CAMELEON : Un Emulateur d'Architectures Parallèles pour la Validation de Programmes Parallèles

Abderrazak HENNI (*) **Réda NOUACER** (**)

(*) Institut National d'Informatique I.N.I.
B.P. 68 M Oued Smar Alger 16270 Algérie
Tel : (213) 2 51 60 01 – Fax : (213) 2 51 61 56 – Email : henni@ini.dz

(**) Département d'Informatique – Faculté des Sciences – Université de Annaba
B.P. 12 Annaba 23000 Algérie
Tel & fax (213) 8 87 27 56 - E-mail : reda.cose@i-France.com

Résumé :
L'étude du parallélisme dans ses différents aspects, nous a permis d'effectuer un certain nombre de constatations.
Le parallélisme est une technique d'accélération de l'exécution, du fait de la coopération de plusieurs processeurs dans l'exécution d'un même programme.
l'approche parallèle est une méthode de conception, vu qu'elle permet d'appréhender naturellement les problèmes et ainsi de les résoudre efficacement. Cette efficacité ne peut être atteinte ou obtenue que si un bon mariage entre l'application et l'architecture est trouvé; malheureusement ce problème est NP-complet.
A cette difficulté s'ajoute le problème de disponibilité de l'équipement nécessaire à la recherche sur le parallélisme. En effet, si la machine parallèle existe alors elle coûte chère et elle est en nombre d'exemplaires réduit. Si elle n'existe pas, et on doit la réaliser, cela demandera de grands investissements et un nombre d'années non négligeable avant sa concrétisation, d'où souvent dans ce cas les chercheurs tendent vers la simulation ou le prototypage.
Le prototypage coûte cher, demande du temps et exploite seulement quelques points de la conception. Et la simulation est flexible, mais lente et complexe quand la conception est simulée en détail.
Ainsi, de tout ce qui a précédé, on a pensé à une nouvelle approche qui est l'émulation d'architecture. L'idée est simple mais nous semble très intéressante. En effet, dans cette approche - qui n'est ni : de la simulation; ni de l'encastrement; et encore moins du prototypage- une architecture présumée (souhaitée) est décrite et est ensuite supportée par une architecture parallèle réelle de type MIMD. Ainsi, on peut : Vérifier la correction d'un programme relativement à une architecture; déterminer l'architecture la mieux adaptée à un problème donné; développer les outils nécessaires à une architecture (langage, débugger, paralléliseurs, ...); et encore mieux, aider dans la conception d'architecture. Et tous ceci à moindre coût, car une fois le système Caméléon construit, tout ce que le chercheur aura à faire pour obtenir l'architecture de travail c'est de la décrire, et elle sera immédiatement supportée.
Actuellement, et depuis quelques temps, l'idée de l'émulation nous parait comme une solution au problème de placement dans le cas de la programmation logique parallèle.

Mots clés : Architecture parallèle, programme parallèle, parallélisme, simulation, émulation

1. Introduction

Un des problèmes essentiels dans le domaine du parallélisme est la corrélation entre le modèle d'exécution et le modèle de programmation. Aussi, la conception et la réalisation de machines parallèles est une tâche fastidieuse qui nécessite beaucoup d'efforts et d'argent. Mais ceci n'a pas empêché les chercheurs à élaborer des programmes parallèles pour des modèles de machines, le plus souvent qui ne leur sont pas accessibles. Ainsi, la quasi majorité de ces programmes ont été validés d'une façon formelle seulement et non pas par une exécution réelle.

Jusqu'à ce jour deux approches ont été utilisées pour vérifier la conception des machines et programmes parallèles : la construction de prototype et la simulation. La construction de prototype est chère, nécessite du temps, et exploite seulement un ou quelques points de la conception. La simulation logicielle est flexible, par rapport au prototype, mais lente quand la conception est simulée en détail, elle a aussi des problèmes de validité vu que la machine simulée est considérablement abstraite pour maintenir les temps de simulation raisonnables.

De tout ce qui a précédé, on s'est fixé pour objectif de proposer un système multiprocesseur avec l'environnement logiciel approprié, dénommé : **CAMELEON** ; qui devra :

- Permettre l 'émulation des architectures parallèles décrites par l'utilisateur.
- Fournir un support d'exécution réel des programmes parallèles, et permettre ainsi d'obtenir à moindre coût des résultats qui se rapprochent de ceux qu'on pourrait avoir sur la machine décrite si celle-ci est concrétisée ou sera concrétisée un jour.

