Parallélisme

SMC - MU5IN162

C1 - Archi

Objectif du cours et Support

- **Objectif** du cours : **S**ystem **M**any **C**ore
  - Etude des problèmes des "Systèmes Many Core"
    Plus précisément des Processeurs Many Core
    du point de vue matériel (architecture) et logiciel (OS)
- **Support** des TME
  - Architecture
    - **SoClib** pour la modélisation et la validation des composants
    - **TSAR** pour le processeur, c'est un manycore conçu au LIP6 dont l'architecte principal était Alain Greiner
  - Logiciel
    - **GIET** pour commencer, un exécutif statique
    - **ALMOS-MKH** un système d'exploitation conçu au LIP6 dont l'objectif est d'éliminer les contraintes d'accès aux structures de données du noyau qui limitent le parallélisme théorique offert par l'architecture (Alain Greiner et al.)

Plan de la séance

- Architecture manycore
- Types de parallélisme
- Limite du parallélisme, loi d'Amdahl
- Parallélisme de tâches
- Modèle de communication
- Infrastructure matérielle
- TME

Architecture Manycore

- C'est une classe des architectures MPSOC

  MPSOC Multi Processor System On Chip

  Architecture multiprocessor intégrée

- 2 grandes classes :
  - "Application Spécific" MPSOC
  - Generic MPSOC

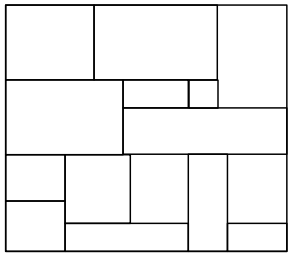
Application Specific MPSoC

- C'est du codesign Matériel - Logiciel
- Un type d'ASIC : Application Specific Integrated Circuit
  - Spécification initiale pour une classe d'applications (traitement d'images, radio logicielle, etc.)
  - Co-développement du matériel et du logiciel
- Architecture très hétérogène parce que ciblée
- Prototypage virtuel de l'architecture avec des langages tel que System-C mais souvent avec une abstraction TLM (Transaction Level Modeling) pour le dév. du logiciel
  ⇒ communication obligatoire entre les équipes.
- Langage et environnement de dev ad hoc

Application Specific MPSoC

Problèmes

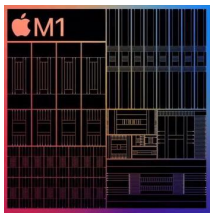
- Coût de développement gigantesques
- Solution peu réutilisable sauf au niveau « IP component »
  (IP component : fonction spécifique intégrée RAM, CPU, etc.)
- ST Microelectronics fait ce type de SoC
  (par ex. des imageurs) ⇒ Beaucoup
  de divisions dans des domaines très précis
- Les coûts augmentent avec la taille
- Ce sont des marchés de niche

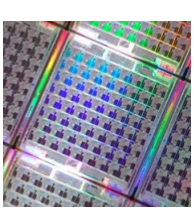


Le cours ne porte pas sur cette classe de MPSOC

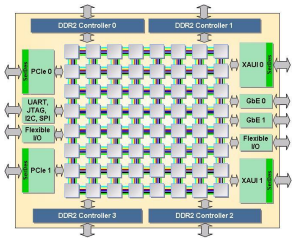
Generic MPSoC

- Distinction franche entre le matériel et le logiciel
- Solution plus homogène et plus régulière
  donc plus simple à réutiliser ou à faire évoluer
- Adapté à une grande classe d'application
  ⇒ diminution des coûts de développement
- C'est le modèle d'Intel
  mais pas seulement

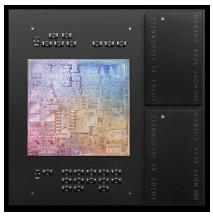




Tilera 100



wikipédia : Tilera 64



Apple M1

C'est cette classe qui nous intéresse

TILERA fait des manycores
APPLE pas encore :-)

