# Commications Collectives

Bruno Raffin
-
Laboratoire ID

# Introduction à MPI
# (Message Passing Interface)

Documentation MPI : `www.mcs.anl.gov/mpi`
`www.mpi-forum.org`

Pour compiler un programme MPI : `mpicc prog.c`

Pour exécuter un programme MPI avec 2 precessus:

```
mpirun -np 2 -machinefile fichier_machine a.out
```

Les processus sont distribués sur les machines nommées dans le fichier de "fichier-machine"

# Programmation par Passage de Messages: Message Passing Interface (MPI)

Communications :

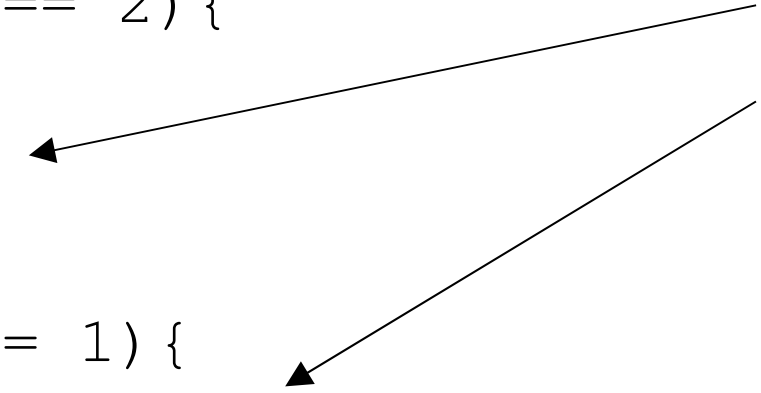
Echanges de données par envois et réceptions de messages (send/receive)

Synchronisations :

Barrière de synchronisation

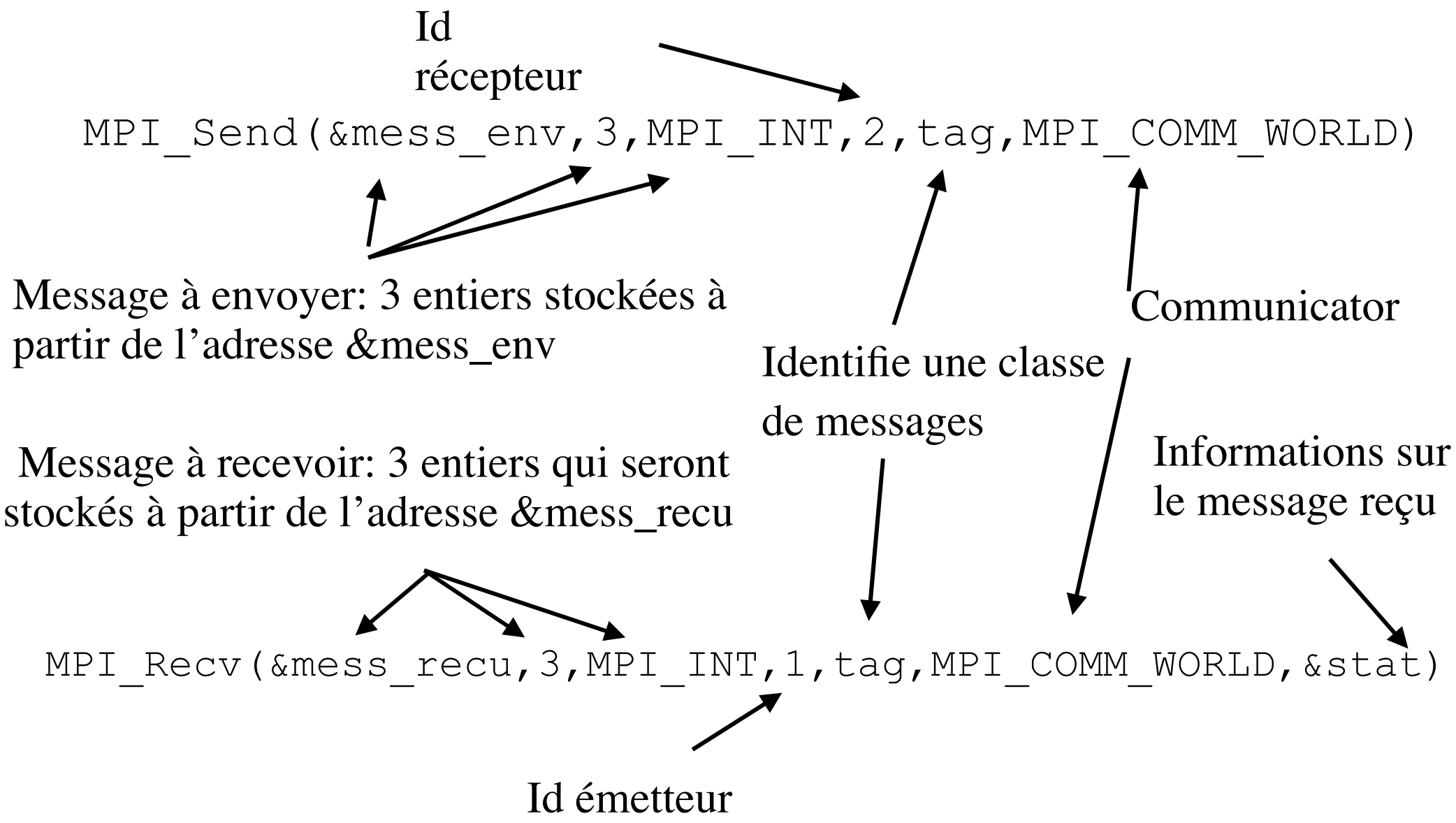
Instructions d'envoi et réception bloquantes

```
if (myrank == 2){
X = 1;
Send X to 1;
}
if myrank == 1){
Receive X from 2 into Y;
}
```



