the measurement of the temperature and releases event HEAT when the temperature is more than 500°C.
- *PRESS:* supervises the measurement of the pressure and releases event PRESSURE when the pressure is more than 1000 pascal.
- *ALARM:* set off an alarm when the temperature is more than 500°C or when the pressure is more than 1000 pascal.
- *FIRE_MAN:* calls the fire−men when the temperature is more than 500°C.
- *EVACUATION:* rings bell when the temperature is more than 500°C and the pressure more than 1000 pascal.

```
with EVENTS,TEXT_IO;
use TEXT_IO,EVENTS;
procedure EXEVTS is
-- pragma TIME_SLICE(0.01);              VAX pragma


ILAST:INTEGER:=123;
HEAT,PRESSURE:EVENT;
function RDM(N :in INTEGER) return INTEGER is
-- a little random number generator

begin
    ILAST:=(ILAST*13456-34456) mod N;
    return ILAST;
end RDM;
--
-- mutual exclusion for the terminal
--

task MUTEX is
entry P;
entry V;
end MUTEX;

task body MUTEX is
```

```
begin
    loop
        accept P;
        accept V;
    end loop;
end MUTEX;
--
-- measurement of temperature
--

task TEMP ;
--
-- measurement of  pressure
--

task PRESS;
--
-- tasks of urgency
--

task ALARM;

task FIRE_MAN;

task EVACUATION;
--
-- definition of tasks
--

task body TEMP is
TEMP_NOW:INTEGER;
TEMP_MAX:INTEGER:=500;

begin
    loop
        TEMP_NOW:=RDM(800);
        MUTEX.P;
        PUT("temperature = ");
        PUT(TEMP_NOW);
        NEW_LINE;
        MUTEX.V;
        if TEMP_NOW>TEMP_MAX
          then
            RELEASE(HEAT);
            exit;
        end if;
    end loop;
end TEMP;

task body PRESS is
PRESS_NOW:INTEGER;
PRESS_MAX:INTEGER:=1000;

begin
    loop
        PRESS_NOW:=RDM(1200);
```

```
            MUTEX.P;
            PUT("pressure    = ");
            PUT(PRESS_NOW);
            PUT_LINE("");
            MUTEX.V;
            if PRESS_NOW>PRESS_MAX
              then
                RELEASE(PRESSURE);
                exit;
            end if;
        end loop;
    end PRESS;

    task body ALARM is

    begin
        WAIT(1,HEAT,PRESSURE);
        MUTEX.P;
        PUT_LINE("********** alarm **********");
        MUTEX.V;
    end ALARM;

    task body FIRE_MAN is

    begin
        WAIT(HEAT);
        MUTEX.P;
        PUT_LINE("********** fire_man **********");
        MUTEX.V;
    end FIRE_MAN;

    task body EVACUATION is

    begin
        WAIT(HEAT,PRESSURE);
        MUTEX.P;
        PUT_LINE("********** evacuation **********");
        MUTEX.V;
    end EVACUATION;

    begin
        null;
    end EXEVTS;
```

The result is

```
        pressure    =          632
        temperature =          136
        pressure    =          360
        temperature =          104
tx->    pressure    =          568
        temperature =          552              {release HEAT}
        pressure    =           56
        pressure    =          280
        pressure    =           24
        ********** alarm **********
        ********** fire_man **********
        pressure    =          488
        pressure    =          472
        pressure    =         1176              {release PRESSURE}
        ********** evacuation **********
```

This execution was realized on a monoprocessor VAX 8600.

The associated data structure at moment tx is :