Type de Parallélisme

Parallélisme d'instructions versus Parallélisme de tâches

- Le parallélisme d'instructions (ILP instruction Level Parallelism) c'est l'exécution de plusieurs instructions en parallèle d'un programme séquentiel.
  - Parallélisation automatique par le matériel à la volée
  - Parallélisation du programmeur ou du compilateur pour des algorithmes spécifiques (traitement du signal ou d'images)
- Le parallélisme de tâches c'est l'exécution concurrente de programmes communicants (ou non) sur le même MPSOC et la même mémoire (même espace d'adressage) (ou pas).
  - Parallélisation auto. guidée par le programmeur (OpenMP)
  - Ré-écriture complète des programmes (Pthread / MPI)

Parallélisme d'instructions

Intel a beaucoup exploité le parallélisme d'instructions

- Exécution superscalaire
  Exécution en // de plusieurs instructions parmi un sous-ensemble
- Exécution dans le désordre
  Avec renommage des registres pour éliminer les fausses dépendances (utilisations du même registre par 2 instructions indépendantes).
- Exécution spéculative
  Exécution en parallèle de plusieurs séquences d'instructions
  (if *cond - then - else* : cond, then et else sont exécutées en // et le processeur garde la séquence *then* ou *else* suivant le résultat de *cond*)
- Prédiction de branchement
  Le processeur fait des hypothèses sur les conditions de sauts

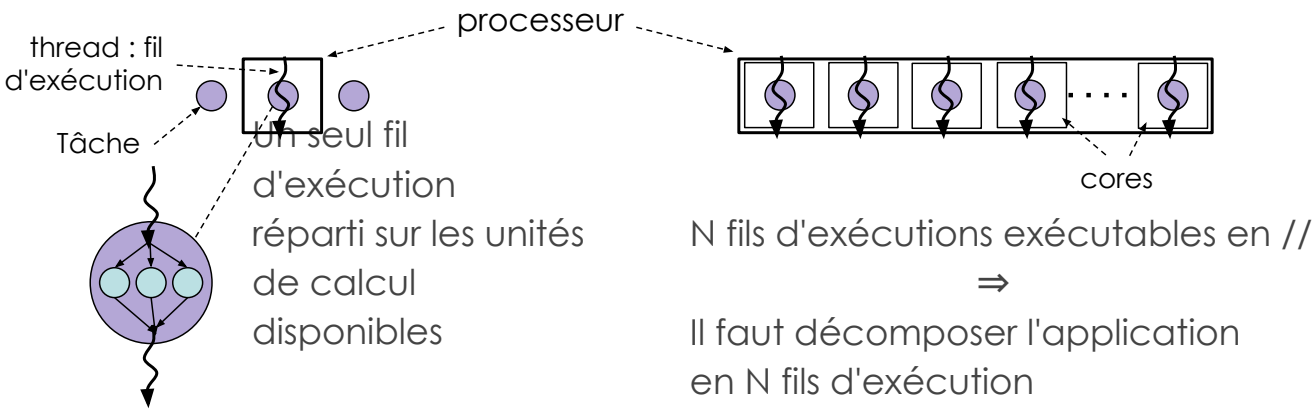
Impasse de l'ILP

Les innovations en ILP imposent une augmentation de l'énergie consommée pour des gains en performance de quelques %.

- Au delà de 100 W/cm^2 le refroidissement à l'air est impossible
- En 2005, Intel renonce à sortir son Pentium à 4GHz
  parce qu'il consomme trop, c'est le début des multicores
- Il faut des processeurs plus simples mais plus nombreux
  - L'idée est de simplifier l'architecture pour garder ce qui fait gagner de la vitesse sans trop coûter en énergie (superscalaire in-order ou prédiction de branchements) et d'augmenter le nombre de cores.