L'idée de l'émulation nous semble intéressante dans le sens où :

- Elle est mieux que la simulation car elle offre un support d'exécution réel.
- Plus économique que le prototype, car une fois un tel système disponible il est facile de revenir sur des points dans la conception et de les vérifier immédiatement à moindre coûts.
- Pratique, comme outil de recherche sur le parallélisme car elle offre un large éventail de modèles par la simple description, et ainsi permet de vérifier et de comparer les résultats sans avoir à effectuer beaucoup : de déplacements, de dépenses, de gymnastique au niveau de la programmation ; car l'image qu'a le chercheur de l'architecture qu'il souhaite est immédiatement émulée.
- Et c'est un véritable multiprocesseur qui peut être utilisé tel qu'il est.

2. Vision globale et organisation de Caméléon

Tous les multiprocesseurs connus à ce jour obéissent au modèle de base suivant :

Un multiprocesseur = P x M x I.

P : ensemble de processeurs ou unité de calcul.
M : ensemble de modules mémoires (locale et/ou partagée).
I : réseau d'interconnexion et/ou intercommunication.
x : opérateur de composition.

L'équation ci-dessus indique qu'un multiprocesseur, quelque soit sa conception, est toujours obtenu en composant d'une façon ou d'une autre des éléments de contrôle et de traitement, des éléments de mémoire et des dispositifs de connexion.
Donc, ce qui différencie réellement les multiprocesseurs, plus que leur performance, c'est leur organisation. Celle-ci, en fait, rend la machine mieux adaptée à une certaine classe de problèmes plus qu'à d'autres.
Partant de cette constatation et de l'idée que Caméléon est un multiprocesseur destiné à l'émulation d'architectures parallèles, on s'est posé un certain nombre de conditions. Caméléon devra ainsi :

- Avoir un profil de communication (N -> N) régulier, et ceci pour permettre le plongement de n'importe quel schéma de communication.
- Fiable du point de vue pannes et résultats.
- Non complexe, afin qu'il soit techniquement réalisable.
- Non coûteux : pour qu'il reste à la portée d'une plus large population.
- De préférence extensible : pour ne pas être limité en nombre de processeurs.
- Disposer des outils nécessaires à sa programmation, paramétrage et l'obtention de mesures de performances pour les programmes exécutés dessus.

De toutes ces conditions, on est arrivé à la conclusion que seule une architecture MIMD à mémoire physiquement distribuée pouvait répondre à nos besoins pour les raisons suivantes :

1. Un MIMD à mémoire physiquement distribuée : car un système à mémo ire partagée ne permet pas la connexion d'un nombre important de processeurs d'un côté, et d'un autre on a préféré un système à mémoire physiquement distribuée et ceci pour pouvoir construire Caméléon à partir de modules existants.
2. Le modèle SIMD peut être obtenu sur un MIMD par l'approche SPMD, mais pas l'inverse.
3. Un système MIMD, est généralement constitué de processeurs à usage général et ceux-ci disposent d'un jeu d'instructions assez important.

Le système Caméléon vu logiquement est organisé de la façon suivante :

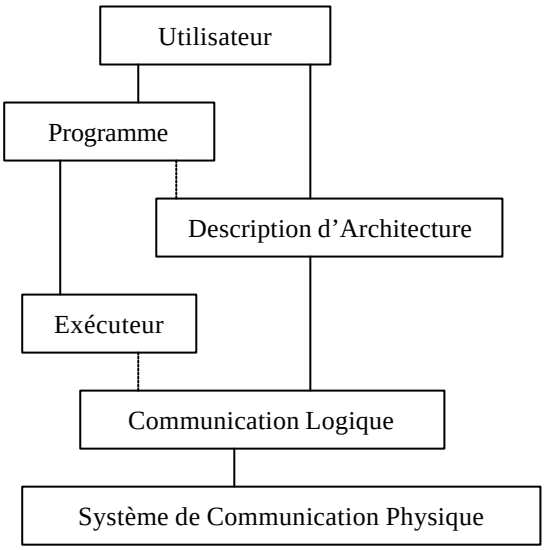


Figure : Organisation Logique de Caméléon

- ***Programme*** : Correspond aux phases usuelles du cycle de vie d'un programme.
- ***Description d'Architecture*** : l'utilisateur (humain ou outil) à ce niveau décrit l'architecture souhaitée par un Net-List et la soumet au système de communication logique.
- ***Exécuteur*** : gère l'exécution du programme parallèle sur l'architecture logique, à ce titre il est chargé du placement, ordonnancement et d'assurer la cohérence des données partagées.
- ***Communication Logique*** : est chargée de masquer l'architecture physique (respectivement : émuler l'architecture logique).
- ***Système de communication physique*** : représente l'architecture physique avec son logiciel de communication.

3. Présentation des différents éléments de Caméléon

3.1. Description d'Architecture : Construction du Net-List

Le Net-List est l'élément fondamental dans Caméléon. En effet, l'architecture à émuler est décrite sous forme d'un Net-List qui exprime les différentes connexions entre nœuds.