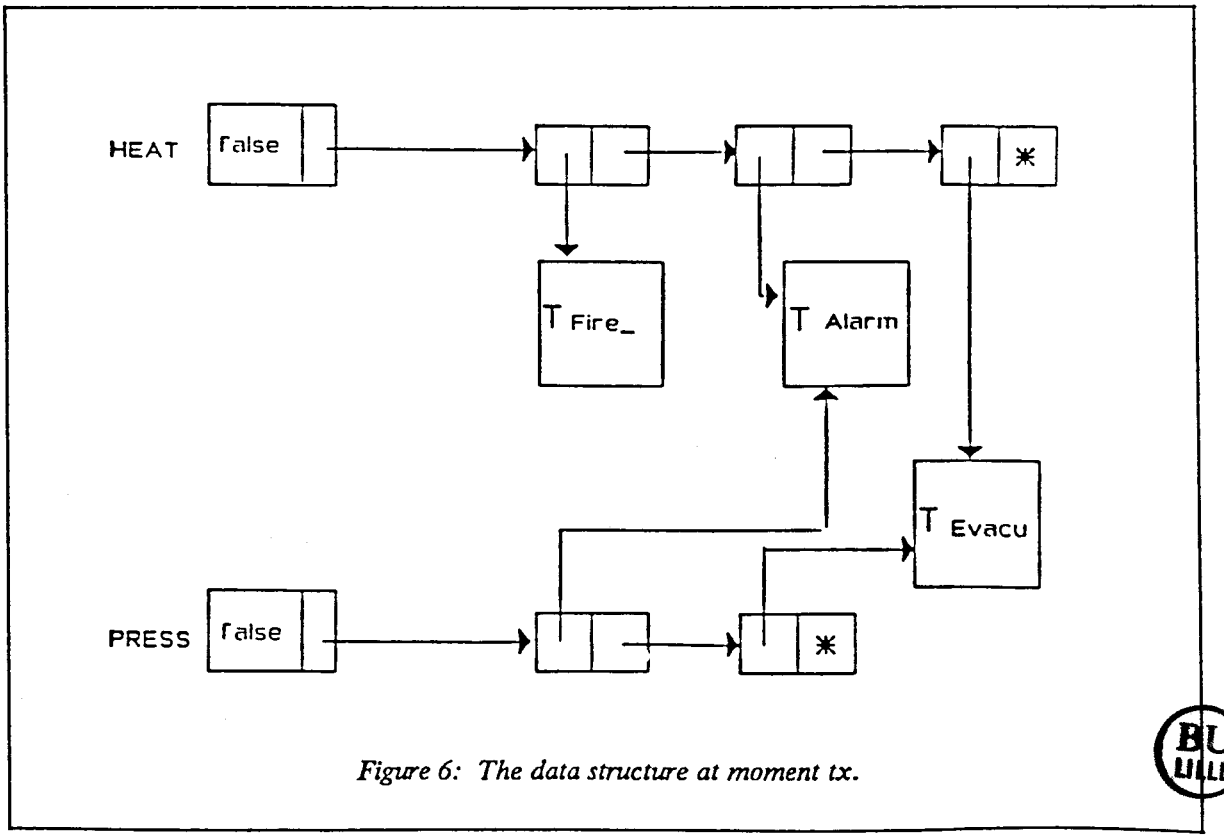


Figure 6: The data structure at moment tx.

3. NOTION OF SEMAPHORES.

3.1 Definition of synchronization by semaphores.

3.1.1 Definitions.

A semaphore [1] s is composed of an integer variable val(s) and a queue of tasks f(s). The variable val(s) can take positive, negative or zero values. At its creation, the value of val(s) is positive or nil, the queue f(s) is empty.

Two primitives P and V are defined as follows

```
P(s) :
        val(s): = val(s) − 1
        if val(s) < 0 then
                                    the task is stopped and put in the
                                    waiting queue f(s)
        end if

V(s) :
        val(s): = val(s) + 1
        if val(s) < = 0 then
                                    one task is removed from the waiting
                                    queue f(s) and is reactived
        end if
```

A semaphore can only be altered by the operations P and V which are indivisible. Only one task at a time can execute a P or V operation on a given semaphore. Therefore, if val(s) = 1 and two tasks try to execute a P(s) simultaneously, only one task will be permitted to continue. The other task will be blocked and put on the waiting queue f(s).

3.1.2 Semaphore of mutual exclusion.

Resources can be shared, although generally they can only be used by one task at a time. This type of resource is called a critical resource. If several tasks wish to share the use of a critical resource, they must synchronize their operations. If one task is using the critical resource, then all other tasks that want the resource must wait until the resource is freed. The regions which access critical resources are called critical sections and must be mutually exclusive.