Terminent lorsque le message a bien été transmis

# MPI_Send MPI_Recv



# Les Tags et Communicators MPI

**Communicator :** Il est possible dans une application MPI de définir des réseaux de communication virtuels entre des sous groupes de processeurs.

Le communicator MPI_COMM_WORLD est le communicator

par défaut contenant tous les processeurs de l'application. Les communicators sont essentiels pour l'utilisation des instructions de communications collectives (broadcast, reduce, etc.)

Très important pour limiter les effets de bords liés aux communications

**Tag :** Un tag identifie une classe de messages. Le tag est un numéro associé à un message lors de l'envoi. Le réception demande un message marqué avec un numéro de tag donné.

Processeur 1 :

```
MPI_Send(&m1,3,MPI_INT,2,tag_0,MPI_COMM_WORLD);
MPI_Send(&m2,3,MPI_INT,2,tag_1,MPI_COMM_WORLD);
MPI_Send(&m3,3,MPI_INT,2,tag_1,MPI_COMM_WORLD);
MPI_Send(&m4,1,MPI_CHAR,2,tag_0,MPI_COMM_WORLD);
```

Buffer partagé par
les processeurs 1 et 2

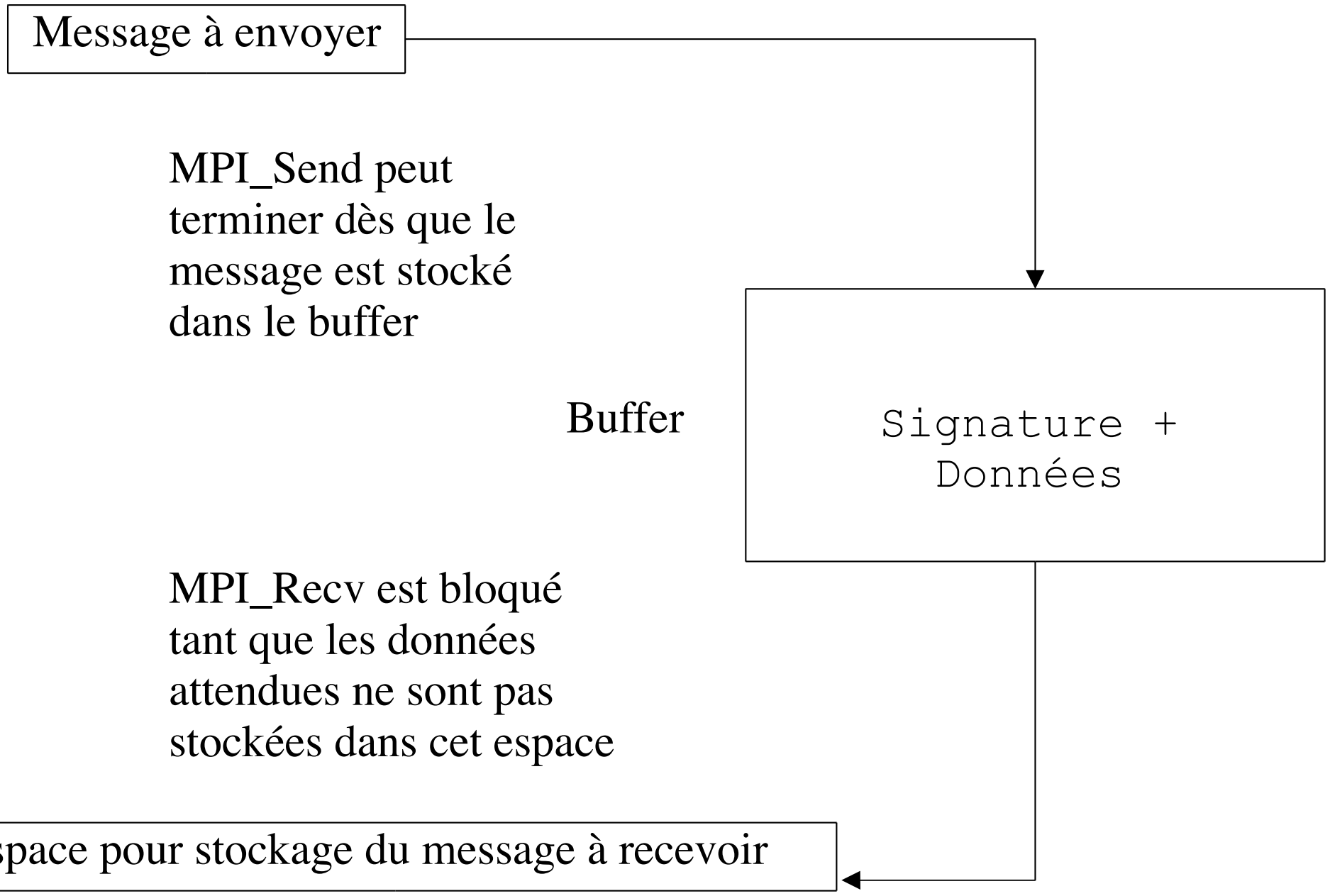
| |
|---|
| 1*MPI_CHAR,tag_0,MPI_COMM_WORLD |
| 3*MPI_INT,tag_1,MPI_COMM_WORLD |
| 3*MPI_INT,tag_1,MPI_COMM_WORLD |
| 3*MPI_INT,tag_0,MPI_COMM_WORLD |

Processeur 2 :

