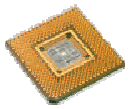
La machine parallèle MPC1

Hardware, protocoles et performances

Université P. & M. Curie (PARIS)

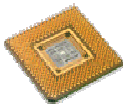
Laboratoire d'Informatique de PARIS6

Olivier Glück



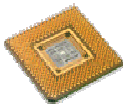
Plan

- Introduction
- Architecture matérielle
- La couche de communication bas-niveau (PUT)
- MPI sur la machine MPC1
  - Problématique
  - Les messages internes à MPI
  - Les primitives MPI
  - Premières performances
  - PUT en mode utilisateur
  - Redistribution de la mémoire
- Conclusion

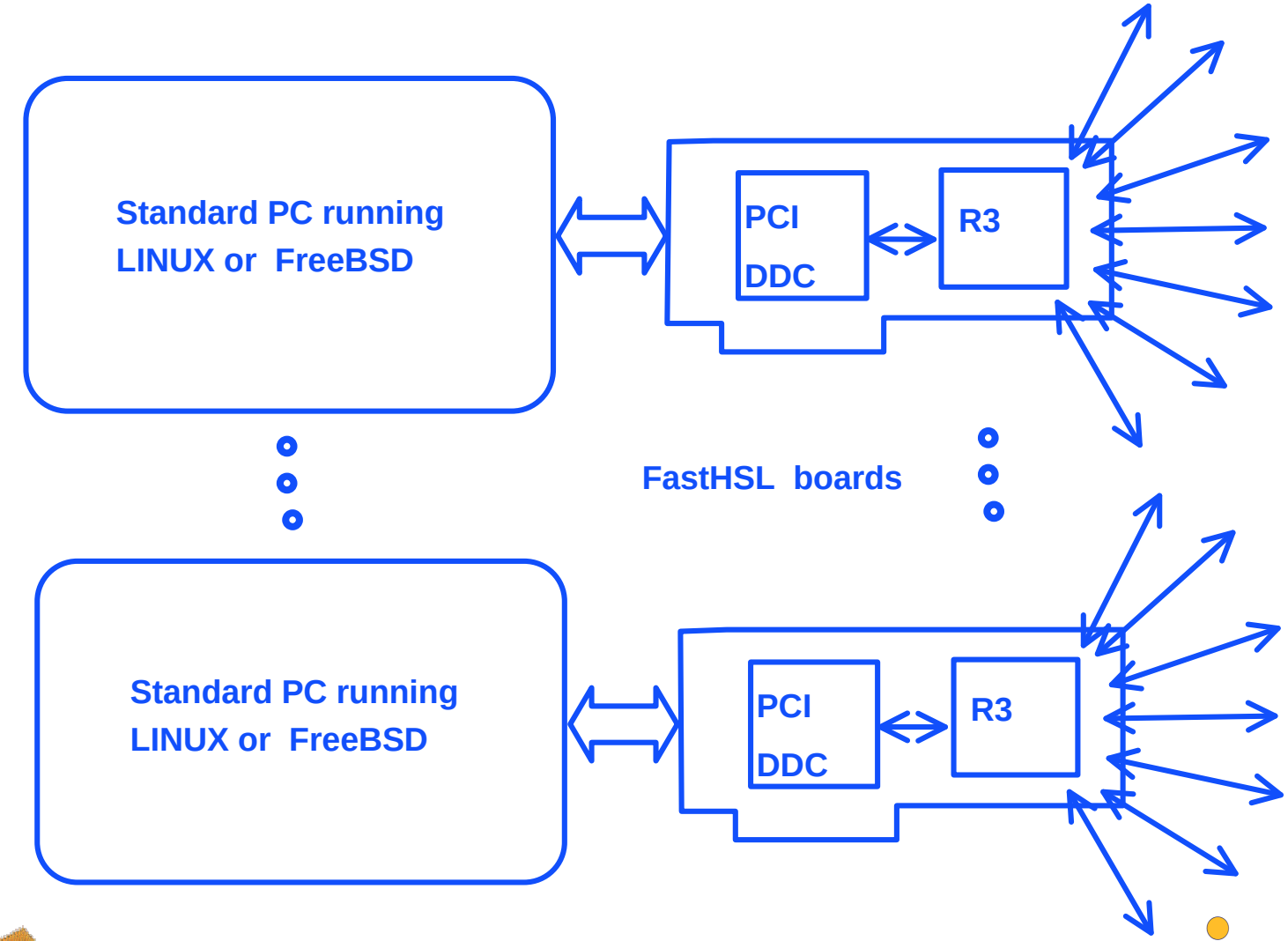


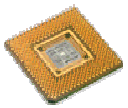
Introduction

- Une carte réseau PCI (FastHSL) développée au LIP6 :
  - Liens série, point à point (HSL,1 Gbit/s)
  - RCUBE : routeur (8x8 crossbar, 8 HSL ports)
  - PCIDDC : contrôleur réseau PCI (protocole de communication de type « Remote DMA »)
- But : fournir des couches logicielles les plus performantes au niveau applicatif (MPI)