Un nœud du point de vue utilisateur est un module capable de : effectuer des calculs, mémoriser l'information qu'il manipule (mémoire locale), et capable de communiquer avec ses voisins (ceux donnés par le Net-List).

Il est à noter que la notion de mémoire partagée est prise en charge via le concept de mémoire partagée virtuelle.

Un élément du Net-List est un n-uplet qui indique pour chaque nœud ses voisins présumés. Par exemple : 1 (0, 2, 7) signifie que le nœud n°1 a pour voisins directs les nœuds : 0, 2 et 7.
Chaque n-uplet est transmis au nœud correspondant afin qu'il puisse gérer la communication selon le schéma prévu par l'utilisateur.

3.1.1.*Exemple de Net-List*

a) Un Hypercube : dimension = 4 et de degré = 4 ; le Net-List correspondant est :

Inter { 0 (1, 2, 4, 8) ; 1 (0, 3, 5, 9) ; 2 (0, 3, 6, 10) ; 3 (1, 2, 7, 11) ; 4 (0, 5, 6, 12) ; 5 (1, 4, 7, 13) ;
6 (2, 4, 7,14) ; 7 (3, 5, 6, 15) ; 8 (0, 9, 10, 12) ; 9 (1, 8, 11, 13) ; 10 (2, 8, 11, 14) ; 11 (3, 9, 10, 15) ;
12 (4, 8, 13, 14) ; 13 (5, 9, 12, 15) ; 14 (6, 10, 12, 15) ; 15 (7, 11, 13, 14) }

b) Un arbre : degré = 4; le Net-List correspondant est :

Inter { 1 (2, 3, 4) ; 2 (1, 5, 6, 7) ; 3 (1, 8, 9, 10) ; 4 (1, 11, 12, 13) ; 5 (2) ; 6 (2) ; 7 (2) ; 8 (3) ; 9(3) ; 10 (3) ;
11 (4) ; 12 (4) ; 13 (4) }

c) Un Tore : dimension = 2 et base = 3 ; le Net-List correspondant est :

Inter { 0 (1, 2, 3, 6) ; 1 (0, 2, 4, 7) ; 2 (1, 0, 5, 8) ; 3 (0, 4, 5, 6) ; 4 (1, 3, 5, 7) ; 5 (2, 3, 4, 8) ; 6 (0, 3, 7, 8) ;
7 (1, 4, 6, 8) ; 8 (2, 5, 6, 7) }

3.1.2. *Construction du Net-List* :

Cette liste des connexions est construite par l'utilisateur sous la forme indiquée ci-dessus. En fait, celui-ci utilise un éditeur pour saisir les n-uplets du Net-List, où chaque n-uplet occupe une ligne.

Durant la phase initialisation du système et après la numérotation des différents nœuds, le Net-List est diffusé vers les différents nœuds où chacun enregistre le n-uplet le concernant et construit la liste des voisins optimaux. Et ce n'est qu'une fois cette opération effectuée, qu'une exécution parallèle peut prendre place.

Il serait intéressant et utile de concevoir un moyen et/ou un formalisme permettant de définir ce Net-List, sans que l'utilisateur soit obligé de donner la liste de toutes les connexions souhaitées. on pense que ceci est faisable pour les architectures régulières.

3.1.3. *Liste des voisins optimaux*

Chaque site entretien sa propre liste des voisins optimaux. Cette liste est utilisée dans le cas des communications indirectes (nécessitant un routage), par rapport à la topologie décrite par l'utilisateur, où il faut prendre le chemin optimal défini selon un ensemble de critères, par exemple le plus court. Elle est nécessaire pour éviter de recalculer, à chaque communication indirecte, le chemin optimal.

C'est une liste propre à chaque nœud, elle est construite durant la phase initialisation – paramétrage au niveau de chaque nœud pour profiter de la puissance de calcul déjà disponible durant cette phase.

Cependant, selon notre vision cette liste ne devra pas contenir tout le chemin à emprunter, ceci est inutile et encombrant, mais seulement l'identité du voisin optimal.

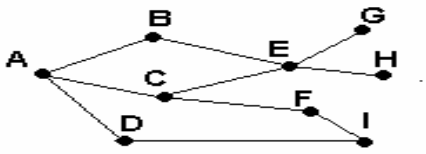
Définition 1 : « Voisin optimal »

Soit un graphe non orienté **G**, constitué de N nœuds (X_1, X_2, …, X_N). Soit le chemin optimal (X_{i1}, X_{i2}, …, X_{ik}) entre deux nœuds (X_{i1},X_{ik}).

Le **voisin optimal** est le nœud X_{i2} ; i.e : le premier nœud traversé par un message en partant de X_{i1} et qui appartient au chemin optimal.