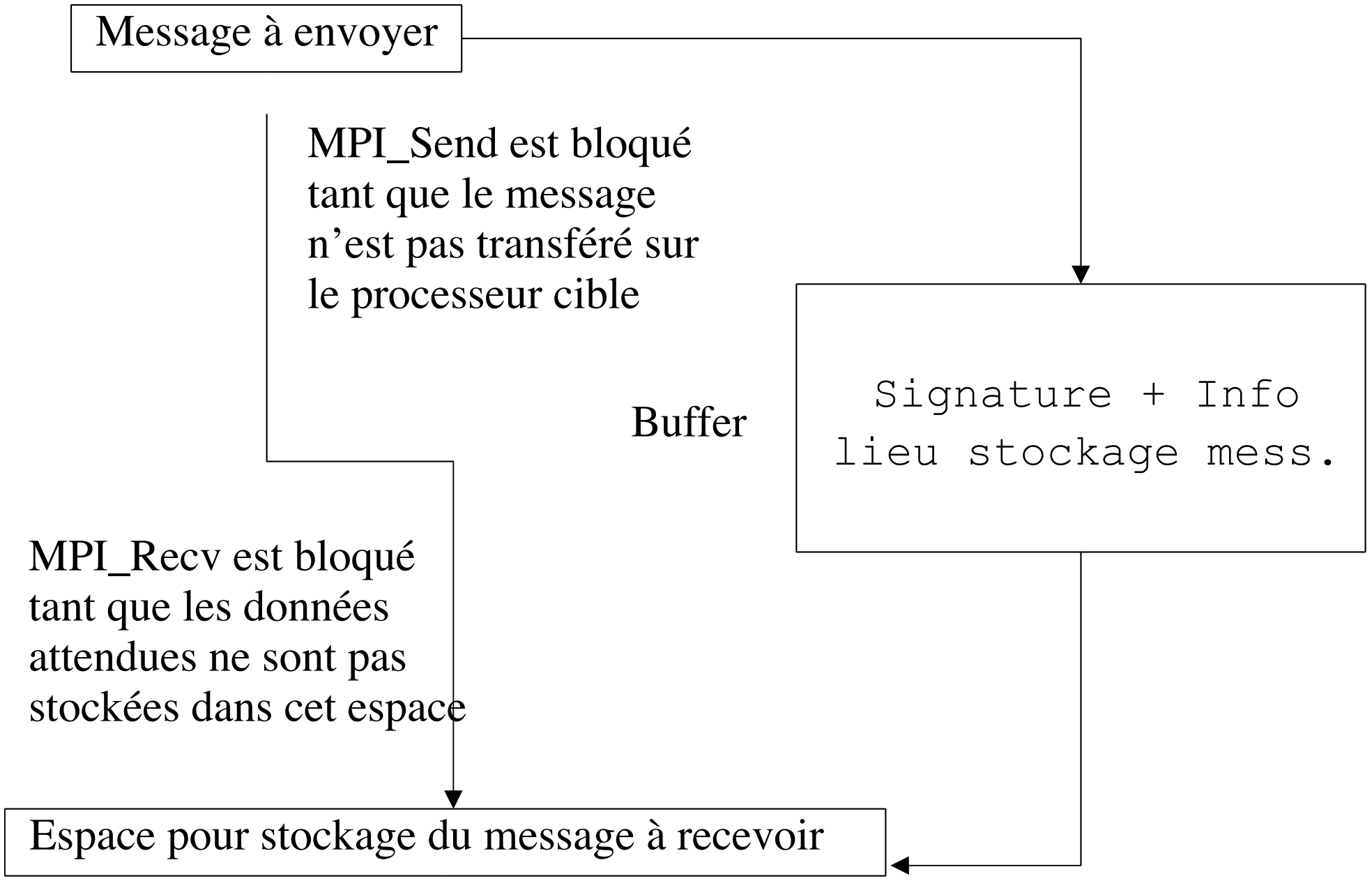
Processeur 2 cherche le plus ancien message avec une signature
`3*MPI_INT,tag_1,MPI_COMM_WORLD`

```
MPI_Recv(&mess_recu,3,MPI_INT,1,tag_1,MPI_COMM_WORLD,&stat)
```

