

# Les systèmes multi-processeurs

Ciarimboli Julien, Hourdin Vincent

Juin 2004



# Table des matières

<b>Introduction</b>	<b>i</b>
<b>1 Fonctionnement des systèmes uniprocesseurs</b>	<b>1</b>
1.1 Les caches . . . . .	1
1.1.1 Fonctionnement de la mémoire cache . . . . .	1
1.1.2 Les différentes organisations de mémoires caches . . . . .	1
1.1.3 Politiques d'écriture . . . . .	2
1.2 Les interruptions . . . . .	2
1.2.1 Qu'est ce qu'une interruption ? . . . . .	2
1.2.2 Le contrôleur d'interruptions . . . . .	3
<b>2 Le SMP et ses problèmes</b>	<b>5</b>
2.1 Le SMP . . . . .	5
2.2 Cohérence des caches . . . . .	6
2.3 Problèmes des interruptions . . . . .	8
<b>3 Les solutions apportées aux problèmes</b>	<b>9</b>
3.1 Solutions pour les caches . . . . .	9
3.1.1 Solutions logicielles . . . . .	9
3.1.2 Solutions matérielles . . . . .	12
3.2 Solutions pour les interruptions . . . . .	12
<b>Conclusion</b>	<b>15</b>



# Introduction

De nos jours, les ordinateurs occupent une place prépondérante dans notre société. Leur puissance ne cesse de croître afin de répondre aux besoins de performances sans limites des super-calculateurs, serveurs, ainsi que ceux des ordinateurs de bureau. Face à la course à la précision de gravage du silicium, de nouvelles architectures sont apparues : les systèmes à plusieurs processeurs (MP). L'augmentation de la fréquence d'un système uniprocresseur étant onéreuse et pas forcement possible, il devient plus abordable de réunir plusieurs processeurs dans un même système.

Le fait d'avoir plusieurs processeurs permet d'exécuter réellement plusieurs tâches en même temps, ce qui peut être non négligeable pour un serveur qui doit fournir des services à plusieurs clients simultanément. De plus, lorsqu'un processeur tombe en panne, on peut le remplacer sans arrêter le système.

Nous allons étudier plus particulièrement les SMP : Ce sont les systèmes multi processeurs qui utilisent symétriquement les ressources du système. Les processeurs ont le même statut, pas de hiérarchie maître-esclave, et chacun peut communiquer avec tous les autres. Tous les processeurs partagent le même espace mémoire et ont accès à cet espace par la même adresse et ils partagent aussi le même système d'entrées/sorties et tous les processeurs peuvent recevoir des interruptions de toutes les sources.

De part ces partages, des incohérences peuvent apparaître en mémoire à cause des accès simultanés et des mémoires caches, et les interruptions ne doivent pas paralyser le système. C'est ce nous allons mettre en évidence dans ce travail d'étude.

Nous commencerons par étudier les systèmes uniprocesseurs, puis les problèmes introduits par le SMP. Nous verrons ensuite les solutions qui ont été mises en place par les constructeurs et les programmeurs de systèmes d'exploitations.



# Chapitre 1

## Fonctionnement des systèmes uniprocesseurs

### 1.1 Les caches

Un cache est une mémoire rapide, représentant un sous-ensemble de la mémoire principale et se trouve entre le CPU et la mémoire principale. Il est proche du processeur pour diminuer la latence. On peut en trouver plusieurs niveaux, en général deux, intégrées au processeur, parfois trois, situés en partie à l'extérieur. La mémoire cache est plus rapide, mais plus coûteuse, c'est la raison pour laquelle on la trouve en quantité moindre dans le système. Aussi, puisqu'elle représente un sous-ensemble de la mémoire centrale, tout n'y est pas représenté, il faut donc étudier leur conception pour qu'elle soit la plus efficace possible. La mémoire cache utilise la localité temporelle (on réutilise souvent la même zone mémoire) et spatiale (on accède souvent aux zones de données contigües). Elle est transparente au programmeur.