Définition 2 : « La liste des voisins optimaux », est constituée de couples (A,B) tel que : B est le nœud destination finale d'un message en partant du nœud local, et A le voisin optimal pour parvenir à B.

Exemple : soit le graphe G suivant :



- Le chemin optimal, selon le critère du plus court chemin, de G à F est (G, E, C, F).
- Le voisin optimal de G est E.
- Ainsi, on retrouvera dans la liste des voisins optimaux de chacun des nœuds du chemin les couples suivants : en G : (E, F) ; en E : (C, F) ; en C : (F, F).

Si le Net-List n'est pas chargé par l'utilisateur, le système supposera que l'architecture présumée est l'architecture physique actuelle utilisée par Caméléon, et donc on supposera que l'utilisateur cherche à utiliser Caméléon en tant que multiprocesseur effectif et non comme un émule.

3.2. Gestion de la communication logique

Cette couche est le véritable émule d'architecture dans Caméléon. Son rôle est de servir d'interface logique entre le système de communication physique d'une part, et l'exécuteur et programme utilisateur d'autre part. Elle offre des points d'accès aussi bien au noyau d'exécution qu'aux programmes utilisateurs. Ceux-ci représentent les routines de communication (Send et Receive) et d'identification du processeur local (idproc).

A cet effet, elle est chargée de :

- Garder la liste des voisins présumés et la liste des voisins optimaux présumés (on entend par présumés : ceux indiqués par le Net-List).
- Gérer la communication avec eux.

La couche de communication logique, est capable d'acheminer trois types de messages logiques :
Message direct, Message indirect et Message diffusion.

3.2.1. *Message Direct*

C'est un message qui est envoyé (resp. reçu) à un nœud figurant dans la liste des voisins définis précédemment par le Net-List.

Lorsqu'un message est émis par le programme utilisateur, la destination est cherchée dans la liste des voisins. Si elle est retrouvée alors le message est classifié de type DIRECT et il est expédié vers sa destination.

3.2.2. *Message indirect*

C'est un message envoyé vers un nœud ne figurant pas dans la liste des voisins et dont l'identité est différente du nœud générique EVERY indiquant la diffusion. Ce type de message est géré ainsi:

a) Nœud émetteur :
- Le programme envoie un message vers une destination X.
- La destination est cherchée dans la liste des voisins et comparée avec la destination générique EVERY, si la recherche n'est pas fructueuse alors le message nécessite un routage.
- La couche détermine le voisin optimal correspondant à la destination en utilisant la liste des voisins optimaux.
- Marque le message indirect et l'envoie vers le voisin optimal.

b) Voisin optimal
- Reçoit un message marqué indirect d'une source Y et destiné à un nœud X.
- La couche de communication logique locale, qui a reçu un message marqué indirecte conclu qu'il ne lui est pas adressé.
- Une recherche dans la liste des voisins optimaux est lancée :

Si *le voisin optimal n'est pas la destination*
Alors
Il lui envoie à son tour le message toujours de type indirect en lui indiquant la destination.
Sinon
Si *ce voisin optimal est la destination*
Alors
Le message est transformé en message direct et lui est envoyé.
Fin
Fin

Remarque : Bien que la topologie physique de Caméléon offre un schéma de communication N-> N, le message indirect a été introduit pour représenter fidèlement la topologie présumée et montrer l'effet d'une communication avec routage sur l'algorithme utilisateur.

3.2.3. *Message de type diffusion*

C'est un message adressé à tous les nœuds du système. Ce message à son émission par le programme indique la destination EVERY.

Ce message est transmit par la couche de communication logique comme suit :

- ✍ Construit un message de type direct et l'envoi à tous les nœuds figurants dans la liste des voisins directs (d'après le Net-List).
- ✍ Construit un message de type indirect et l'envoi à tous les autres nœuds, en utilisant cette fois-ci la liste des voisins optimaux.

3.3. Topologie et système de communication physique

Caméléon est doté d'une couche logiciel, intitulée : Système de communication physique, qui a pour rôle d'assurer l'indépendance vis-à-vis de la topologie physique. En effet, cette couche se charge de gérer la communication directe avec le hard. A cet effet, c'est elle qui est chargée de l'émission et réception de messages, ainsi que de leur construction et interprétation physique. Cette couche travaille en étroite collaboration avec la couche communication logique en lui fournissant des points d'accès sous forme d'appels superviseurs nécessaires à l'envoie et réception de messages.

Physiquement, Caméléon se présente comme étant un multiprocesseur MIMD faiblement couplé dont le réseau d'interconnexion à un schéma de communication N -> N, et les différents nœuds sont homogènes du point de vue opérationnelle.

La mise en œuvre actuelle utilise des micro-ordinateurs comme nœuds de l'architecture de Caméléon. **Pourquoi utiliser des PCs** ?