# Gestion du buffer



# Gestion du buffer



# Gestion du buffer

**Stratégie 1 => Buffer : Signature + données**

- Meilleur asynchronisme au niveau du MPI_Send
- Plus de données à stocker dans le buffer

**Stratégie 2 => Buffer : Signature + info sur lieu de stockage des données**

- Evite la copie des données dans le buffer :
  - Moins de données à stocker dans le buffer
  - Evite le temps nécessaire à la copie des données dans le buffer

Stratégie 1 : pour les petits messages

Stratégie 2 : pour les longs messages

Point de basculement entre les deux stratégies contrôlable par un paramètre pour certaines implantations de MPI (Cray, SGI)

# Gestion du buffer

**Exo:** Comment implanter MPI_Send et MPI_Recv lorsque la taille du buffer est limitée ?

**Exo:** Cray livrait ses machines avec MPI configuré pour utiliser la stratégie 1 pour toutes les tailles de messages. Dans ces conditions les performances pour l'envoi de longs messages n'étaient pas optimales, mais par ce choix Cray s'assurait que les programmes de ces clients ne conduisaient pas à des deadlocks. Trouver un programme MPI qui conduise à un blocage si la stratégie 2 est utilisée au lieu de la stratégie 1 (indice: un tel programme MPI n'est pas forcément un «bon» programme").

# Evaluer les performances d'un algorithme parallèle

**Exécuter cet algorithme sur une/plusieurs machine(s) :**

Il faut avoir les machines à sa disposition

Tester plusieurs configurations (faire varier le nombre de processeurs par exemple)

S'assurer de la fiabilité des résultats (influencés par la présence d'autres utilisateurs sur la machine par exemple)

➡ Résultats expérimentaux qui sont souvent remis en cause.

**Disposer d'une fonction de coût :**

Doit capturer un certain nombre des paramètres pertinants des machines que cette fonction est sensée modélisée.

Simple à utiliser

➡ Pas de consensus sur un modèle donné pour le parallélisme (PRAM, LogP, BSP,…)

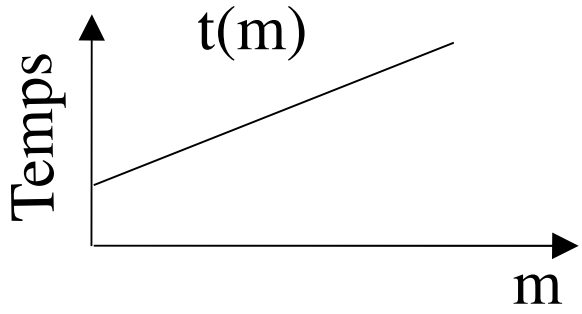
**Emulation:** Complexe. Mis en oeuvre par les constructeurs. Utilisé aussi en P2P

# Coût d'une communication

Une fonction très (trop) simple mais qui intègre les paramètres les plus essentiels :

$$t(m) = \alpha + 1/\beta * m$$

- $\alpha$: latence
- $\beta$ : bande passante
- m : taille du message




On compte un coût t(m) au niveau de l'émetteur et du récepteur lors de l'exécution d'un Send/Recv pour un message de taille m.

De nombreux facteurs peuvent mettre en défaut cette fonction :

Recouvrement communication/communication ou communication/calcul.

Buffers intermédiaires de tailles finies.