#### 1.1.1 Fonctionnement de la mémoire cache

Une mémoire cache est organisée en plusieurs lignes. Pour que ce soit rapide et efficace, on associe un index à une adresse en utilisant une fonction qui hache l'adresse de la mémoire pointée ; l'implémentation la plus simple est le modulo, qui, puisque la taille de la cache est une puissance de 2, est simple à calculer (et binaire). Chaque ligne de cache est formée d'un tag et de données ; le tag est l'information nécessaire à la reconstitution de l'adresse mémoire (en plus de l'index qui est sa partie inférieure), pour pouvoir la comparer avec l'adresse demandée. En effet, puisque le cache est plus petit, plusieurs adresses de la mémoire centrale ont le même index donc il faut être sûr que l'on accède bien à la bonne ligne de cache.

Lorsqu'un processeur veut accéder à une adresse mémoire, deux cas se présentent : soit le cache contient la valeur associée à cette adresse, on parle alors de *cache hit*, soit le cache ne la contient pas et on parle de *cache miss*. Dans ce cas la valeur est chargée dans le cache pour ne pas avoir deux ratés de suite.

#### 1.1.2 Les différentes organisations de mémoires caches

La plus simple organisation est la *direct mapped* dite aussi *one-way set associative cache*. Dans ce type de cache, l'adresse hachée produit un index pour une seule ligne où sera stockée la donnée. Quand on a un cache hit, la donnée est extraite du cache et envoyée directement au

cpu. En revanche si le tag ne correspond pas à l'adresse, on se retrouve dans le cas d'un cache miss. L'inconvénient d'une telle organisation est alors que beaucoup d'adresses correspondent à la même ligne dans le cache.

Une seconde organisation aussi assez simple est le *n-way set associative* cache. Elle essaie d'améliorer les performances du *direct mapped* en indexant un ensemble de N lignes. L'avantage est que si plusieurs adresses dans un programme génèrent le même index, on pourra garder N de ces données. Par contre ces caches sont légèrement plus couteuses car le tag des N lignes d'un ensemble doivent être comparés en même temps pour avoir un temps d'accès raisonnable, et la politique de remplacement est plus compliquée qu'en *one-way set associative*, comme on a plusieurs lignes par index, il faut savoir laquelle remplacer.

La dernière organisation est la *fully associative* cache. Là, on n'a qu'un seul ensemble qui contient toutes les lignes du cache. Il n'y a plus de fonction de hachage ni d'index, mais toutes les lignes doivent être comparées en parallèle, ce qui la rend assez complexe à réaliser, et augmente le coût. C'est pour cette raison que ce type de cache est rarement utilisé pour les caches de données des processeurs. Avec ces caches on peut mettre les données de n'importe quelle adresse sur n'importe quelle ligne, donc on peut arriver à remplir tout le cache avec les données du programme et obtenir un ratio hit/lookups proche de 1.

### 1.1.3 Politiques d'écriture

Quand un processeur veut écrire une donnée en mémoire, la mémoire cache modifie la valeur dans ses données mais peut choisir suivant l'implémentation de ne pas mettre à jour la mémoire centrale, qui deviendra donc inconsistante, et mettre un bit *modifié* à 1 dans la ligne qui contient la donnée. Si cette ligne est destinée à être remplacée à un moment, l'algorithme verra ce bit et mettra à jour la valeur dans la mémoire centrale. Cette politique est appelée *write back policy*. L'autre politique, dans tous les cas, écrit les données en mémoire centrale (en plus de l'écrire dans le cache bien sûr), d'où son nom *write through policy*. La mémoire centrale sera donc toujours cohérente, mais chaque écriture sera plus lente car elle devra accéder à la mémoire centrale qui est plus lente que la cache. On perd donc le bénéfice du cache en écriture, mais dans tous les cas on accèdera à la valeur en lecture à partir du cache.

## 1.2 Les interruptions

### 1.2.1 Qu'est ce qu'une interruption ?

Une interruption consiste en une modification d'état sur un port d'entrée/sortie, ou une attention asynchrone envers le processeur. Le processeur doit accomplir des tâches comme rafraîchir l'écran, afficher l'heure, au lieu d'utiliser le polling qui consiste à une routine qui tourne en permanence, utilise du temps CPU, on utilisera les interruptions. Quand un port d'entrée/sortie souhaite l'attention car une donnée est arrivée ou le statut a changé, il envoie une requête d'interruption (IRQ) au processeur.

Quand le processeur reçoit une interruption, il finit son instruction en cours, avant de placer ce qu'il faut sur la pile et sauvegarder ses données privées pour savoir où il doit revenir et exécuter l'action associée au signal (ISR - Interrupt Service Routine).