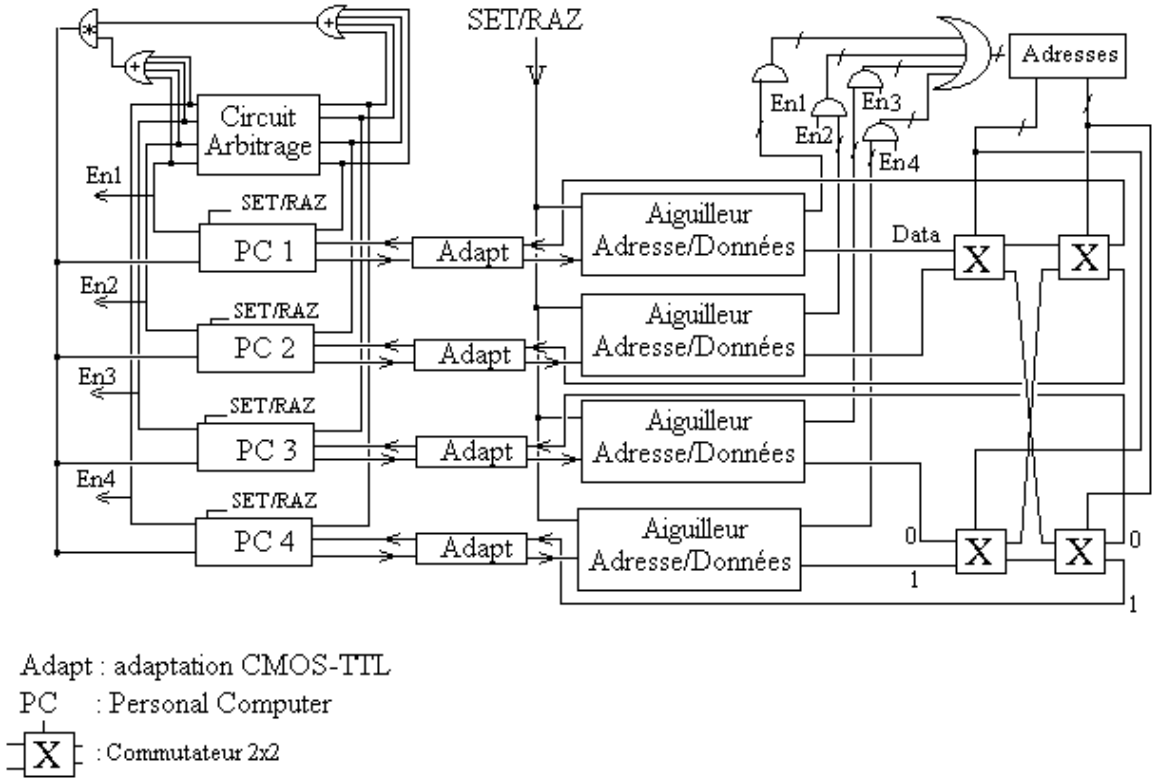
- ✍ L'abstraction d'un PC donne : {Processeur, Mémoire, Bus Interne, Interface E/S}
- ✍ Ils sont pratiquement disponibles partout
- ✍ Leurs prix tendent à diminuer considérablement à l'inverse de leurs capacités qui augmentent considérablement.
- ✍ Ils offrent différentes possibilités d'interfaçage et de connexion qui permettent une construction aisée de Caméléon

Le réseau d'interconnexion implémante un réseau Omega et apparaît sous la forme d'un boîtier placé entre les PCs. Ce boîtier présente plusieurs prises numérotées. Celles-ci devront être utilisées dans l'ordre croissant de leur numéro et ceci à cause du mode de numérotation employé.

A la place d'un réseau multi-étages Omega, on pouvait utiliser une topologie en anneau ou bus, permettant aussi le schéma N->N et qui ne coûte pas cher; mais leur problème est le temps de communication entre deux noeuds qui n'est pas uniforme. Ce qui rend la tâche de projection (plongement), et surtout de conversion du temps de communication réelle en un temps théorique très difficile, alors que dans le cas d'un réseau multi-étage ce temps est le même et peut être considéré comme unité de temps, à la place de la seconde, pour les temps de communication.

En effet, cette substitution de la seconde par la notion d'unité de temps (égal à t) permet la comparaison des une la comparaison objective et dépend de la technologie utilisée.

Schéma du réseau d'interconnexion :



Protocole :

1. Le PC envoie un signal demande l'autorisation d'accès au réseau adressé au circuit d'arbitrage.
2. Le circuit d'arbitrage, conçu pour privilégier le PC de faible numéro, renvoie un signal d'autorisation vers le PC demandeur prioritaire et un signal de refus vers les autres.
3. Le PC autorisé, commence par envoyé une trame adresse composée des deux adresses source et destination.
4. Ensuite, il envoie l'information utile.
5. En fin, il envoie un signale de réinitialisation du réseau.