Using a semaphore s, tasks can implement mutual exclusion by bracketing their critical sections by P(s) and V(s). If val(s) = 1 then neither task is executing its critical section. If val(s) = 0 then one task is executing its critical section. If val(s) = − 1 then one task is executing its critical section and one task is on the waiting queue f(s), waiting to be allowed to enter the critical section.

Example : semaphore of mutual exclusion :mutex (initialized to 1)

```
task 1                          task 2
  begin                           begin
  A1,A2...Ai                      B1,B2...Bj
  P(mutex)                        P(mutex)
  critical section 1              critical section 2
  V(mutex)                        V(mutex)
  An...Ak                         B1...Bh
  end 1                           end 2
```

3.1.3 Synchronization by semaphores.

Semaphores can be used to resolve general synchronization problems : an activation signal is sent using primitive V, it is waited for by primitive P. Semaphore s is private for a task t if only the latter can execute calls to the primitive P(s), while all other tasks can only execute calls to the primitive V(s).

A task waiting for emission of a signal can block itself with primitive P behind its private semaphore s initialized to 0. The waking up of this task is obtained by the execution of primitive V with the same private semaphore s by another task.

Example : private semaphore : signal (initialized to 0)

```
task 1                          task 2
  begin                           begin
  A1,A2,...Ai                     B1,B2,...Bj
  V(signal)                       P(signal)
  .........                       .......
  end 1                           end 2
```

2 possible execution pattern arise :

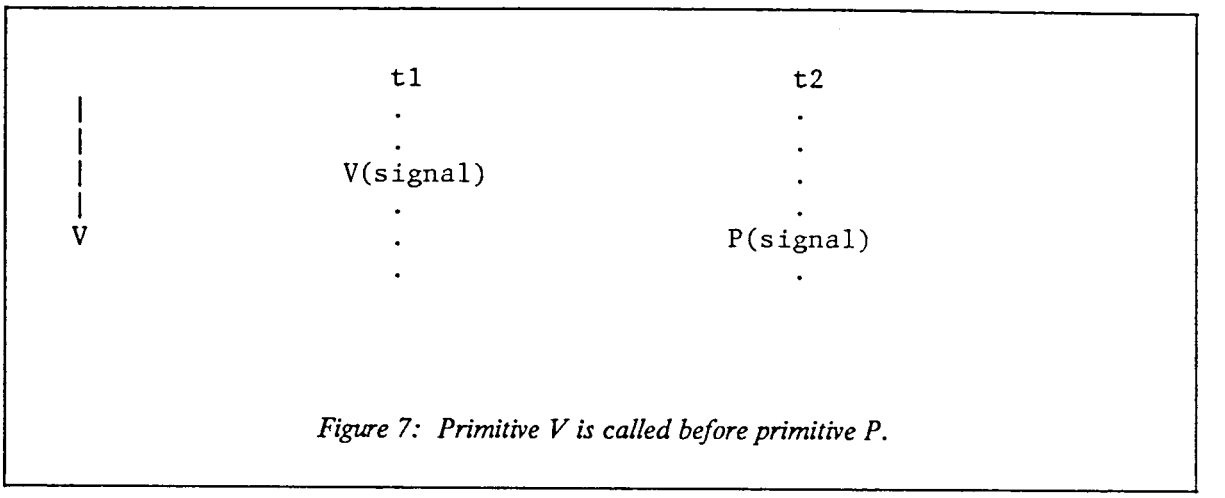


Figure 7: Primitive V is called before primitive P.