Topologie du réseau

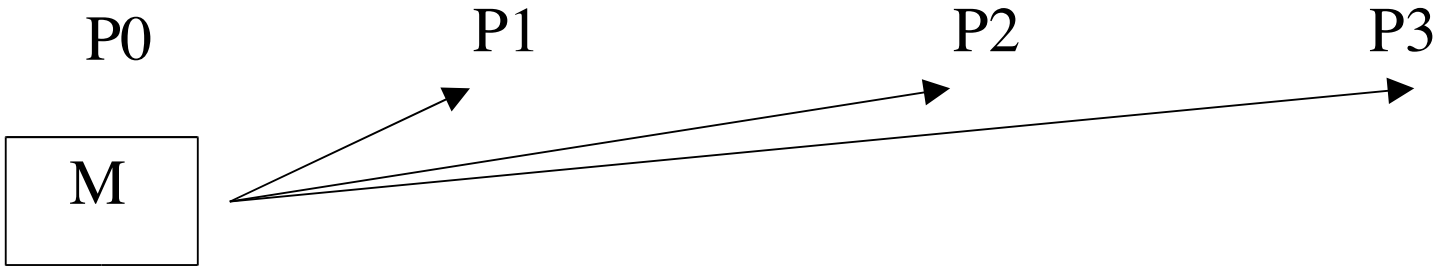
Collisions entre communications exécutées en parallèle

# Communications collectives

Des algorithmes pour organiser une ensemble de communications.

Il existe des schémas classiques de communications collectives que l'on retrouve implantés dans les librairies classiques telles que MPI (Message Passing Interface)

**Le broadcast :** un processeur envoie le même message à un ensemble de processeurs.

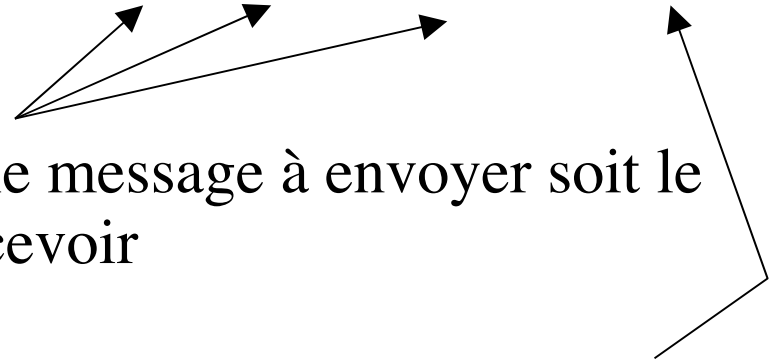



Coût d'une implantation naïve avec p processeurs :

$$t(m) = (p-1)*(\alpha+1/\beta*m)$$

# Broadcast avec MPI

```
MPI_Bcast(&mess,3,MPI_INT,0,MPI_COMM_WORLD)
```



Désigne soit le message à envoyer soit le message à recevoir

Emetteur du message

Communicator désignant l'ensemble des processeurs impliqués dans le broadcast

Les instructions de communications collectives :

Evitent à l'utilisateur de programmer les schémas de communication les plus courants

Permet une implantation optimisée de ces schémas de communication suivant l'architecture cible (en théorie .... la pratique est quelque peu différente)

# Algorithmes de communications collectives

On considère :

p=2^k processeurs

Un réseau complet (un processeur peut directement envoyé un message à un autre processeur)

Un processeur ne peut émettre qu'un message à la fois

Un processeur peut recevoir et émettre un message en même temps

# Gather

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | ABCD | | | |

Données initiales de taille n

On utilise un arbre pour rassembler les données :

$$t(m) = \alpha \log p + (p-1) * n/\beta$$

utiliser la formule classique:

$$\sum_{i=0}^{i=k} q^i = (q^{(k+1)} - 1) / (q-1)$$

# Scatter

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | ABCD | | | |
| Final: | A | B | C | D |

Données initiales de taille n

Gather à l'envers :

$$t(m) = \alpha \log p + (p-1)/p * n/\beta$$

utiliser la formule classique:

$$\sum_{i=0}^{i=k} q^i = (q^{(k+1)} - 1) / (q-1)$$

# Broadcast

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | | | |
| Final: | A | A | A | A |

Données initiales de taille n

On utilise un arbre comme pour le scatter:

$$t(m) = \log p \, (\alpha + n/\beta )$$

# Gather to All

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | ABCD | ABCD | ABCD | ABCD |