Remarques

1. Si la réalisation de ce boîtier s'avère délicate, on pourrait utiliser n'importe quel réseau d'interconnexion disponible ou faisable au prix de la perte de l'uniformité du temps de communication entre nœuds.

2. Tous les autres aspects de Caméléon ne devront pas poser de problèmes pour leur réalisation, puis qu'ils relèvent de la programmation.

3.4. Caméléon et la mémoire partagée

Pour que Caméléon puisse être réellement un émule d'architectures parallèles, il faut qu'il supporte les architectures aussi bien fortement que faiblement couplées. Donc, il lui faut supporter les deux modes d'échange et de partage de l'information.

De part sa construction à partir de module existant, Caméléon est physiquement un système faiblement couplé à échange de message et lui faut supporter la notion de mémoire partagée. Pour y parvenir on a suivi le mouvement de ces dernières années dans la conception de multiprocesseur qui consiste à exploiter la notion de mémoire partagée virtuelle.

3.4.1. *Constructions pour la définition et l'exclusion d'accès aux structures partagées*

Les structures partagées dans Caméléon sont d'abord définies et puis manipulées, leur modification étant effectuée en exclusion mutuelle.

La définition de la liste des structures partagées au niveau du programme est effectuée grâce à la construction **$Shared** dont la syntaxe est la suivante :

$Shared < Liste des structures partagées > **$EndShared**

Après précompilation, cette construction est traduite en une routine intitulée **Shared** qui se charge de la construction de la liste des structures partagées entretenues au niveau de la couche système d'exécution (l'exécuteur), et qui servira au maintien de la cohérence de ces structures durant l'exécution.

A chaque structure partagée est associé un verrou binaire utilisé lors des manipulations en exclusion mutuelle. Les opérations en exclusion mutuelle sont contrôlées par deux constructions, l'une pour le verrouillage et l'autre pour l'opération inverse. Une section critique a la forme suivante :

$Lock < liste des structures partagées > **$EndShared**

< Suite d'instructions >

$UnLock

L'opération de verrouillage est à deux phases, et est régie par un droit de verrouillage, ce qui permet d'éviter l'interblocage.

3.4.2. *Comment la cohérence est maintenue*
Durant les phases de précompilation et compilation, les constructions **$Shared** et **$Lock/$UnLock** ont été traduit en leurs équivalents opérationnels. Aussi, durant la même opération une analyse du programme est effectuée où le compilateur repère les structures partagées manipulées au niveau de chaque tâche parallèle, de cette façon pour chaque tâche le système ne traitera que les structures partagées manipulées à son niveau.
Aussi, lors de l'affectation d'une tâche à un nœud le système ne lui transmettra que ce qui est manipulé par elle. A la fin de son exécution, l'information modifiée contenue dans ces structures est récupérée pour mettre à jour la copie principale.

3.5. Le système d'exécution : **l'exécuteur**
Le noyau d'exécution qu'on propose manipule un programme entièrement parallélisé. Celui-ci peut être écrit directement par un programmeur ou être le résultat d'une opération de parallélisation en utilisant un outil approprié.
Le modèle d'exécution, actuellement adopté, est de type SPMD, le site initiateur de l'exécution se comporte comme étant le maître ou le superviseur de l'exécution, et il se chargera à ce titre de l'opération d'ordonnancement.

4. Validation de l'émulation de topologie par Caméléon
Jusqu'au stade actuel de la conception du système Caméléon l'accent a été mis essentiellement sur l'aspect organisationnel des architectures parallèles : le réseau d'interconnexion ou de communication ; il reste le deuxième aspect qui est le placement et l'ordonnancement.

Un réseau de communication peut être caractérisé par les paramètres suivants :

- Diamètre : qui correspond au temps (distance) maximum pour qu'un message aille d'un nœud à un autre dans le réseau. Il permet de placer une barre minimum sur le délai nécessaire à la propagation de l'information à travers le réseau.
- Débit : C'est le nombre de message que le réseau est capable de délivrer par unité de temps. Il représente la puissance du réseau, c'est un facteur qui est beaucoup plus lié à la technologie utilisée qu'à l'organisation du réseau.
- Degré : c'est le nombre de canaux d'entrée/sortie d'un nœud. Ceci affecte le prix d'une nœud et la complexité de la logique de contrôle des routeurs.

Ainsi, pour montrer la validité de notre approche, on va montrer que les caractéristiques topologiques de l'architecture émulée sont préservées par Caméléon.

4.1. Le degré

Preuve

- ✍ Chaque nœud physique détient à son niveau une liste des voisins directs logiques, donnés par le Net-List.
- ✍ Le système de communication de Caméléon avant d'envoyer un message d'un nœud à un autre consulte cette liste et détermine ainsi si la communication est possible ou non.