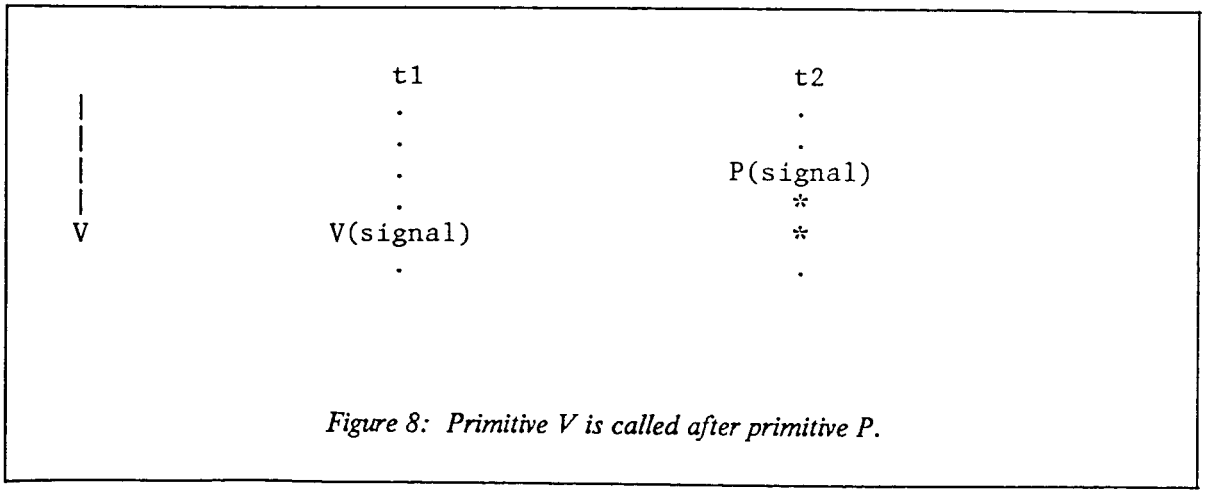


Figure 8: Primitive V is called after primitive P.

BU LILLE

3.2 Implementation of semaphores in Ada.

Listing of the package SEMAPH for using semaphores.

```
package SEMAPH is
    type PAIR_TASK is limited private;

    type PTFIFO is access PAIR_TASK;
    type SEMAPHORE is record
        VAL:INTEGER:=0;
        LIST:PTFIFO:=null;
    end record;

    procedure P(SEM:in out SEMAPHORE);

    procedure V(SEM:in out SEMAPHORE);
    private
    --
    -- mutual exclusion for one semaphore
    -- we can not execute at the same time P(s) and V(s)
    --
    task type MUTEX is
        entry P;
        entry V;
    end MUTEX;

    --
    -- task queue for one semaphore (FIFO)
    --
    task type FIFO is
        entry P_OUT;
        entry P_IN;
    end FIFO;
    type PAIR_TASK is record
        FIF:FIFO;
        MUTX:MUTEX;     -- P and V are indivisible
    end record;
end SEMAPH;

package body SEMAPH is

    task body MUTEX is

    begin
        loop
            accept P;
            accept V;
        end loop;
    end MUTEX;

    task body FIFO is

    begin
```

```
        loop
            accept P_OUT;
            accept P_IN;
        end loop;
    end FIFO;

    procedure P(SEM:in out SEMAPHORE) is

    begin
        if SEM.LIST=null
          then
            SEM.LIST:=new PAIR_TASK;
        end if;
        SEM.LIST.MUTX.P;
        SEM.VAL:=SEM.VAL-1;
        if SEM.VAL<0
          then
            SEM.LIST.MUTX.V;
            --
            -- stopped P out of critical section
            --
            SEM.LIST.FIF.P_IN;
          else
            SEM.LIST.MUTX.V;
        end if;
    end P;

    procedure V(SEM:in out SEMAPHORE)is

    begin
        if SEM.LIST=null
          then
            SEM.LIST:=new PAIR_TASK;
        end if;
        SEM.LIST.MUTX.P;
        SEM.VAL:=SEM.VAL+1;
        if SEM.VAL<=0
          then
            SEM.LIST.FIF.P_OUT;
        end if;
        SEM.LIST.MUTX.V;
    end V;
end SEMAPH;
```