La performance doit être extraite du parallélisme à gros grain, ce qui reporte le problème de speedup sur les programmeurs

Nécessité de paralléliser les applications



Un seul fil d'exécution réparti sur les unités de calcul disponibles

N fils d'exécutions exécutables en //

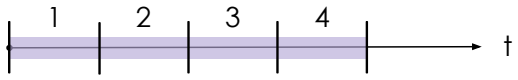
⇒

Il faut décomposer l'application en N fils d'exécution

Optimisation est mesurée en Instruction / Cycle (IPC)

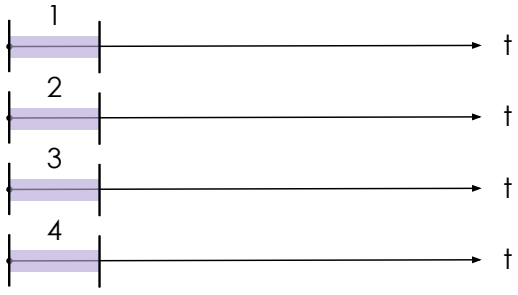
Optimisation est mesurée en Instruction / mWatt

Limite du parallélisme, loi d'Amdahl

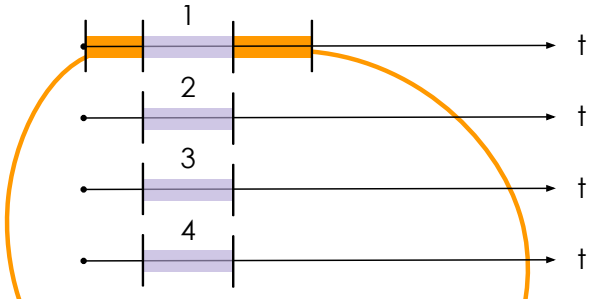


Un programme séquentiel

Parallélisation en rêve



Parallélisation plus réaliste

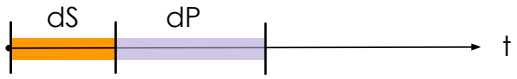


Il faut préparer le travail et collecter les resultats

Limite du parallélisme, loi d'Amdahl

accélération du prog. sur n cores = speedup (n) = $\frac{\text{Durée (1)}}{\text{Durée (n)}}$

Le but est que Speedup (n) = n mais en vrai speedup (n) < n



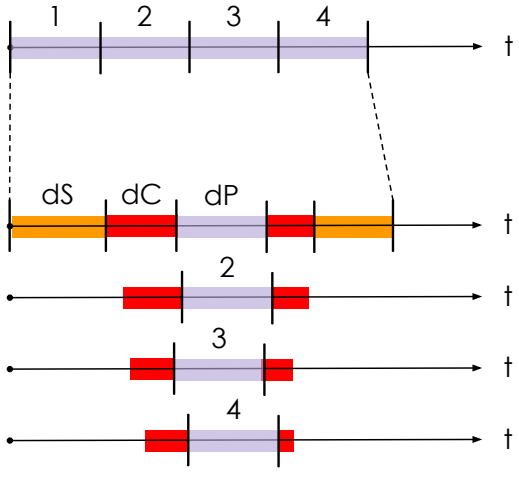
$$\text{speedup (n)} = \frac{dS + dP}{dS + \frac{dP}{n}} = \frac{1}{1 - p(1 - \frac{1}{n})}$$

$$\text{avec } p = \frac{dP}{dS + dP}$$

p : pourcentage de parallélisme

si p = 0,5 ⇒ speedup (∞) = 2

Limite du parallélisme, loi d'Amdahl



dC correspond à la durée de communications et de synchronisation entre tâches

$$\text{speedup (n)} = \frac{dS + dP}{dS + \frac{dP}{n} + dC}$$

Le temps d'exécution en parallèle peut être plus long qu'en séquentiel !