Conclusion : on voit donc que par construction, le système de communication préserve le degré des nœuds logiques via la liste des voisins directs.

4.2. Chemin et distance

Preuve

- ✍ Toute architecture, qu'on note : X, peut être représentée par un graphe **G**. Soit CO_X l'ensemble des chemins optimaux (P_i, P_{i1}, P_{i2}, … , P_j) reliant les sommets P_i et P_j du graphe **G**. Soit CP_X l'ensemble des chemins possibles entre les sommets de N dans le graphe **G**. L'ensemble CO_X est un sous-ensemble de CP_X.
- ✍ Le schéma de communication dans Caméléon est N -> N (équivalent à une topologie complètement connectée). L'ensemble CP_C (chemin possible de Caméléon) de celui-ci couvre donc toutes les combinaisons de chemins possibles.

Conclusion 1: ainsi, l'ensemble des chemins possibles de toute architecture X est inclus dans celui de Caméléon : CP_X ? CP_C.

- ✍ Le routage dans Caméléon est effectué à la base du Net-List qui décrit la topologie X émulée. En effet, deux nœuds pour communiquer disposent de trois types de messages :

? Message diffusion : utilisé pour l'envoi d'un message d'un nœud vers tous les autres.

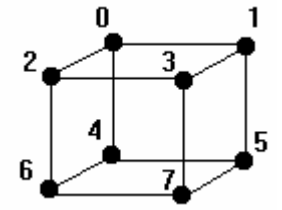
? Message direct : utilisé pour les nœuds logiquement voisins.

? Message indirect : utilisé pour les nœuds logiquement non voisins. Ce message emprunte le chemin optimal calculé durant la phase paramétrage-initialisation à la base du Net-List.

Conclusion 2 : On voit que l'ensemble CO_C (des chemins optimaux) dans Caméléon n'est pas fixe mais change en fonction de la topologie émulé. Et qu'un message emprunte (par conception du routage dans Caméléon) le même chemin optimal qu'il aurait pu emprunter sur l'architecture émulée. Ce qui implique que $CO_C = CO_X$.

Conclusion : Ainsi, du fait que Caméléon préserve (par émulation) les caractéristiques fondamentales de toutes les topologies, le degré et la distance, on peut dire que l'émulation topologique est assurée.

5.Un exemple : Hypercube de degré = 3 et de dimension = 3



a) Liste des chemins optimaux de l'hypercube

- ✍ Critère : plus court chemin.
- ✍ Règle de choix : plus faible numéro de nœud.

Source\Destination	0	1	2	3	4	5	6	7
0		Direct	Direct	1	Direct	1	2	1 , 3
1	Direct		0	Direct	0	Direct	0 , 2	3
2	Direct	0		Direct	0	0 , 1	Direct	3
3	1	Direct	Direct		1 , 0	1	2	Direct
4	Direct	0	0	0 , 1		Direct	Direct	5
5	1	Direct	1 , 0	1	Direct		4	Direct
6	2	2 , 0	Direct	2	Direct	4		Direct
7	3 , 1	3	3	Direct	5	Direct	Direct	

b) Dans Caméléon

✍ Liste des voisins directs

Noeuds	Liste des voisins par Nœud
0	1, 2, 4
1	0, 3, 5
2	0, 3, 6
3	1, 2, 7
4	0, 5, 6
5	1, 4, 7
6	2, 4, 7
7	3, 5, 6

✍ Liste des voisins optimaux

Source\Destination	0	1	2	3	4	5	6	7
0		1	2	1	4	1	2	1
1	0		0	3	0	5	0	3
2	0	0		3	0	0	6	3
3	1	1	2		1	1	2	7
4	0	0	0	0		5	6	5
5	1	1	1	1	4		4	7
6	2	2	2	2	4	4		7
7	3	3	3	3	5	5	6	

6. Conclusion

En effet, chaque stratégie correspond à un type de charge. Si l'on sait que la charge du système doit être faible au cours du temps, mieux vaut utiliser les stratégies actives, si la charge doit être très élevée, il est préférable d'adopter une stratégie passive, et enfin dans le cas d'une charge subissant de fortes fluctuations, les stratégies mixtes sont les plus indiquées.