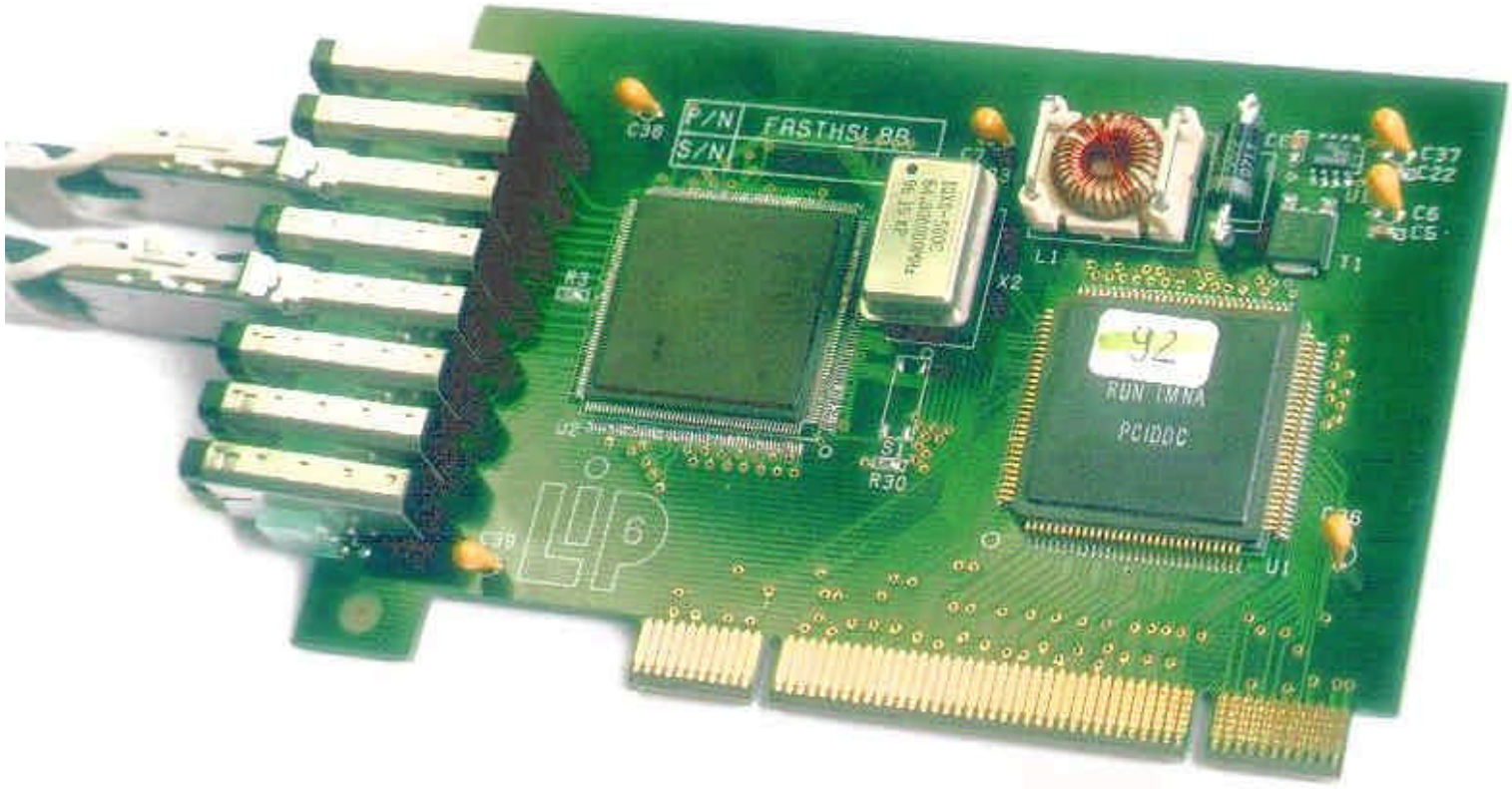


Architecture matérielle



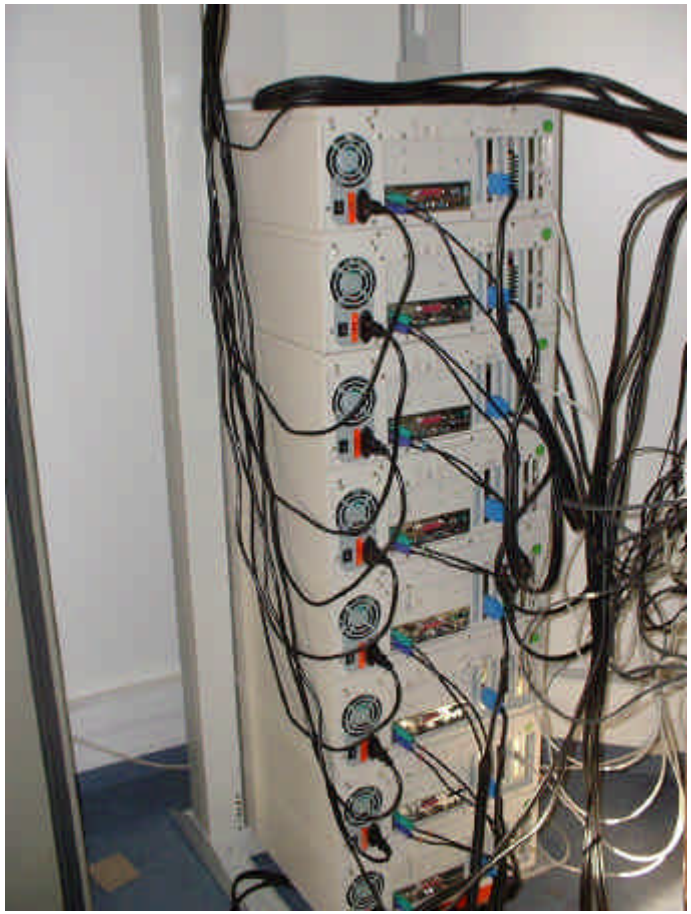


La carte FastHSL

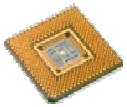


MPC avant

MPC maintenant



8 nœuds bi-pro 1GHz, 8 Go de RAM, 160 Go de disk



Le matériel

Le lien HSL (1 Gbit/s)

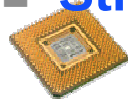
- Câble coaxial, lien série, point à point, full-duplex
- Les données sont encodées sur 12 bits
- Contrôle de flux matériel

RCUBE

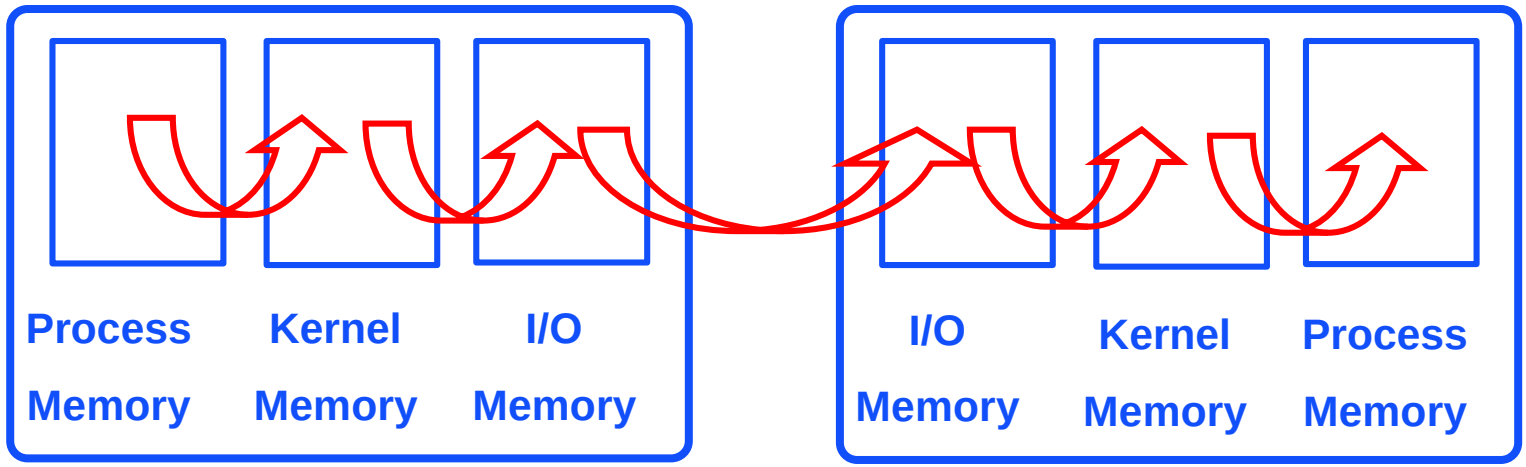
- Routeur rapide, reconfigurable et à grande extensibilité
- Latence : 150 ns
- Routage wormhole

PCIDDC

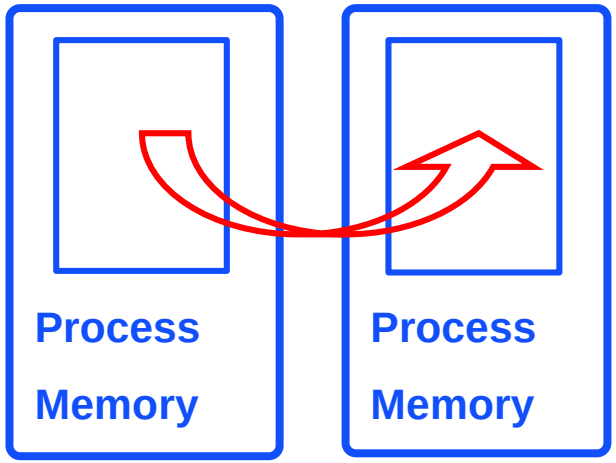
- Le contrôleur réseau, interface avec le bus PCI
- Réalise le protocole de communication : Remote DMA
- Stratégie zéro-copie