Si l'interruption apparaît quand le processus est dans une section critique du noyau ou utilisateur, le traitant d'interruption peut modifier ces données critiques et altérer la stabilité

du système. Il faut alors masquer les interruptions et utiliser des verrous pour éviter ces problèmes. Les interruptions masquées sont ignorées, (sauf certaines qui ne peuvent pas l'être, en général dans des cas graves), et les périphériques associés les relancent périodiquement jusqu'à ce qu'elles soient traitées.

Quand le traiteur d'interruption est terminé, le processeur retourne à sa position empilée, et restore ses données privées pour continuer son exécution normale.

L'intérêt des interruptions se résume par le fait que le processeur ne se charge pas de savoir si on a besoin de lui mais qu'il peut être interrompu quand on le veut.

Les interruptions ne sont pas associées qu'aux entrées/sorties. Il existe par exemple dans les architectures Intel 256 interruptions, un grand nombre ne sont utilisées que pour la partie logicielle. Parmi elles, certaines peuvent être masquées. Elles sont gérées par un contrôleur d'interruptions programmable (PIC).

### 1.2.2 Le contrôleur d'interruptions

Sa tâche principale est d'associer une adresse de branchement à une routine ISR pour chaque interruption. De plus, quand une interruption surgi, toutes celles de plus basse priorité sont alors masquées et placées dans la file d'attente des interruptions retardées. Elles seront traitées à la fin de l'ISR courante, par ordre de priorité.



# Chapitre 2

## Le SMP et ses problèmes

Les architectures multi-processeurs sont composées de deux ou plusieurs CPU sur une même carte mère. Nous nous arrêterons à l'étude des SMP, le type de multi-processeurs le plus utilisé avec les systèmes UNIX. Il met en parallèle automatiquement différents processus faits à l'origine pour un système uniprocesseur, permettant la transparence au programmeur.

### 2.1 Le SMP

Un système SMP (Symmetric Multi-Processing) est un système multi-processeur dans lequel chaque processeur possède sa mémoire cache privée, où tous les processeurs sont interconnectés par un bus de données à la mémoire centrale et aux périphériques de manière symétrique, comme l'illustre le schéma ci-dessous :

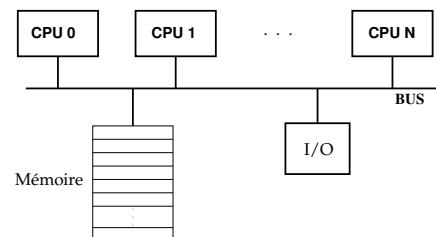


FIG. 2.1 – Illustration SMP

Lors de la conception d'un tel système le constructeur doit faire attention à l'encombrement du bus de données. Si la bande passante du bus est de 3 Gbps, et que le DMA nécessite 1,2 Gbps, il reste donc 1,8 Gbps pour les processeurs or si chaque processeur nécessite une bande passante de 400 Mbps pour effectuer ses transferts sans être ralenti, on ne pourra mettre que 4 processeurs dans le système si on veut profiter des bienfaits du parallelisme. Cependant grâce aux caches des processeurs, la bande passante nécessaire pour chaque processeur peut être diminuée donc suivant l'usage, on peut réévaluer le nombre de processeurs maximal du système (dans les limites de bande passante du bus), dans cet exemple probablement 5 voire 6 au lieu de 4.

La conception d'un système d'exploitation pour machine SMP n'est pas simple. L'ordonnanceur, la cohérence des caches et l'intégrité des données, le routage des interruptions font partie des problèmes à prendre en compte.

Une attention est aussi faite pour l'affinité processus-processeur. On veut éviter qu'un processus s'exécute parfois sur CPU0 et parfois sur CPU1, par exemple pour éviter de perdre le contenu des caches privés. Il faut aussi effectuer les opérations de changement de contexte telles que la sauvegarde des registres et de la pile d'exécution. Cependant pour la répartition de la charge il est nécessaire d'effectuer de telles opérations, appelées *migration* d'un processus.