Type de parallélisme

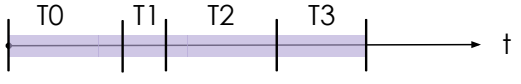
Petite analogie , pour ramasser du raisin

- Travail à la chaîne → pipeline logiciel
  - Plusieurs personnes, chacun fait une partie différente du travail sur un « objet » qu'il reçoit de son prédécesseur
- Cueillette à plusieurs → « Task farm »
  - Plusieurs personnes, chacun s'occupent d'une partie du champ
- Vendange → modèle hybride
  - Plusieurs personnes, certains donnent des paniers, d'autres cueillent et passent les paniers pleins à d'autres qui les vident…

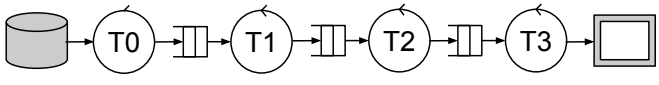
Pipeline logiciel

Parallélisme à gros grain, généralement asynchrone

- Les applications sont décomposées en tâches successives
- L'ordre de traitement des données est important
- La durée des tâches est inégale et dépend des données



On suppose 4 cores C0, C, C2, C3

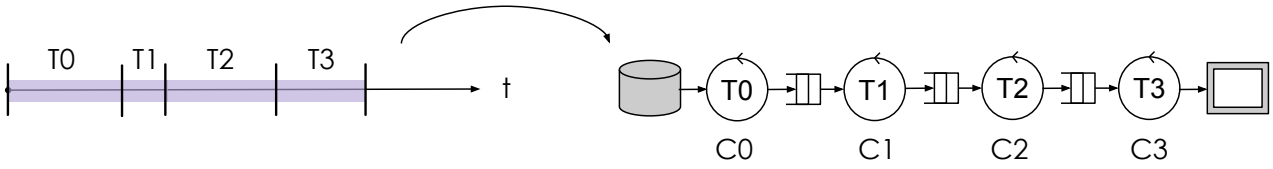


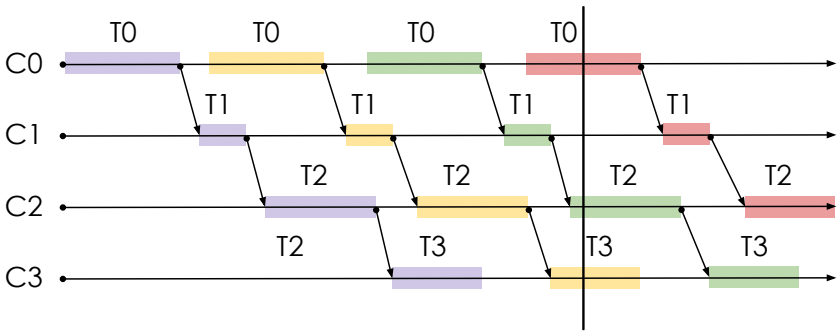
Le pipeline est représenté par un graphe bipartite :

- Les ronds sont les tâches
- les carrés sont les canaux

Adapté par exemple au traitement d'un flux d'images

Pipeline logiciel asynchrone

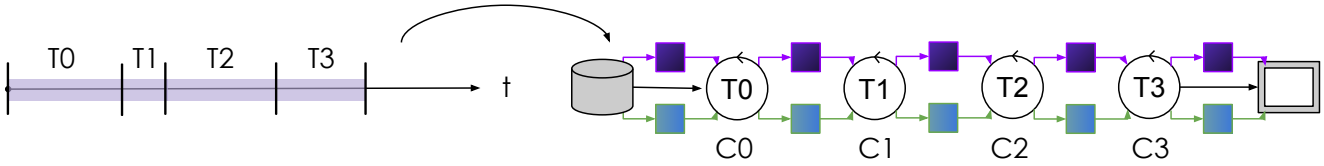


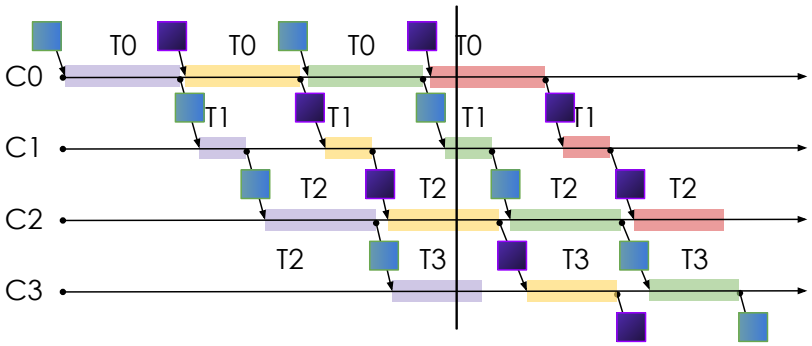


Les FIFOs permettent de s'adapter aux différences de débits et à la durée des tâches