Données initiales de taille n

On fait circuler les données le long de l'anneau :

$$t(m) = (p-1)\ (\alpha + n/\beta)$$

# Broadcast (variante)

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | | | |
| Final: | A | A | A | A |

Données initiales de taille n

Scatter + Gather to All sur données de taille n/p:

$$t(m) = \alpha ( p - 1 + \log p) + 2 * n/\beta * (p-1)/p$$

# Broadcast

Comparaison des deux variantes:

$$t(m) = \alpha \log p + \log p * n/\beta$$
$$t(m) = \alpha(p-1 + \log p) + 2 * n/\beta * (p-1)/p$$

Le premier cas sera plus efficace pour les messages courts (latence de poids important relativement à la transmission des données)

Dans le cas de longs messages $n/\beta$ est grand devant $\alpha$ ce qui favorise le second algorithme

**Deux types d'algorithmes suivant la taille des messages**

Sur chaque architecture il reste à identifier le point de basculement entre les deux statégies.

# Gather to All

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | ABCD | ABCD | ABCD | ABCD |

Données initiales de taille n

Messages courts: On fait un gather + broadcast :

$$t(m) = 2 * \alpha * \log p + n/\beta * ((p-1) + p * \log p)$$

Messages Longs: On fait circuler les données le long de l'anneau :

$$t(m) = (p-1)\ (\alpha + n/\beta)$$

# Reduce

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | (A+B+C+D) | | | |

Données initiales de taille n

Message courts : Similaire au broacast à l'envers mais avec des étapes de réduction

$$t(m) = \log p\ (\alpha + n/\beta + n * g)$$

(g = coût d'une opération de reduction)

Messages longs :

# Reduce-Scatter

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | (A+B+C+D)[0] | (A+B+C+D)[1] | (A+B+C+D)[2] | (A+B+C+D)[3] |

Données initiales de taille n

Message courts : reduce + scatter :

$$t(m) = 2 * \alpha * \log p + n/\beta * [( p-1)/p + \log p)] + n * g * \log p$$

Messages longs : messages coupés en p paquets qui circulent sur l'anneau avec réduction à chaque étape

$$t(m) = \alpha * (p-1) + n/\beta\ (p-1)/p + g * n * (p-1)/p$$

(g = coût d'une opération de reduction)

# Reduce

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | (A+B+C+D) | | | |

Données initiales de taille n

Message courts : Similaire au broacast à l'envers mais avec des étapes de réduction

$$t(m) = \log p\ (\alpha + n/\beta + n * g)$$

Messages longs : Reduce-scatter + Gather

$$t(m) = \alpha * [(p-1)+\log p] + 2*n/\beta *(p-1)/p + g * n *(p-1)/p$$

# All Reduce

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | (A+B+C+D) | (A+B+C+D) | (A+B+C+D) | (A+B+C+D) |

Données initiales de taille n

Message courts : Reduce + Broadcast

$$t(m) = 2^* \log p \, (\alpha + n/\beta + n/2 \, ^*g)$$

Messages longs : Reduce-scatter + Gather to All

$$t(m) = 2^*(p-1)^* \alpha + 2^*n/\beta \, ^* (p-1)/p + g \, ^* n \, ^*(p-1)/p$$

(g = coût d'une opération de reduction)

# All-to-All

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | $A_0\,B_0C_0\,D_0$ | $A_1\,B_1C_1\,D_1$ | $A_2\,B_2C_2\,D_2$ | $A_3\,B_3C_3\,D_3$ |
| Final: | $A_0A_1A_2A_3$ | $B_0B_1B_2B_3$ | $C_0C_1C_2C_3$ | $D_0D_1D_2D_3$ |

Données initiales de taille n

Envoi direct de chaque morceau

$$t(m) = (p-1) * (\alpha + n/(\beta * p))$$

Risque de collision entre messages.
Optimisation possible en alternant les envois aux processeurs distants et ceux aux processeurs proches.