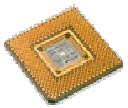


Le protocole de communication bas-niveau

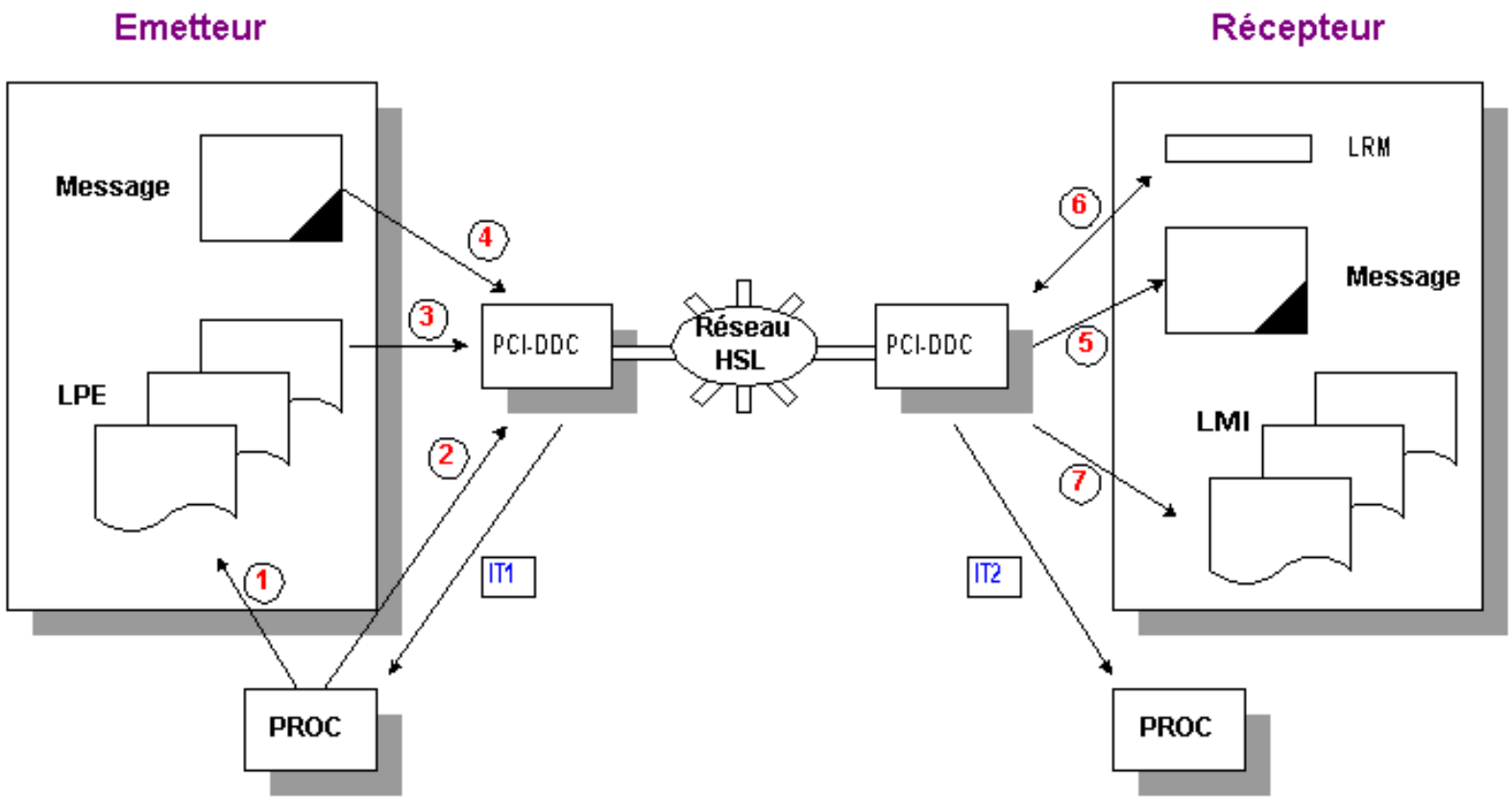


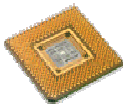
- Protocole zéro-copie (dépôt direct dans la mémoire du destinataire)
- Accès direct à la mémoire du nœud hôte





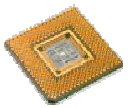
L'écriture distante





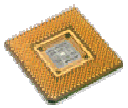
Plan

- Introduction
- Architecture matérielle
- La couche de communication bas-niveau (PUT)
- MPI sur la machine MPC1
  - Problématique
  - Les messages internes à MPI
  - Les primitives MPI
  - Premières performances
  - PUT en mode utilisateur
  - Redistribution de la mémoire
- Conclusion



PUT : la couche de communication bas-niveau

- Supportée par les Unix standards : FreeBSD / Linux
- Stratégie zéro-copie
- Fournit une API noyau simple utilisant l'écriture distante de PCIDDC
- Paramètres d'un appel à PUT() :
  - numéro de nœud distant
  - adresse physique locale
  - adresse physique distante
  - taille des données à émettre
  - un identifiant de message
  - fonctions de callback pour la signalisation



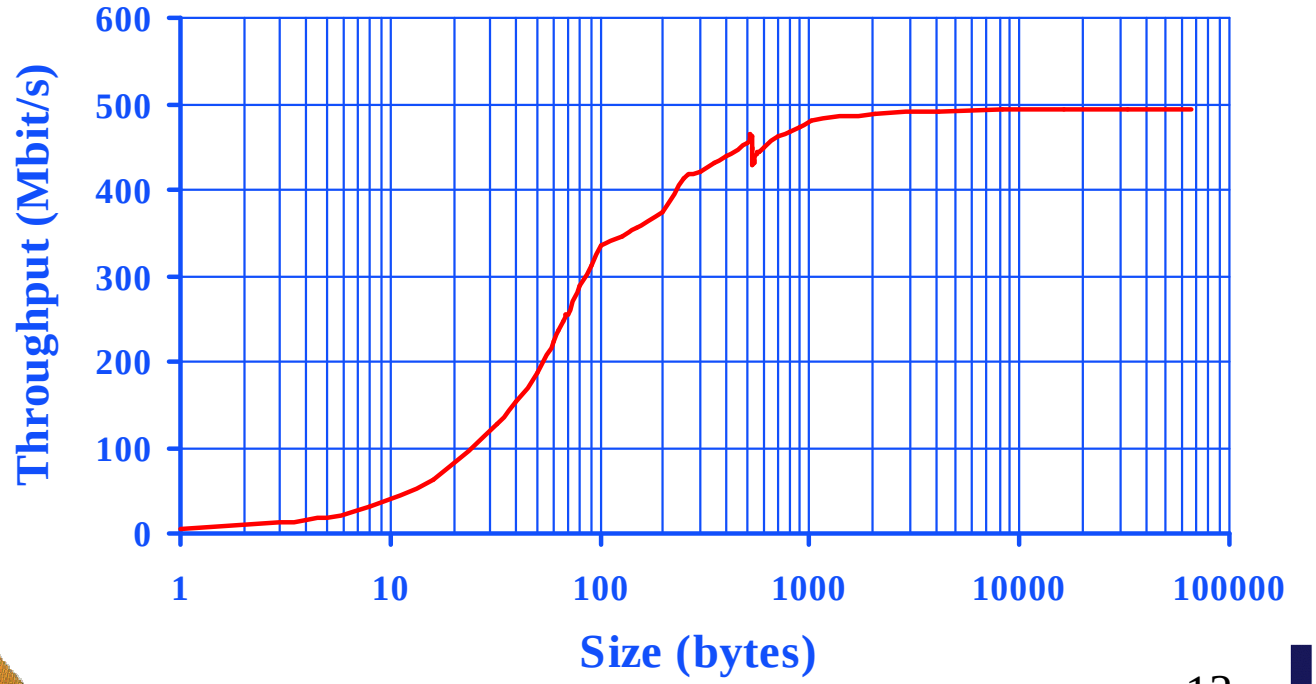
Performances de PUT

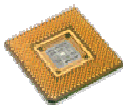
PC Pentium II 350MHz

Débit : 494 Mbit/s

Demi-débit : 66 octets

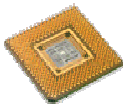
Latence : 4 µs (sans appel système, sans interruption)