## 2.2 Cohérence des caches

Commençons par définir la cohérence d'un cache : un cache est cohérent avec la mémoire principale lorsqu'il est impossible à un programme de lire des données périmées, qui existent dans une version plus récente, ailleurs dans le système. Ce problème de cohérence ne se pose que pour des données qui peuvent être modifiées ; ainsi puisque dans les architectures courantes les caches de niveau 1 sont composées de deux parties, une pour les instructions et l'autre pour les données, le problème ne sera pas soulevé dans la partie instruction, ces dernières n'étant accessible en lecture seule (hormis cas éventuel où un programme modifie lui-même son code), donc ne peuvent être incohérentes avec la mémoire centrale.

Voici deux exemples pour illustrer ces cas :

Supposons que nous ayons une variable en mémoire centrale de valeur 0. Au départ, lorsque le programme se lance, elle n'est dans aucun cache.

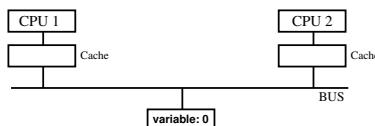


FIG. 2.2 – La variable n'est pas dans les caches

Voyons premièrement ce qui se passe avec un cache de type *write back* :

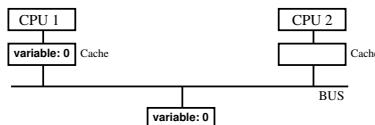


FIG. 2.3 – Le processeur 1 charge variable

Le processeur 1 charge la variable, il y a un *cache miss*. La variable est alors cachée par le processeur, sa valeur étant 0.

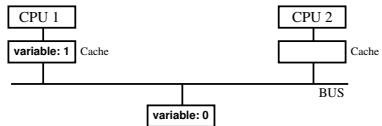


FIG. 2.4 – Le processeur 1 incrémente variable

Le processeur incrémente la variable. Sa valeur devient 1 et est enregistrée dans le cache puisqu'il y a un *cache hit*. La valeur en mémoire centrale reste 0 puisque nous sommes en *write back*.

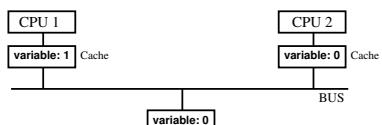


FIG. 2.5 – Le processeur 2 charge une valeur périmée

Lorsque le deuxième processeur veut charger la valeur, il a un *cache miss* et la prend en mémoire centrale. Mais cette valeur n'est pas à jour, puisque sa valeur en cache du processeur 1 est modifiée.

Deuxièmement, ce qui se passe avec un cache *write through* :

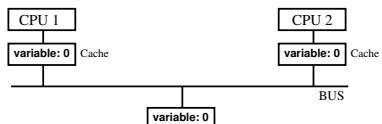


FIG. 2.6 – Les deux processeurs chargent la valeur

Supposons que les deux processeurs chargent la valeur (pas forcément simultanément). Elle sera alors dans les deux caches avec la valeur d'origine.

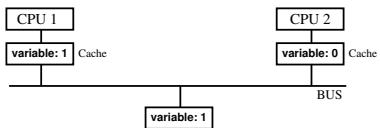


FIG. 2.7 – Un des deux incrémente la variable

Un des deux processeur incrémente la variable. Elle sera enregistrée à 1 dans son cache et aussi en mémoire puisque nous sommes en *write through*. Maintenant si l'autre processeur veut accéder à cette variable, il aura un *cache hit* mais sur une ancienne valeur.

On comprend bien qu'il faut trouver un moyen de synchroniser les caches entre eux pour que de telles incohérences disparaissent.

### 2.3 Problèmes des interruptions

Quand une interruption matérielle (IRQ) est déclenchée, lors d'un accès au disque par exemple, il est nécessaire de savoir quel processeur avait demandé la lecture pour l'avertir qu'elle a été effectuée. Tout étant connecté au bus système, si l'interruption est lancée dessus, tous les processeurs vont la recevoir, et toutes les tâches du système vont être interrompues. Ce n'est pas idéal pour un système SMP dont le but est de pouvoir faire plusieurs tâches en même temps...

Il faut donc trouver une solution pour router les interruptions, ou leur permettre de ne pas tout interrompre.

## Chapitre 3

# Les solutions apportées aux problèmes

### 3.1 Solutions pour les caches

Nous allons premièrement commenter les solutions logicielles de consistence de caches. Heureusement, des solutions matérielles existent pour faciliter le travail de l'OS, mais l'étude des solutions logicielles nous permet d'avoir une bonne approche de l'ensemble des solutions.