Parmi les algorithmes existants, seuls ceux basés sur une connaissance totale de l'état du système permettent une bonne sélection de station, quelque soit le nombre de sites inactifs dans le réseau. Dans cette classe les algorithmes centralisés ont une très bonne extensibilité par rapport aux algorithmes totalement répartis mais une faible tolérance aux pannes. L'extensibilité, pour les algorithmes répartis est limitée à cause du support de communication (surcharge). Ces derniers sont plus simples d'un point de vu conceptuel et ne nécessitent pas de procédures de reprise sur les panne sophistiquées étant donné que toutes les stations jouent le même rôle dans la répartition de charge globale. Mais ils introduisent un overhead supplémentaire à cause de l'interruption régulière des stations pour une évaluation de la charge. Mais il semble, d'après la littérature, que dans la majorité des systèmes, une conception décentralisée simple est plus adéquate.

La régulation de charge entraîne avec elle d'autres problèmes dus à la migration de tâches, et ainsi on se trouve confronté aux problèmes de coût de transfert du contexte actif d'un processus y compris son espace d'adressage et de coût de communication entre processus dus à la complexité des protocoles de désignation et de routage : quand un processus migre, les messages qui lui parviennent pendant la durée de migration doivent lui être retransmis. Le processeur initial doit donc re-émettre les messages qui lui sont désignés et informer tous les processeurs du réseau de la nouvelle localisation du processus.

REFERENCES BIBLIOGRAPHIQUES:

[Cab 86] : L.F.Cabrera. "the influence of work load on load balancing strategies".Proceeding of summer 1986 Usenix COnference, Atlanta (USA), janvier 1986, pp 446-458.

[Chow 79] : Y.C.Chow, W.H.Kowler, "Models for dynamic load balancing in heteregenous multiple processor systems", IEEE Trans. on Comp., Vol.C-28, Nø 5, Mai 1979.

[Eage 86] : Derek, Eager, Edaword D.Lazowska, and John Zahorjan, "Adaptive load sharing in homogeneous distribued systems,"IEEE, pp 662-667, mai 86.

[Efe 82] : K.Efe: "Heuristic models of task assignement scheduling in distributed systems", computer, 15, june 82, 50/56.

[Foll 89a]: BertilFolliot,Michel Ruffin,GATOS:"un g,rant de tƒches distribu,",Convention UNIX 89,Paris,AFUU,fevrier 1989,Rapport de recherche du laboratoire MASI nø 263, janvier 89.

[Foll 89b] : Bertil Folliot.r,paatition de charge, "Outils pour le d,veloppement d'applications parallŠles et distribu,es", Rapport de recherche du laboratoire MASI nø308 Universit, Piere et Marie Curie, Paris, juin 89.

[HCG82] : K.Hwang et al., "A Unix-based local computer network with load balancing", IEEE Computer, Vol.15, Nø4, Avr 1982.

[Kuc 91] : H.Kuchen and A.Wagner, "Comparison if dynamic load balancing strategies", jouranal of parallel and distributed processing,303-314,1991.

[Lann 81] : G.Lelann, "A distributed system for real time trabsaction processing", IEEE computer Vol 14, pp 43-48, fevrier 81.

[Litz 88] : M.J.Litzkow, M.Lizny et M.W.mutka. "Condor a hunter of idle workstations". Proceedings of 8th International conference on distributed compating systems, San jose (USA), IEEE,Juin 88, pp104-111.

[Livn 82] : M.Livy and M.Melman, "load balancing in homogenous broadcast distributed systems", in proc.ACM comput.network perfermance symps, pp 47-55.1982.

[RAJ88] : T.M.Ravi, D.Jefferson, "A basic protocol for routing messages to migrating processes", Proc. of the Int. Conf. on Parallel Processing, The Penn. State, pp 188-197, Aout 1988.

[SMI88] : R.G.Smith, "The contract net protocol : High level communication and control in a distributed problem solving", Readings in Distributed Artificial Intelligence", Morgan Kauffman Publishers, California, pp 357-366, 1988.

[Sta84] : J.A.Stankovic, "A perspective on distributed computer systems", IEEE Trans, on Comp., Vol.C-32, Nø12, Dec 1984.

[STS84] : J.A.Stankovic, I.S.Sidhu, "An adaptive bidding algorithm for processes, clusters and distributed groups", Proc, 1984 Int. Conf. on Parallel Proc., pp 49-59, Aug 84.

[Stan 85] : J.A.Stankkovic. "An application of Bayesian Decision theory to decentralised Control of job scheduling", IEEE Trans-on Computer, Vol. C-34, nø2, pp117-130, fevrier 85.

[Thei 88]: M.M.Theimer et K.A.Lautz.Finding idle machines in a workstation.Based distributed system.Proceeding of international conference, IEEE,1988,pp112-122.

[Wang 85] : M.T.Wang et R.J.T.Moris: "load staring in distributed systems",IEEE Transaction on computers, Vol.34,nø3, Mars 1985, pp 204-217.

[Witt 80] : L.Wittie abd A.M.Van Tilborg, "Micros, a distributed operating system for micronet, A reconfigurable network computer", IEEE Trans.Comput., Vol.C-29, december 80.