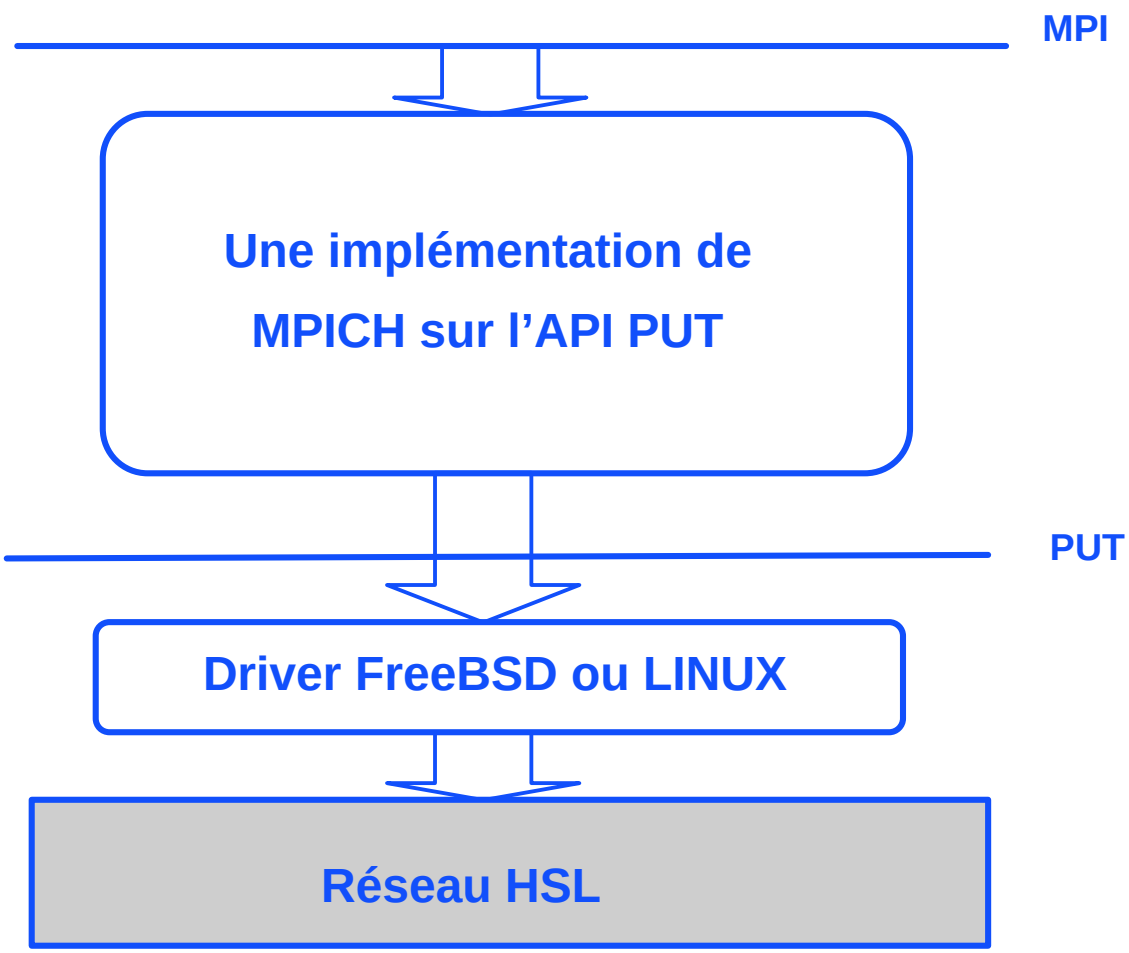


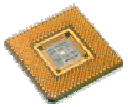
Plan

- Introduction
- Architecture matérielle
- La couche de communication bas-niveau (PUT)
- MPI sur la machine MPC1
  - Problématique
  - Les messages internes à MPI
  - Les primitives MPI
  - Premières performances
  - PUT en mode utilisateur
  - Redistribution de la mémoire
- Conclusion

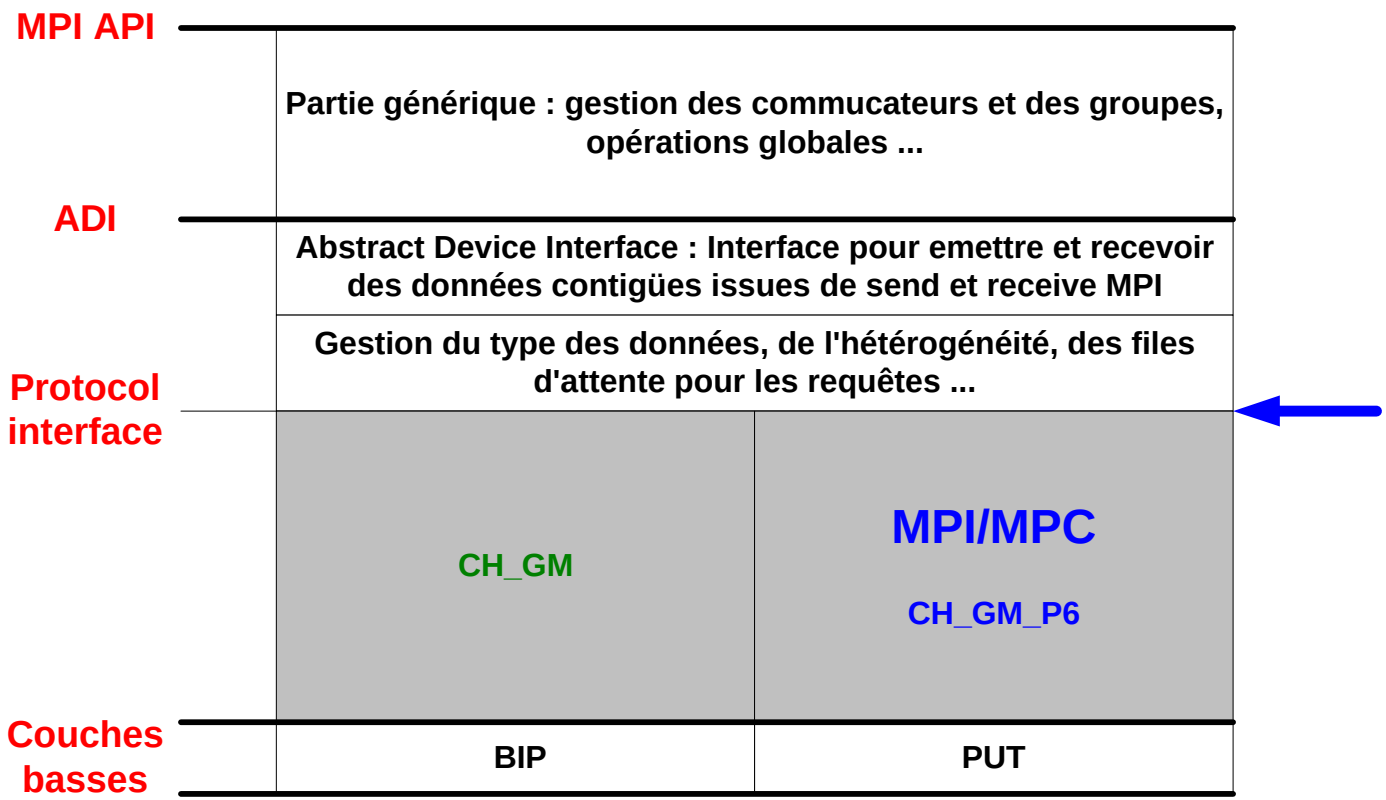


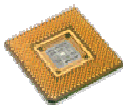
MPI sur MPC (1)


MPI
Une implémentation de
MPICH sur l'API PUT
PUT
Driver FreeBSD ou LINUX
Réseau HSL



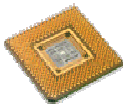
MPI sur MPC (2)


MPI API
Partie générique : gestion des commucateurs et des groupes, opérations globales ...
ADI
Abstract Device Interface : Interface pour emettre et recevoir des données contigües issues de send et receive MPI
Gestion du type des données, de l'hétérogénéité, des files d'attente pour les requêtes ...
Protocol interface
CH_GM
MPI/MPC
CH_GM_P6
Couches basses
BIP
PUT

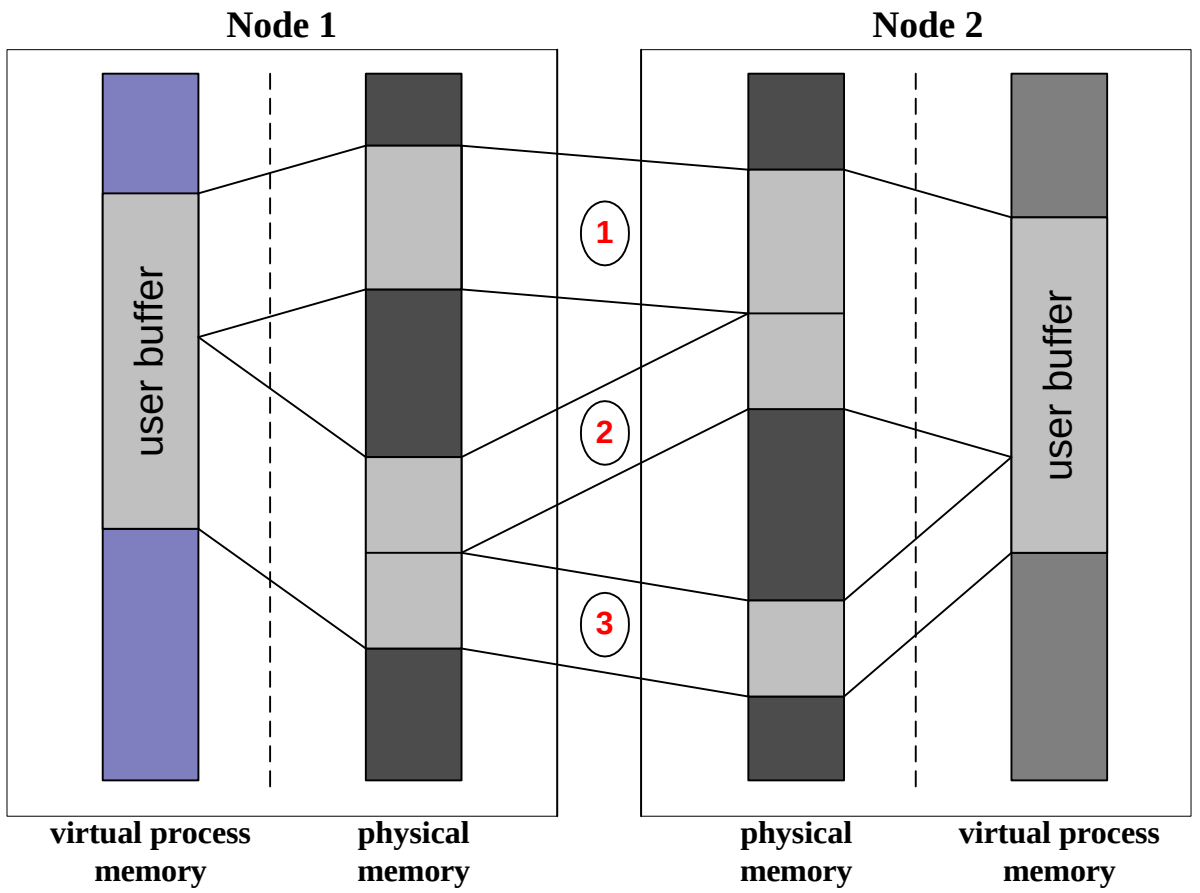


Problématique (1)

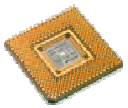
- 2 principaux problèmes :
  - Où écrire les données dans la mémoire physique du nœud récepteur ?
  - On ne peut transmettre que des données contiguës en mémoire physique



Problématique (2)



Ici, 3 buffers physiques sont nécessaires.