#### 3.1.1 Solutions logicielles

La première évidence qui apparaît est qu'un processus qui se termine doit valider le cache du processeur ou il s'exécutait et invalider la mémoire centrale et les caches des autres processeurs. Cependant, un processeur ne peut vider que son cache et doit alors avertir les autres qu'ils doivent en faire de même. C'est là qu'entre en jeu l'IPI : Inter- Processor Interrupts. C'est un moyen de communication entre les processeurs, qui leur permet de s'envoyer des interruptions entre eux de façon matérielle. Un processeur peut alors dire à tous les autres ou une partie d'effacer son cache invalide. L'implémentation de ce système de maintien de cohérence de caches sera étudiée dans les solutions hardware, on peut par exemple effectuer une optimisation qui consiste à ne pas vider le cache d'un processeur qui ne contient pas la zone mémoire invalide.

Le problème principal avec les consistences de caches arrive lorsque plusieurs processeurs peuvent accéder et modifier des données partagées. Il y a deux manières principales pour gérer ces données : les opérations non cachées et les vidages sélectifs de données partagées.

#### Opérations non cachées pour les données partagées

Les opérations non cachées sont rendues possibles par la plupart des architectures, sur des pages de données sélectionnées. Avec cette approche, on peut résoudre le problème en interdisant les processeurs de mettre en cache les données partagées, et ce à tous niveaux.

Dans un système où le noyau est multi-threadé, ses structures partagées telles que la table des processus, les sémaphores et les verrous qui les protègent, ne doivent pas être cachées pour pouvoir être utilisées par tous les processus. Les zones de mémoires partagées pour les processus utilisateurs ne doivent pas non plus être cachées.

C'est une solution plutôt simple mais elle ralentit considérablement le système quand il commence à y avoir beaucoup de processeurs ou processus qui veulent accéder aux mêmes données puisque les caches ne sont plus utilisés.

### Vidages sélectifs de caches

Les vidages sélectifs de caches (selective cache flushing) sont plus complexes à réaliser que les opérations non cachées mais sont aussi beaucoup plus efficaces. Un processeur peut utiliser son cache pour les données partagées, mais doit avertir qu'il le fait en positionnant un verrou sur ces données. Les caches des autres processeurs seront alors invalidés par le système, la mémoire centrale validée. Ainsi un autre processeur ne pourra pas être autorisé par le système à lire ou écrire dans ces données **en mémoire principale** tant que le verrou ne sera pas enlevé.

Quand un processeur enlève le verrou d'une zone, il doit explicitement mettre à jour la mémoire centrale avec ce qui se trouve dans son cache (valider la mémoire centrale et invalider son cache). Grâce à cette invalidation, au prochain accès à cette zone mémoire, on sera assuré d'avoir un cache miss et donc le cache sera mis à jour avec une valeur correcte. Bien sûr la structure de données du verrou ne peut être cachée puisque c'est la base du système.

### Problèmes des lignes de caches

En plus de cela, les ressources critiques ne doivent pas être dans la même ligne de cache. Sinon il y aurait un risque d'accéder à des données figées comme on peut le voir avec ces illustrations :

Supposons que nous avons des lignes qui peuvent contenir deux compteurs et qu'ils valent 0.

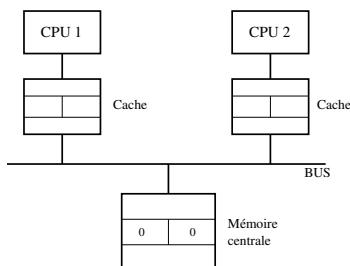


FIG. 3.1 – Etat initial

Si un processeur veut utiliser le premier, il charge les deux dans son cache ; Il incrémentera le premier, on se retrouve donc comme ceci :

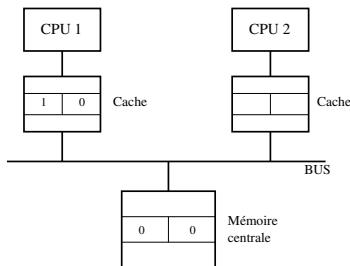


FIG. 3.2 – Le premier incrémente la première valeur

Lorsque le deuxième processeur veut modifier le deuxième compteur, il charge les deux valeurs aussi dans son cache et modifie la deuxième.