Plusieurs tâches s'exécutent en parallèle sur des lots de données différents

Pipeline logiciel asynchrone





Pour chaque tâches :
on a 1 BUFIN et 1 BUFOUT

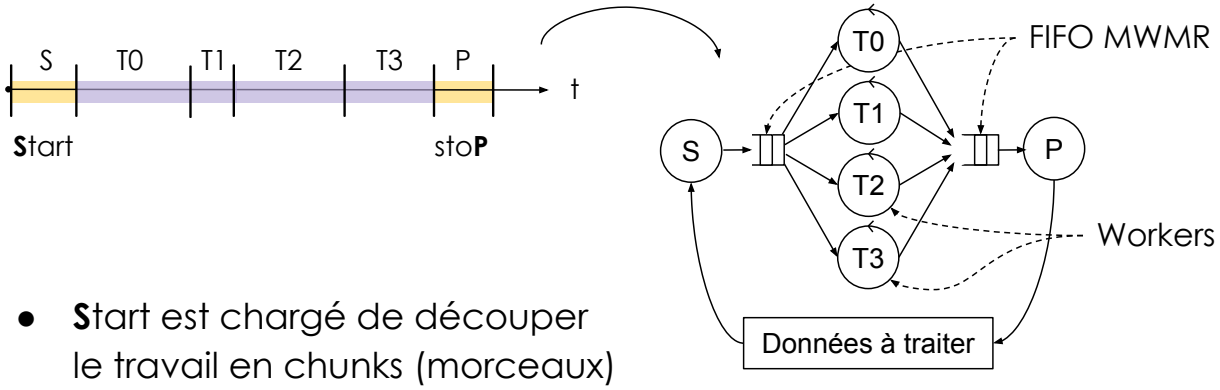
BUFIN → Tâche → BUFOUT

Les tâches ont deux conditions pour démarrer

- BUFIN Plein
- BUFOUT vide

Utilisation de buffers utilisés en alternance
À un instant T un buffer est soit lu, soit écrit

Task Farm

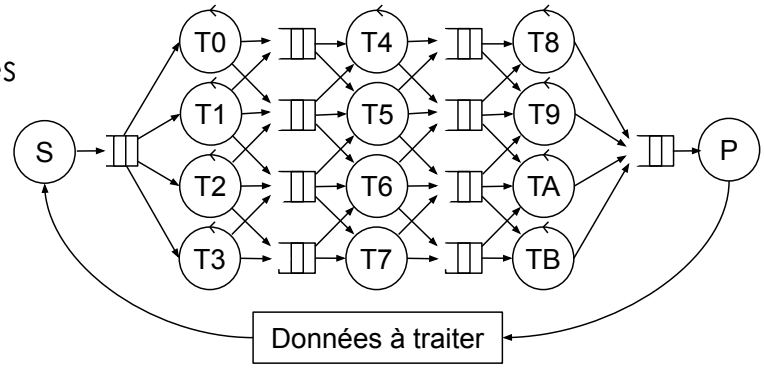


- **S**tart est chargé de découper le travail en chunks (morceaux)
- Chaque Worker prend un chunk, le traite et poste son résultat.
- sto**P** rassemble les chunks pas forcément dans l'ordre !