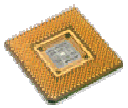
Les messages de l'API GM_P6

Les messages de contrôle

- **SHORT** : émission des données utilisateur avec copie intermédiaire dans des tampons alloués par MPI.
- **REQ** : envoi d'une requête pour un protocole de rendez-vous.
- **RSP** : réponse à une requête (contient la description du tampon de réception).
- **CRDT** : contrôle de flux au niveau MPI (algorithme à crédit).

Les messages data

- Envoi des données utilisateur en mode zéro-copie après un protocole de rendez-vous



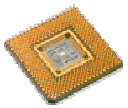
Format des messages de contrôle

REQ : 24 octets

RSP : 20 octets + recv_map

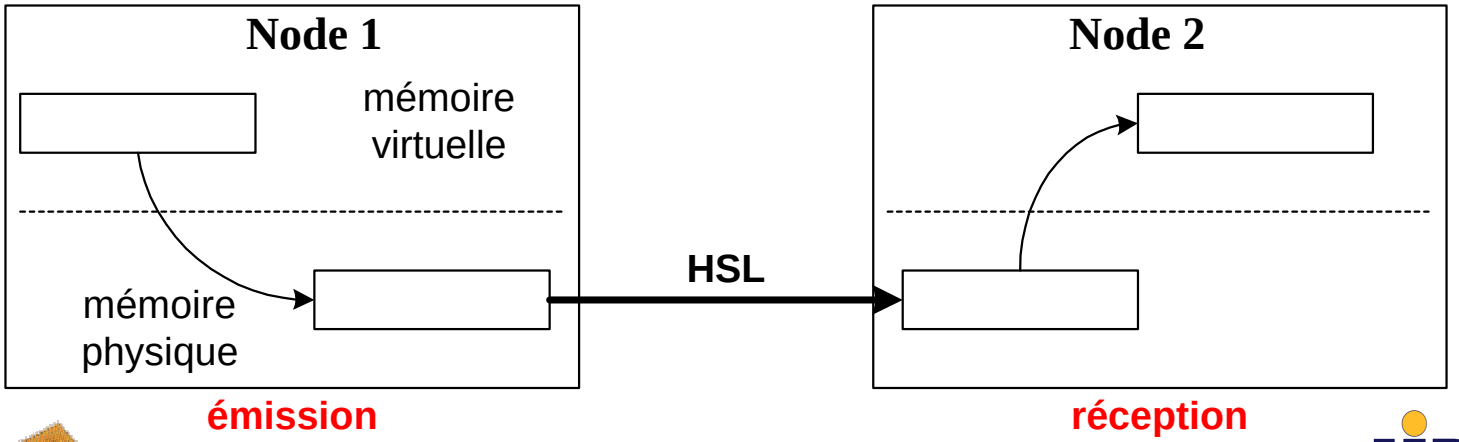
CRDT : 20 octets

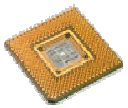
SHORT : 20 octets + données user



Les messages de contrôle

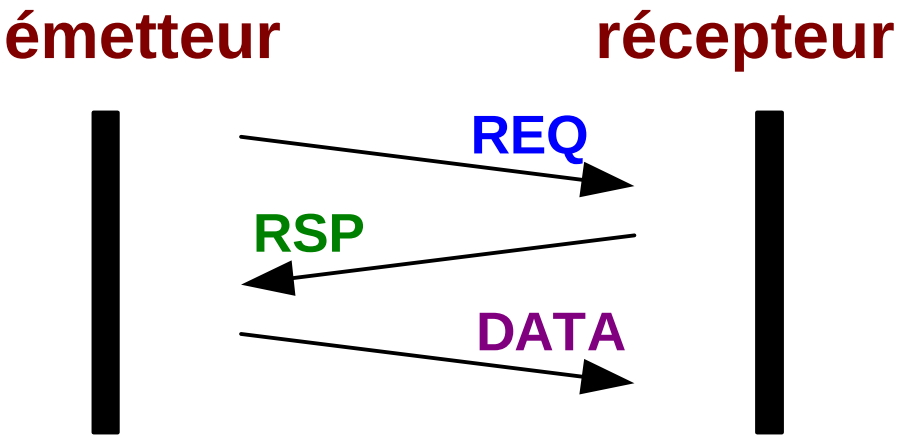
- Transfert d'une information de contrôle (REQ, RSP, CRDT) ou de données utilisateur (SHORT)
- Utilisation de tampons contigus en mémoire physique pré-alloués par MPI au démarrage de l'application (boîtes aux lettres)
- Une copie en émission et en réception (pas grave pour des petits messages)

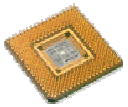




Les messages de données

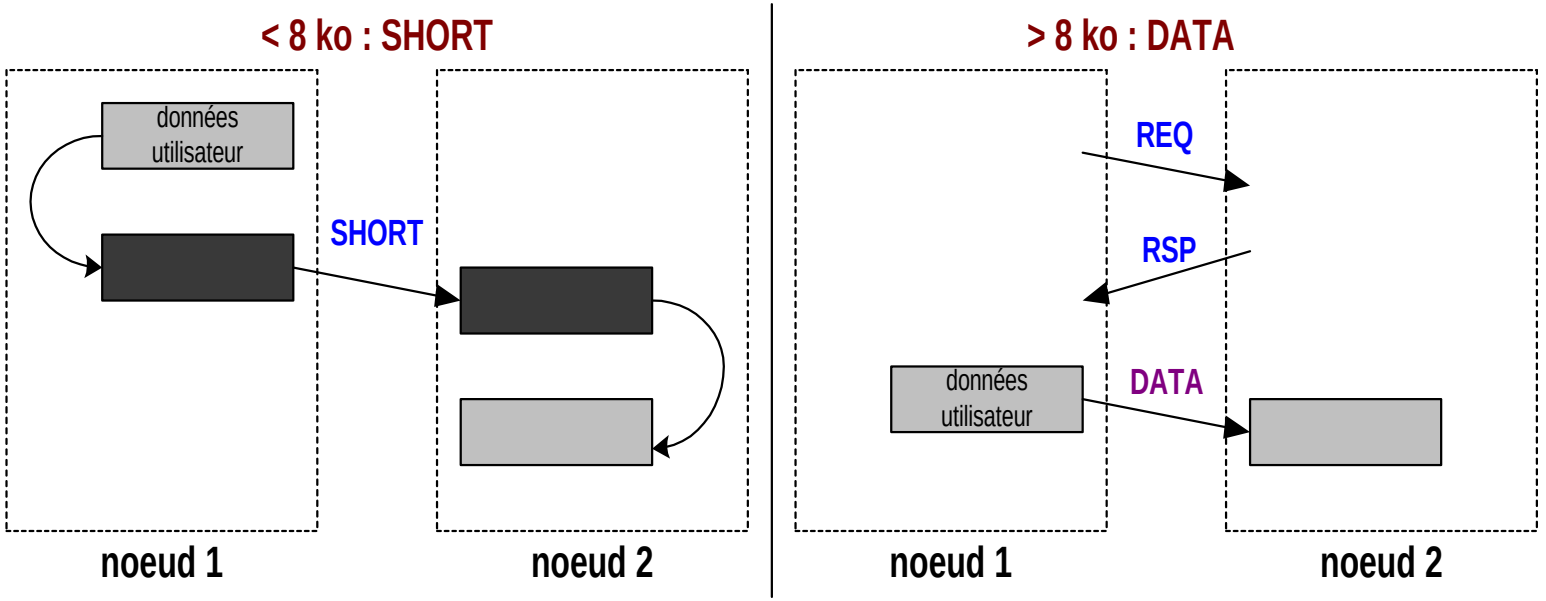
- Transfert des données utilisateur par un protocole de rendez-vous
- Mode zéro-copie
- La description en mémoire physique du tampon de réception est transmise à l'émetteur dans un message RSP