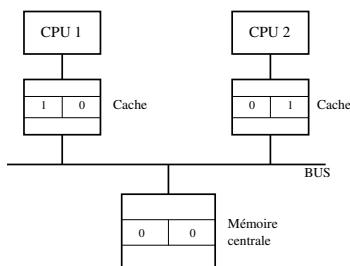


FIG. 3.3 – Le deuxième incrémente la deuxième valeur

Maintenant on se rend bien compte que cela pose un problème. Quand les processeurs écriront leurs caches dans la mémoire principale pour la valider, à un moment ou un autre, le premier des deux qui le fait verra ses données écrasées par l'écriture du deuxième. On peut noter aussi que ce genre de problème peut arriver non seulement avec des variables en mémoire, mais aussi avec les tampons de données des entrées/sorties que le DMA a placé.

Les mémoires caches des processeurs étant vidées à chaque migration de processus d'un cpu à un autre, et ne pouvant mettre en mémoire cache les verrous, les avantages de la mémoire cache sont réduits par rapport à un système uniprocesseur. De plus les solutions logicielles demandent beaucoup de travail pour les programmeurs de systèmes d'exploitation et prennent du temps, elles sont donc rarement utilisées telles quelles. Il est possible d'outre-passé ces problèmes en utilisant des solutions matérielles, fournies par les constructeurs. C'est ce que nous voyons dans la section suivante.

### 3.1.2 Solutions matérielles

L'idée principale des solutions matérielles est l'utilisation de l'espionnage de bus (bus watching). Comme tout est connecté sur le même bus, lorsqu'un processeur ou un périphérique écrit dans la mémoire, les autres peuvent le voir et donc savoir que la valeur présente à une adresse a changé.

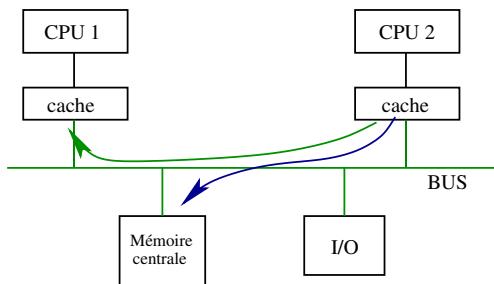


FIG. 3.4 – Exemple de bus watching

Dans cet exemple le processeur 2 écrit des données en mémoire centrale. Le processeur 1 voit alors la transaction et met à jour ces données.

Il y a deux politiques de mise à jour des données par espionnage de bus : Le *write invalidate* est proche des solutions logicielles. Lorsqu'un contrôleur de cache observe une transaction à propos d'une adresse qu'il contient, il marque cette donnée invalide, ce qui provoquera un cache miss à la prochaine requête de ce processeur.

Avec le *write update*, la ligne en cache est directement modifiée, ainsi le cache est toujours valide.

Grace à ces techniques, la gestion des cache est efficace et transparente aux systèmes d'exploitation.

## 3.2 Solutions pour les interruptions

La solution actuelle a été introduite par Intel dans la **Multiprocessor specification**. C'est une solution matérielle donc elle est rapide et ne dépend pas du système d'exploitation.

Chaque processeur intègre un contrôleur d'interruptions programmable avancé appelé *local APIC*. Il y a au moins un *I/O APIC* dans le système, qui lance les interruptions des périphériques.

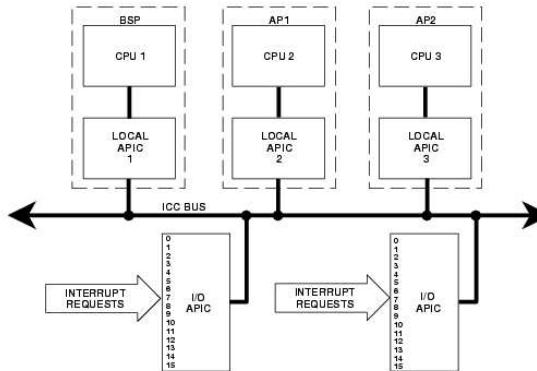


FIG. 3.5 – local APIC et I/O APIC

De plus, un nouveau bus a été introduit, pour contenir tous les transferts d'interruptions. C'est le bus ICC. Il vient en complément du bus mémoire qui a besoin d'une grande bande passante, donc en substituer les interruptions ne peut être qu'un bénéfice. Les processeurs, les mémoires et périphériques sont alors tous connectés par deux bus comme ceci :