-Même coût par rapport à notre modèle
-Gain de 20% sur Cray T3E

# Conclusion

- L'utilisation de stratégies différentes suivant la taille des messages peut conduire à des améliorations de performances intéressantes.

- Même une fonction de coût simple peut apporter des informations pertinentes pour la conception d'algorithmes

- En théorie les instructions de communication collectives de MPI permettent des implantations optimisées. En pratique l'implantation n'est pas toujours très soignée.

➔Vers des fonctions de coût plus sophistiquées

➔ la mesure des performances d'un algorithme sur une machine donnée n'est pas toujours simple (attention aux optimisations cachées)

# Gather to All

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | ABCD | ABCD | ABCD | ABCD |

Données initiales de taille n

Messages courts: On fait un gather + broadcast :

$$t(m) = 2 * \alpha * \log p + n/\beta * ((p-1) + p * \log p)$$

Messages Longs: On fait circuler les données le long de l'anneau :

$$t(m) = (p-1)\ (\alpha + n/\beta)$$

**On echange deux à deux (swap):**

$$\mathbf{t(m) = \alpha * \log(p) + (p-1) * n/\beta}$$

# Broadcast (variante)

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | | | |
| Final: | A | A | A | A |

Données initiales de taille n

Scatter + Gather to All  sur données de taille n/p:

$$t(m) = \alpha ( p - 1 + \log p) + 2 * n/\beta * (p-1)/p$$

**Scatter + Gather to All (swap)  sur données de taille n/p:**

$$\mathbf{t(m) = 2 \alpha \log(p) + 2 * n/\beta * (p-1)/p}$$

# Reduce-Scatter

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | (A+B+C+D)[0] | (A+B+C+D)[1] | (A+B+C+D)[2] | (A+B+C+D)[3] |

Données initiales de taille n

Message courts : reduce + scatter :

$$t(m) = 2 * \alpha * \log p + n/\beta * [( p-1)/p + \log p)] + n * g * \log p$$

Messages longs : messages coupés en p paquets qui circulent sur l'anneau avec réduction à chaque étape

$$t(m) = \alpha * (p-1) + n/\beta\ (p-1)/p + g * n * (p-1)/p$$

**Swap:**

$$\mathbf{t(m) = \alpha \log(p) + n/\beta\ (p-1)/p + g * n * (p-1)/p}$$

# Reduce

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | (A+B+C+D) | | | |

Données initiales de taille n

Message courts : Similaire au broacast à l'envers mais avec des étapes de réduction

$$t(m) = \log p\ (\alpha + n/\beta + n * g)$$

Messages longs : Reduce-scatter + Gather

$$t(m) = \alpha * [(p-1)+\log p] + 2*n/\beta *(p-1)/p + g * n *(p-1)/p$$

**Swap : reduce-scatter (swap) + gather**

$$\mathbf{t(m) = 2\ \alpha\ \log(p) + 2 * n/\beta * (p-1)/p + g * n *(p-1)/p}$$

# All Reduce

| | P0 | P1 | P2 | P3 |
|---|---|---|---|---|
| Init: | A | B | C | D |
| Final: | (A+B+C+D) | (A+B+C+D) | (A+B+C+D) | (A+B+C+D) |

Données initiales de taille n

Message courts : Reduce + Broadcast

$$t(m) = 2 * \log p \, (\alpha + n/\beta + n/2 * g)$$

Messages longs : Reduce-scatter + Gather to All

$$t(m) = 2*(p-1)* \alpha + 2*n/\beta * (p-1)/p + g * n *(p-1)/p$$

**Swap: Reduce-scatter (swap) + Gather to All (swap)**

$$\mathbf{t(m) = 2\alpha \log(p) + 2\, n/\beta\, (p-1)/p + g * n *(p-1)/p}$$

**(idem reduce)**