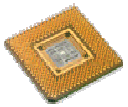
Les primitives MPI (1)

MPI_Send, MPI_ISend



2 copies, 1 message

0 copie, 3 messages



Les primitives MPI (2)

MPI_Ssend et MPI_ISsend

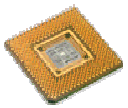
- On se place dans le mode DATA : quelle que soit la taille du message, on a besoin du protocole de rendez-vous pour faire la synchronisation

MPI_Bsend et MPI_IBsend

- Idem que MPI_Send et MPI_ISend

MPI_Rsend et MPI_IRsend

- Idem que MPI_Send et MPI_ISend mais on n'a plus besoin de REQ (on gagne un message de contrôle dans le cadre du protocole de rendez-vous)



Les différentes API dans MPI

API MPI

MPI_SEND(VAD, len, tid_dst, TAG)

MPICH

MPI_RECV(VAD, len, tid_src, TAG)

API GM_P6

GMPI_SEND_DATA(VAD, len, tid_dst, TAG, recv_map, canal)

GMPI_SEND_SHORT(VAD, len, tid_dst, TAG)

MPI-MPC

GMPI_SEND_REQ(tid_dst, TAG)

GMPI_SEND_RSP(tid_dst, TAG, recv_map, canal)

GMPI_SEND_CREDIT(tid_dst)

API RDMA

RDMA_SEND(Nsrc, Ndst, PRSA, PLSA, len, canal, NS, NR)

RDMA_SENT_NOTIFY(Ndst, canal)

RDMA

RDMA_RECV_NOTIFY(Nsrc, canal)

RDMA_LOOKUP()

API PUT

PUT_ADD_ENTRY(Ndst, PRSA, PLSA, len, mi, flags)

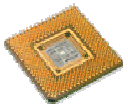
PUT_SENT_NOTIFY(Ndst, PRSA, PLSA, len, mi, flags)

PUT ou BIP

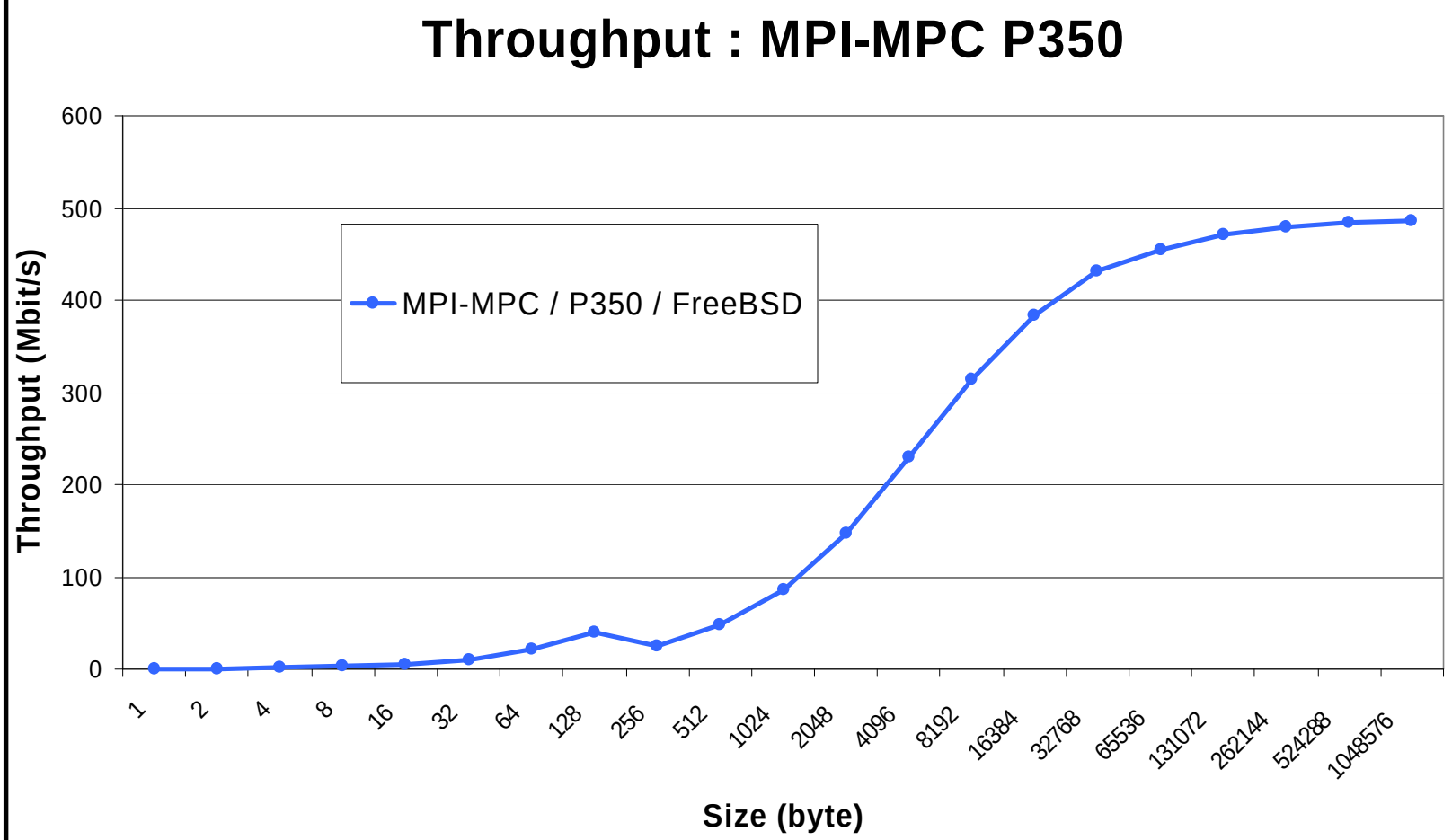
PUT_RECV_NOTIFY(mi, data1, data2)

PUT_FLUSH_LPE()

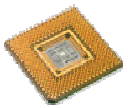
PUT_FLUSH_LMI()



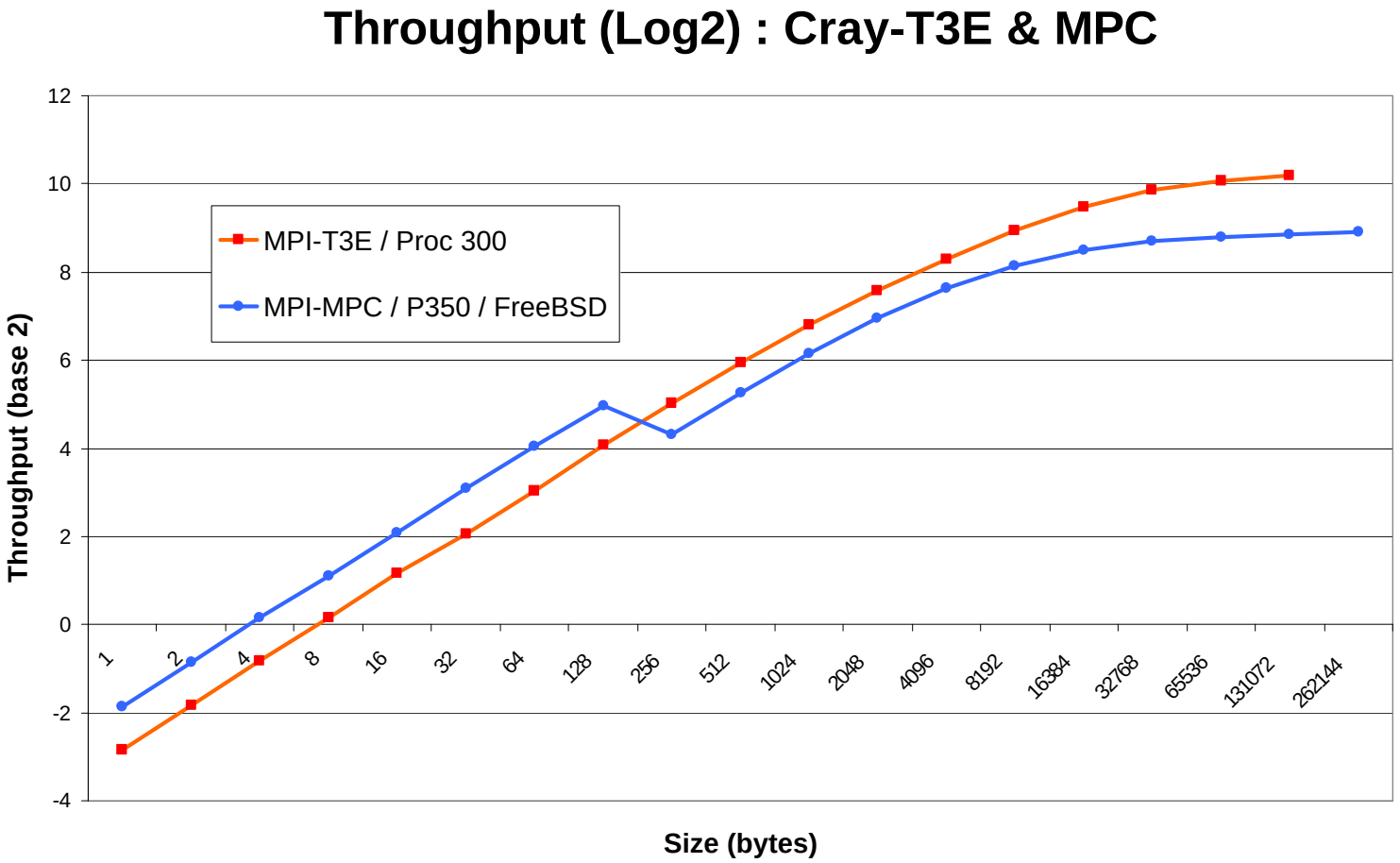
Performances MPI (1)