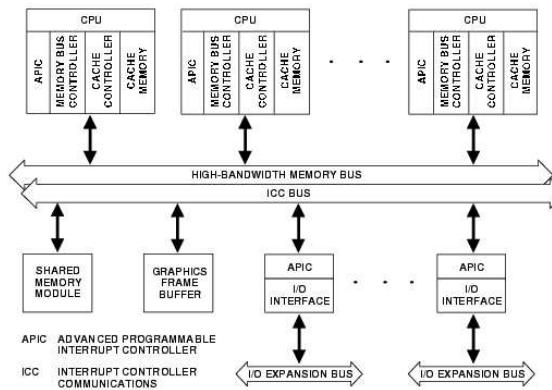


FIG. 3.6 – Connexion des éléments du système

Si un processus veut faire une lecture sur le disque, Il fait l'appel système correspondant au read, le processus passe en mode noyau pour gérer cet appel. Il positionne alors les verrous nécessaires aux protections des données, et le *local APIC* du processeur est reprogrammé pour recevoir l'interruption du disque dur. Le processus est alors passé en état d'attente ou

sommeil. Quand l'appel système est terminé, l'ordonnanceur donne le processeur à un autre processus (si il y en a), et le système reprend son activité normale jusqu'à ce que la lecture du disque soit complète. L'*I/O APIC* du disque envoie l'information sur le bus ICC, le *local APIC* du processeur programmé traite l'interruption et est déprogrammé par le noyau pour ne plus recevoir ces interruptions. Le processus qui avait fait la demande de lecture repasse en état éligible et est remplacé en tête de la file d'attente de processus de sa priorité, et tout redevient comme avant.

## Conclusion

Les constructeurs et les programmeurs ont trouvé différentes solutions aux problèmes du SMP. Cependant l'un sans l'autre n'arrive à un résultat satisfaisant : les solutions logicielles sont trop lentes, allant à l'encontre du but recherché par le SMP, tandis que les solutions matérielles ne peuvent pas tout gérer.

Les problèmes de cohérence de caches sont résolus par cette interaction entre le logiciel et le matériel. Dans certains cas, on choisira de ne pas mettre en mémoire cache certaines données critiques comme les verrous. Autrement, on les mettra mais on veillera à ce que l'information de modification arrive à tous les autres processeurs.

Pour les problèmes des interruptions, les constructeurs ont introduit un nouveau bus spécifique. Ce bus permet de relier tous les contrôleurs d'interruptions entre eux et implémente un nouveau protocole de gestion d'interruptions amélioré.

Aujourd'hui les systèmes multi-processeurs occupent une place considérable, à la fois dans les serveurs mais de plus en plus dans les ordinateurs de bureau (exemple Apple avec les bi-G4 et bi-G5). Leurs performances et leur efficacité sont sans cesse améliorées par un gros travail des programmeurs système et des fabricants. Ils remplaceront sans doute les systèmes mono-processeur qui arriveront bientôt à leurs performances maximales en terme de vitesse si on ne trouve pas l'alternative en informatique quantique ou génétique.



## Bibliographie

- [1] David Decotigny. Point de reflexion sur l'architecture du coeur. *draft-0.pdf*, 2004. [http://kos.enix.org/~d2/snapshots/kos-doc\\_current/draft-0.pdf](http://kos.enix.org/~d2/snapshots/kos-doc_current/draft-0.pdf).
- [2] Daniel Etiemble. Multiprocesseurs symétriques. *Calpar-SMP.pdf*, 2003. <http://www.lri.fr/~de/Calpar-SMP.pdf>.
- [3] Inconnu. Using interrupts. *CP\_interrupt.pdf*, 1997. [http://kos.enix.org/pub/CP\\_interrupt.pdf.gz](http://kos.enix.org/pub/CP_interrupt.pdf.gz).
- [4] Intel. Multiprocessor specification. *24201606.pdf*, 1997. <http://developer.intel.com/design/pentium/datashts/24201606.pdf>.
- [5] Curt Schimmel. *UNIX Systems for Modern Architectures*. Addison-Wesley Professional Computing Series, 1994.
- [6] Beck Böhme Dziadzka Kunitz Magnus Verworner. *Linux Kernel internal*. Addison-Wesley, 1997.