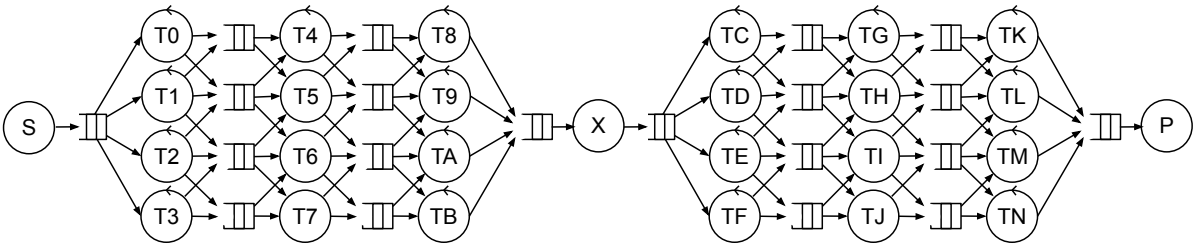
Adapté par exemple au filtrage des paquets réseau

Modèle hybride

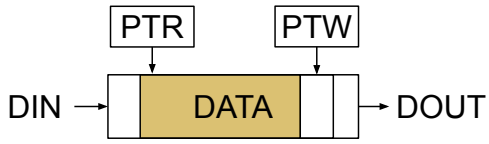
une ferme de pipelines



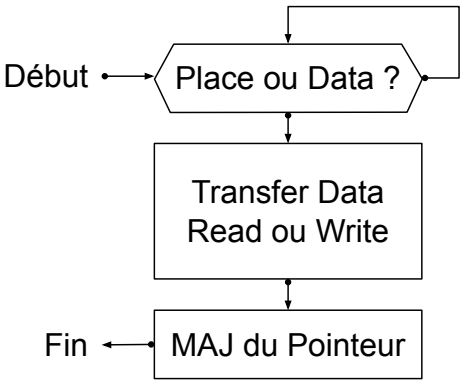
Ou un pipeline de ferme de pipelines !



Protocole FIFO simple



Protocole en 3 phases sans lock



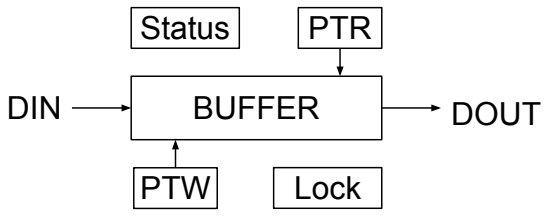

```
int fifoempty (fifo_t * f) {
  return (f→SIZE + f→PTR - f→PTW) % f→SIZE;
}
int fifofull (fifo_t * f) {
  return (f→SIZE + f→PTW - f→PTR) % f→SIZE;
}
fifowrite (fifo_t * f, buf_t * buf, size_t n) {
  if (fifoempty (f) > n) {
    memcpy (f→BUF, buf, n);
    f→PTW = (f→PTW + n) % f→SIZE;
  }
}
fiforead (fifo_t * f, buf_t * buf, size_t n) {
  if (fifofull(f) >= n) {
    memcpy (buf, f→BUF, n);
    f→PTR = (f→PTR + n) % f→SIZE;
  }
}
```