Example of use : producer/consumer

Let PRODUCER be a task which transmits blocks of data to another task called CONSUMER. Between PRODUCER and CONSUMER there is a queue of full buffers containing messages. When PRODUCER wishes to send a message, it adds a buffer to the end of the queue. CONSUMER receives a message by removing the buffer which is on the front of the queue. PRODUCER can transmit only as long as empty buffers are available. Similarly, CONSUMER can receive only when the queue is nonempty. AVAIL is a semaphore whose value is the number of empty buffers in the buffer pool. FULL is a semaphore whose value is the number of full buffers that have been sent to the consumer.

```
with TEXT_IO,SEMAPH;
use SEMAPH,TEXT_IO;

procedure ESSEM is
-- pragma TIME_SLICE(0.001);          VAX pragma
ILAST:INTEGER:=123;
--
-- size of buffer is 3
--
AVAIL:SEMAPHORE:=(3,null);
FULL:SEMAPHORE;
MUTEX_BUFF:SEMAPHORE:=(1,null);

function RDM(N:in INTEGER) return INTEGER is

begin
    ILAST:=(ILAST*1233-126) mod N;
    return ILAST;
end RDM;

task PRODUCER;

task CONSUMER;

task body PRODUCER is
K:INTEGER;

begin
    loop
        PUT_LINE("prepare message");
        P(AVAIL);
        P(MUTEX_BUFF);    -- pool of buffers is critical resource
        PUT_LINE("    send message");
        V(MUTEX_BUFF);
        V(FULL);
        --
        -- random wait
        --
        K:=RDM(100);
        for J in 1..K loop
            PUT("");
        end loop;
    end loop;
end PRODUCER;

task body CONSUMER is
K:INTEGER;

begin
    loop
        P(FULL);
        P(MUTEX_BUFF);
        PUT_LINE("            receive message");
        V(MUTEX_BUFF);
```

```
        V(AVAIL);
        PUT_LINE("                process message");
        K:=RDM(150);
        for J in 1..K loop
            PUT("");
        end loop;
    end loop;
end CONSUMER;

begin
    null;
end ESSEM;
```

result of the execution :

```
prepare message
    send message
prepare message
    send message
             receive message
                 process message
             receive message
                 process message
prepare message
    send message
prepare message
    send message
             receive message
                 process message
             receive message
                 process message
prepare message
    send message
prepare message
    send message
prepare message
    send message
prepare message
             receive message
                 process message
             receive message
                 process message
             receive message
                 process message
    send message
prepare message
     send message
             receive message
                 process message
             receive message
                 process message
prepare message
    send message
prepare message
```

```
    send message
prepare message
    send message
prepare message
            receive message
                process message
            receive message
                process message
            receive message
    send message
                process message
            receive message
                process message
prepare message
    send message
```

3.3 Implementation of semaphores with messages.

Semaphores are generally considered to be sufficient for the synchronization and communication requirements of an operating system. However, semaphores are often quite inconvenient because they are too primitive. One can choosing more elegant communication tools. Between a semaphore and a rendez−vous, we can use a semaphore with a message. [2] The latter is based upon the same idea as the general semaphore, only a queue for messages fm not directly sent is added.

Primitive P and V become :

```
P(s,mes) :
        val(s): = val(s) − 1
        if val(s) < 0 then
                                the task is stopped and put in the
                                waiting queue f(s)
                        else
                                remove one message m from fm(s)
                                mes = m
        end if

V(s,mes) :
        val(s): = val(s) + 1
        if val(s) < = 0 then
                                one task is removed from the waiting
                                queue f(s) and is reactived with
                                message mes.
                          else
                                put the message mes in the queue fm(s)
        end if
```