Latency : 29 µs **Throughput : 490 Mbit/s**

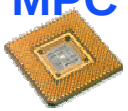


Performances MPI (2)

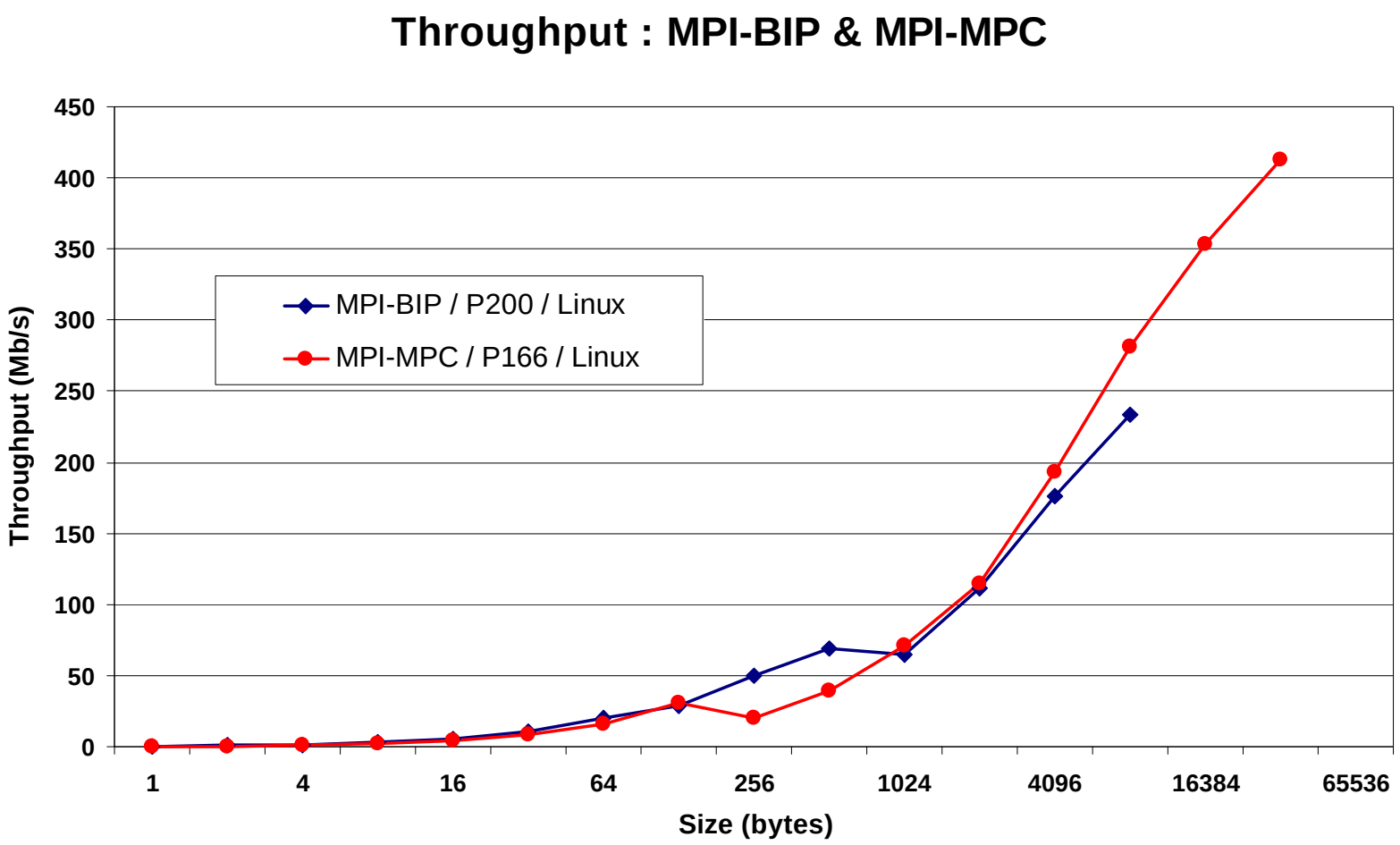


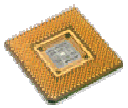
Cray **Latency : 57 µs** **Throughput : 1200 Mbit/s**

MPC **Latency : 29 µs** **Throughput : 490 Mbit/s**



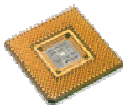
Performances MPI (3)





Comment améliorer les performances ?

- On se place dans le cas d'une seule application MPI à la fois (mode batch)
- On cherche à éliminer les appels système
  ⇨ PUT en mode utilisateur
- Il faut simplifier les opérations de verrouillage et de traduction d'adresses
  ⇨ Redistribution de la mémoire
- On veut rester dans le standard MPI



Les appels système

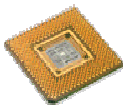
La couche de bas-niveau PUT se trouve dans le noyau.

Appels système les plus fréquents :

- émission : appel de put_add_entry()
- signalisation : soit par interruption soit par polling
  - appel de put_flush_lpe (signalisation en émission)
  - appel de put_flush_lmi (signalisation en réception)

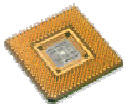
Appels système dus à la traduction d'adresse :

- verrouillage des données en mémoire (mlock)
- traduction adresse virtuelle en adresse physique



Plan

- Introduction
- Architecture matérielle
- La couche de communication bas-niveau (PUT)
- MPI sur la machine MPC1
  - Problématique
  - Les messages internes à MPI
  - Les primitives MPI
  - Premières performances
  - PUT en mode utilisateur
  - Redistribution de la mémoire
- Conclusion



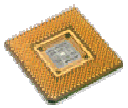
PUT en mode utilisateur (1)

Un PUT simplifié pour MPI et en mode utilisateur :