fifo vide PTR == PTW

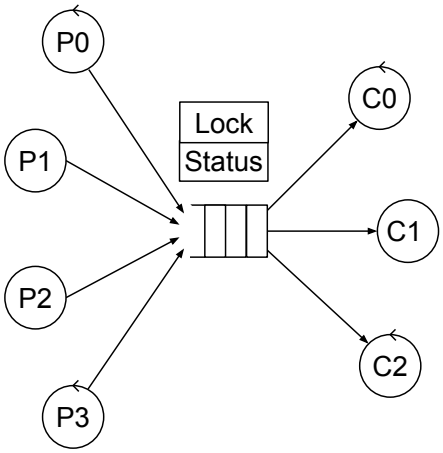
fifo pleine PTW + n == PTR modulo SIZE

Canaux de communication MWMR

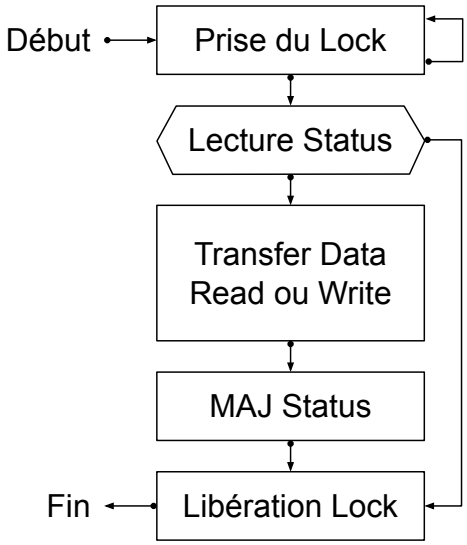
- Se comporte comme une FIFO :
  Production et consultation simultané
- Accessible indifféremment par une tâche logiciel ou une tâche matérielle (c'est-à-dire un automate FSM)
- Supporte les deux modes
  - Multiples producteurs / Multiples consommateurs
  - Simple producteur / Simple consommateur
- Implémente un tampon de mémoire partagée protégé par un verrou



Canaux de communication MWMR



Protocole en 5 Phases



Modèle de communication

Communications Read/Write vs Send/Receive

- Sur une machine sans espace d'adressage partagé :
  - Communication send-receive
    MPI (Message Passing Interface)

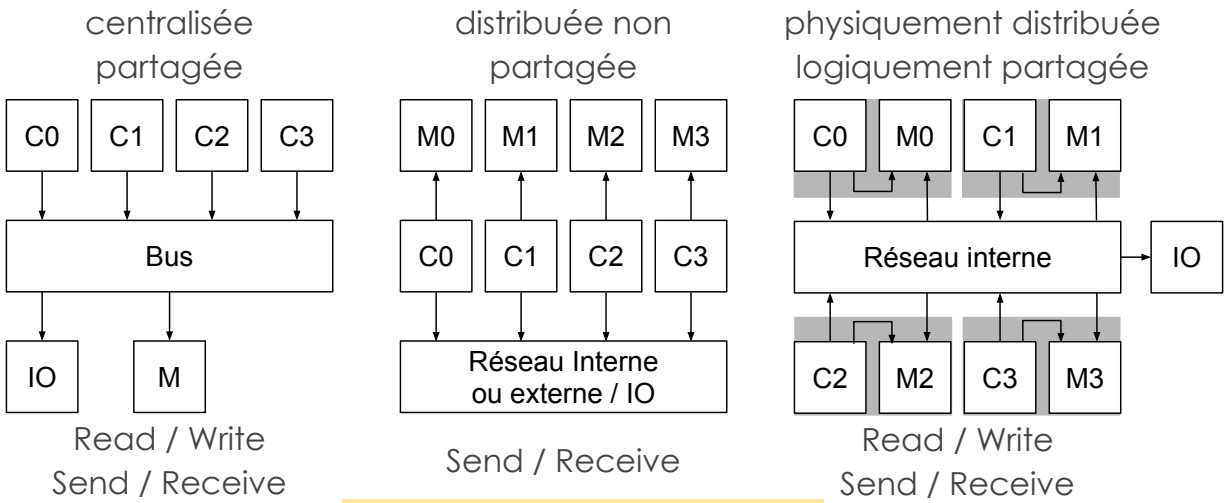
```
send ( canal_id, buffer, size)
receive ( canal_id, buffer size)
```

- Sur une machine avec espace d'adressage partagé :
  - Communication par lecture / écriture directe
    POSIX Threads

Les manycores généralistes (qui peuvent exécuter un OS) sont adaptés aux deux modèles, mais les communications par canaux sont plus simples à concevoir et valider !

Infrastructure matérielle

3 modèles de communications matérielles



en TME

- Prise en main du langage de prototypage SystemCASS (langage SystemC avec une structure imposée pour la description des automates)
- Description d'automates d'états finis communicants par un protocole FIFO 1 → 1
- Simulation du circuit (topcell)
- Vous devez écrire un compte rendu au format markdown avec la réponse aux questions et surtout vous « réflexions » pendant la conception (pour mémoriser les difficultés rencontrées).