Listing of the package MESSAGE for using semaphores with messages.

```
generic type MESSAGE is private;

package SEMMES is
    type SEMAPHORE is limited private;

    procedure P(SEM:in out SEMAPHORE;
    MES:out MESSAGE);

    procedure V(SEM:in out SEMAPHORE;
    MES:in MESSAGE);
    private
    type FIFO_MES;
    type PTMES is access FIFO_MES;
    --
    -- queue of messages
    --
    type FIFO_MES is record
        MES:MESSAGE;
        NEXT:PTMES:=null;
    end record;
```

```
    task type MUTEX is
        entry P;
        entry V;
    end MUTEX;

    task type FIFO is
        entry P_OUT(MES:in MESSAGE);
        entry P_IN(MES:out MESSAGE);
    end FIFO;
    type PAIR_TASK is record
        FIF:FIFO;
        MUTX:MUTEX;
        FMES,LAST:PTMES:=null;
    end record;
    type PTFIFO is access PAIR_TASK;
    type SEMAPHORE is record
        VAL:INTEGER:=0;
        LIST:PTFIFO:=null;
    end record;
end SEMMES;
with UNCHECKED_DEALLOCATION;

package body SEMMES is

    procedure FREE is
    new UNCHECKED_DEALLOCATION(FIFO_MES,PTMES);

task body MUTEX is

begin
    loop
        accept P;
        accept V;
    end loop;
end MUTEX;

task body FIFO is
BUFMES:MESSAGE;

begin
    loop
        accept P_OUT(MES:in MESSAGE) do
            BUFMES:=MES;
        end;
        accept P_IN(MES:out MESSAGE) do
            MES:=BUFMES;
        end;
    end loop;
end FIFO;

procedure P(SEM:in out SEMAPHORE;
MES:out MESSAGE) is
PC:PTMES;

begin
    if SEM.LIST=null
```

```
      then
        SEM.LIST:=new PAIR_TASK;
    end if;
    SEM.LIST.MUTX.P;
    SEM.VAL:=SEM.VAL-1;
    if SEM.VAL<0
      then
        SEM.LIST.MUTX.V;
        SEM.LIST.FIF.P_IN(MES);
      else
        --
        -- take a message in the queue
        --
        MES:=SEM.LIST.FMES.MES;
        PC:=SEM.LIST.FMES;
        SEM.LIST.FMES:=SEM.LIST.FMES.NEXT;
        if SEM.LIST.FMES=null
          then
            SEM.LIST.LAST:=NULL;
        end if;
        FREE(PC);
        SEM.LIST.MUTX.V;
    end if;
end P;

procedure V(SEM:in out SEMAPHORE;
MES:in MESSAGE)is
PC:PTMES;

begin
    if SEM.LIST=null
      then
        SEM.LIST:=new PAIR_TASK;
    end if;
    SEM.LIST.MUTX.P;
    SEM.VAL:=SEM.VAL+1;
    if SEM.VAL<=0
      then
        SEM.LIST.FIF.P_OUT(MES);
      else
        --
        -- put a message in the queue
        --
        PC:=new FIFO_MES;
        if SEM.LIST.LAST/=null
          then
            SEM.LIST.LAST.NEXT:=PC;
            SEM.LIST.LAST:=PC;
          else
            SEM.LIST.LAST:=PC;
            SEM.LIST.FMES:=PC;
        end if;
        PC.MES:=MES;
    end if;
    SEM.LIST.MUTX.V;
end V;
end SEMMES;
```

4. Conclusion

Event and semaphore synchronization mechanisms use data passing in its most elementary form : permission or ban to continue. They are useful when a task does not know the names of the others tasks. If this type of communication is not sufficient, semaphores with messages can be used. These mechanisms can obviously always be replaced by the general Ada rendez – vous.

References

[1] DIJKSTRA E W
cooperating sequential process in programming languages
Academic press (1967)
The structure ot the multiprogramming system
Cacm 11 5 (1968)

[2] SAAL H J, RIDDLE W E
Communicating semaphores
SLAC,CGTM 117 Stanford (1970)

[3] ANSI
The programming language Ada reference manual
Springer – Verlag 155 (1983)

[4] CROCUS
Systemes d'exploitation des ordinateurs
Dunod (1981)

= =

Références

[1] Abolins M. et al (1985)
GEANT3 evaluation
D0 Note 287 Fermilab.U.S.A.

[2] Appelbe W.F. (1984)
Abstract data types in Ada
UCSD Journal of Pascal and Ada V3 N1

[3] Auguin M.,Jalby W. and Mailliard J. (1985)
Use of SIMD – SPMD machine for simulation in particle physics,
Lassy,Nice,France.

[4] Bobrowicz F.W.,Lynch J.E.,Fisher K.J. and Tabor J.E. (1983)
Vectorized Monte Carlo photon transport,
LA – 9752 – MS,Los Alamos National Laboratory.

[5] Brown F.B. (1981)
Vectorized Monte Carlo, Ph.D. dissertation,
University of Michigan, Ann Arbor, Michigan.

Brown F.B.,Martin W.r. and Calahan D.A. (1981)
A discrete sampling method for vectorized Monte Carlo calculations,
Trans. am. nucl. soc. 38,354.

Brown F.B. (1983)
Vectorized Monte Carlo methods for reactor lattice analysis,
Advances in reactor computations,Salt Lake City, Utah, pp108 – 123.