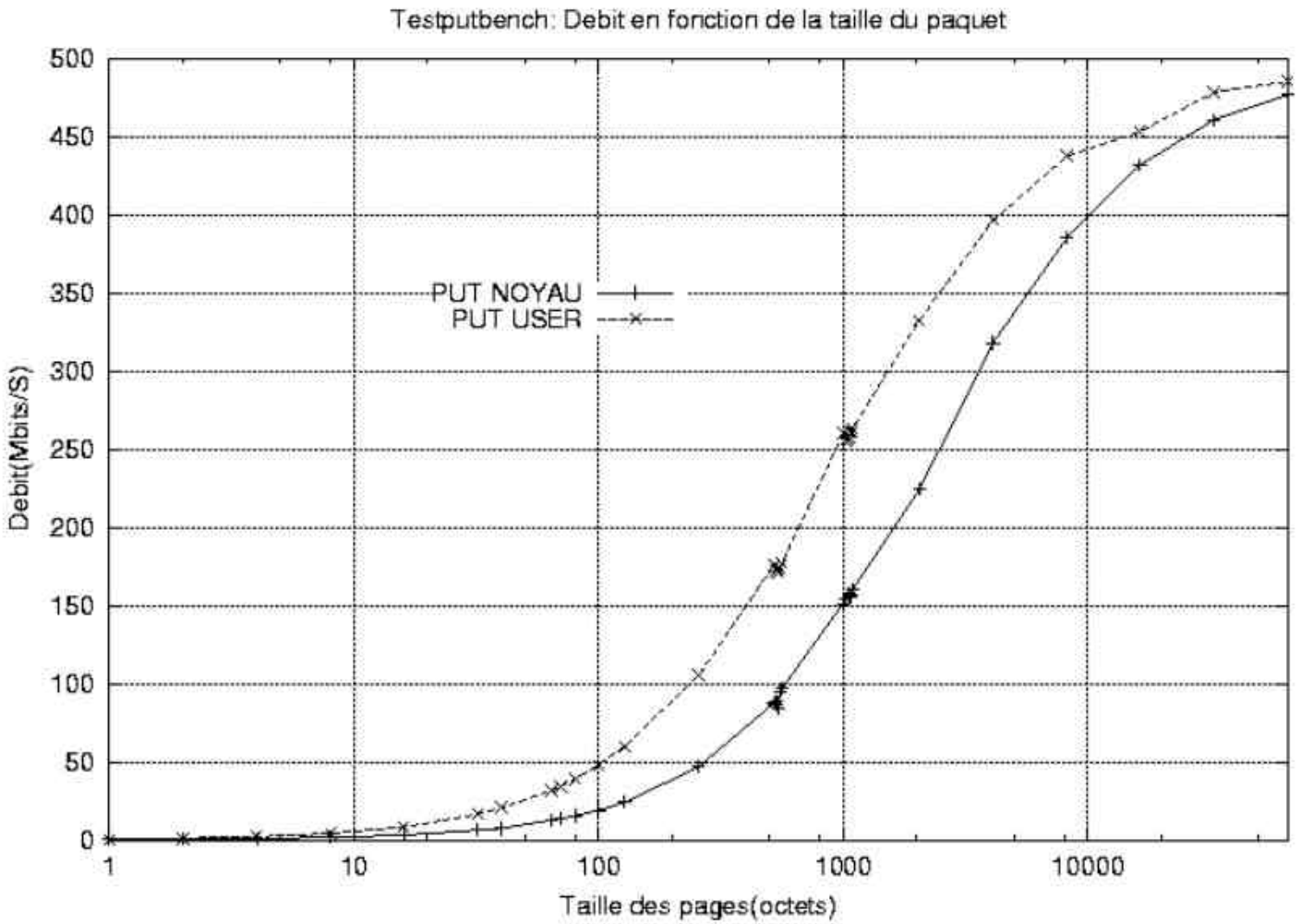
- une seule application utilisatrice de PUT (MPI)
- plus d'appels système pour faire les émissions et la signalisation

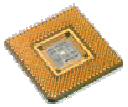
Les problèmes qui se posent :

- accès en configuration à la carte FastHSL en mode utilisateur
- partage des ressources globales (en particulier de la carte réseau) entre les différentes tâches MPI
- Elimination des interruptions trop coûteuses



PUT en mode utilisateur (2)

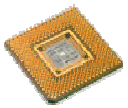




PUT en mode utilisateur (3)

Sur un PII-350MHz, 128 Mo de RAM :

	MPI sur PUT noyau	MPI sur PUT utilisateur
Latence	29 µs	21 µs



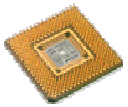
Redistribution de la mémoire (1)

On souhaite éviter les coûts de verrouillage et de traductions d'adresse.

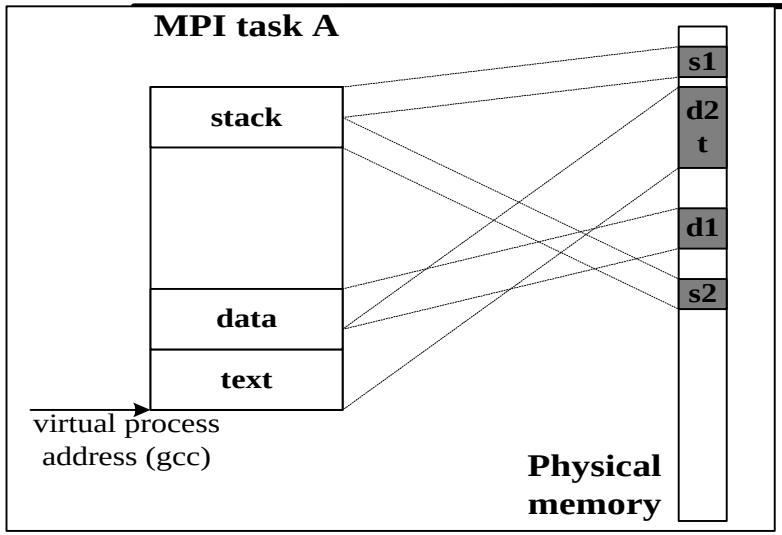
Idée : la mémoire virtuelle de chacune des tâches MPI correspond à une zone contigüe en mémoire physique.

Lors de l'initialisation, on attribue statiquement la mémoire physique de la machine de façon contigüe à chacune des tâches.

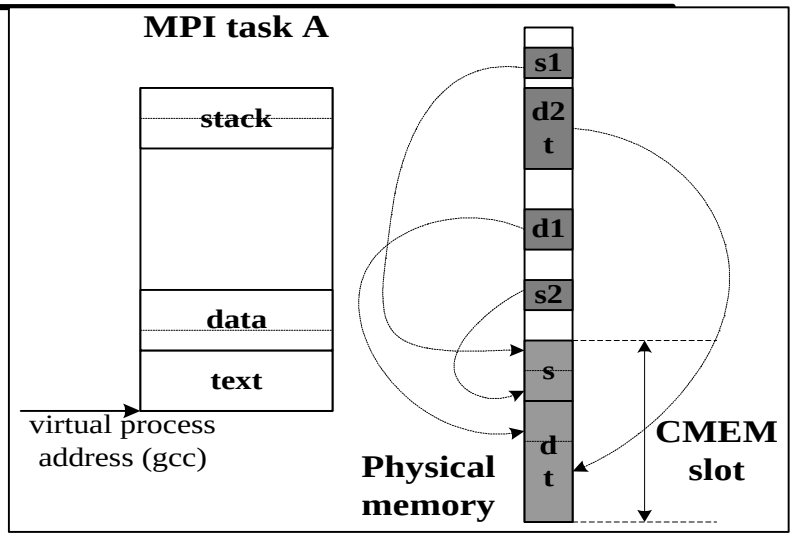
But : adresse virtuelle = adresse physique + offset



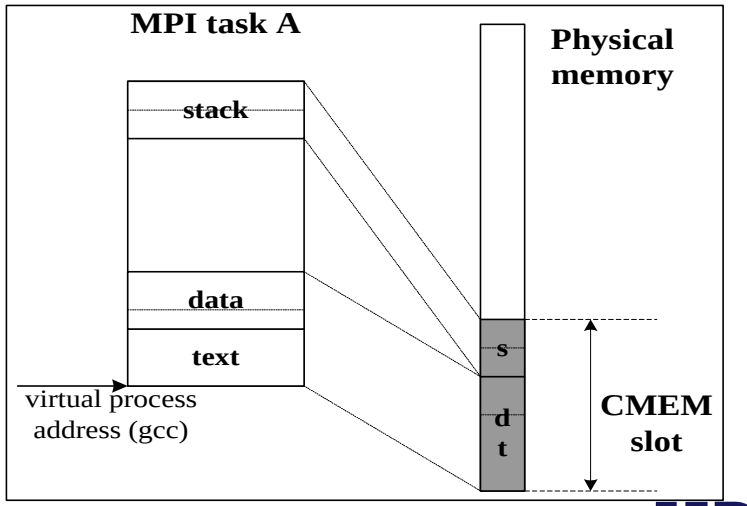
Redistribution de la mémoire (2)



1. locking and mapping



2. copy in CMEM memory

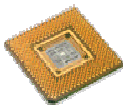


3. unmap and remap

Coûteux lors de l'initialisation

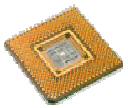
Pas de modification de l'OS

Pas de modification de la libc



Etat d'avancement

- Une première implémentation de MPI sur MPC
  - résultats encourageants
  - fonctionne avec PUT noyau ou PUT utilisateur
- PUT utilisateur
  - mesures à affiner
  - amélioration de la signalisation
  - on a gagné en portabilité
- Redistribution de la mémoire
  - pas implémenté et pas testé sur MPI
  - aucune mesure



Perspectives

- Amélioration de PUT utilisateur
  - on espère passer en dessous des 20 µs de latence avec MPI
- Redistribution de la mémoire
  - n'améliorera pas la latence
  - amélioration du débit pour les moyens messages