[6] Brown F.B. and Martin W.R. (1984)
Monte Carlo methods for radiation transport analysis on vector computer,
Progress in nuclear energy,Vol 14,N 3 ,pp269 – 299

[7] Brun R.,Caillat M.,Maire M.,Patrick G.N. and Urban L. (1985)
The GEANT3 electromagnetic shower program and comparaison with the EGS3 code,
CERN – Data Handling Division DD/85/1.

[8] Calahan D.A.,Martin W.R et al (1980)
Final report for preliminary studies on vectorized Monte Carlo,
Los Alamos National Laboratory, University of Michigan report

[9] Control Data Corporation (1985)
CDC cyber fortran langage,
Reference manual.

[10] R Brun and all

HBOOK user's guide
CERN DD/77/9

[11] R Brun and all
GEANT user's guide
CERN DD/EE/84 – 1

[12] R Brun and all
ZEBRA user's guide
CERN DD/EE/85 – 6

[13] Dekeyser J – L. (1985)
Simulation du système de gestion de mémoire pour ISIS,
Université des sciences de Lille 1, UER IEEA.

[14] Dekeyser J – L.,Georgiopoulos C.H.,Levinthal d. et al (1986)
Vectorization of GEANT3 geometrical procedures on a Cyber 205,
SCRI Florida State University report.

[15] P.M. Ferran,A. Fucci et all (1985)
The 3081/E emulator, a processor for use in On – Line and Off – Line arrays
CERN DD/85/16.

[16] Flynn M. (1972)
Some computer organizations and their effectiveness,
IEEE Trans. on computer C – 21 N 9,948

[17] I. Gaines et all (1985)
Use of the Fermilab Advanced Computer Program Multi – Microprocessor
IEEE Trans Nuclear Science V32, N4.

[18] Georgiopoulos C.H. et al (1985)
A non – numerical method for track finding in experimental high energy phyisics using vector computer,
Florida States University FSU – SCRI – 85 – 11

[19] Habermann A.N. (1980)
Efficient implementation of Ada tasks
Tech Rep CMUCS80103 Carnegie Melon University.

[20] Lecouffe M.P. (1979)
MAUD : A dynamic single assignement system
Computers and digital technics april 79 Vol 2 N 2

[21] Lindquist T.E. and Joyce R.C. (1985)
Ada task synchronization in a multi – processor with shared memory
Journal of Pascal Ada and Modula 2,V4 N1 pp 9 – 19

[22] Martin W.R. (1983)
Vectorized Monte Carlo on the Cyber 205,
Final reprot for Control Data Corporation, University of Michigan.

[23] OPAL committee (1983)
The OPAL detector technical proposal
CERN/LEPC/83 – 4

[24] Pair C. and Gaudel M.C (1980)
Les structures de données et leur représentation en mémoire
presse universitaire de Grenoble.

[25] Sherman m. Hisgen a. and Rosenberg j. (1982)
A methodology for programming abstract data types in Ada
Sigplan Adatec 82 Conference on Ada Arlington VA pp 66 – 75.

[26] Timsit C. (1985)
Isis, une machine destinée au calcul scientifique de haute performance,
BULL Louveciennes 78430

[27] Troubetzkoy E.,Steinberg H. and Kalos M. (1973)
Monte Carlo radiation penetration calculations on a parallel computer,
Trans. am. nucl. soc. 17,260.

[28] United states departement of defense (1983)
Reference manual for the Ada programming language
Draft revised MIL – STD 1815

- *RESUME:*

La simulation en physique des particules est un gros consommateur de temps CPU. De plus, la plupart des logiciels de simulation existants ont été écrits en FORTRAN pour des machines séquentielles.

Aussi avons-nous voulu développer de nouveaux algorithmes pour des architectures présentant un aspect parallèle.

Une implémentation de type multi-évènements a été étudiée et complètement réalisée sur une architecture de type maître/esclaves.

Nous poursuivons par l'étude de la vectorisation des algorithmes de simulation, la partie géométrique a été vectorisée et testée sur un CYBER 205.

Enfin, une approche Ada pour ce type de problème, avec les notions de type d'objets, de paquettage générique et de multi-tâches est ensuite présentée.

- *MOTS CLES:*

Simulation, multi-traitements, multi-processeurs, vectorisation, Ada, multi-tâches.