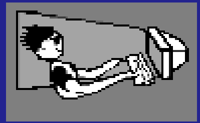
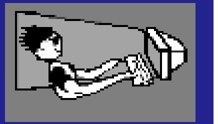


PVM / MPI La Quête de la Bibliothèque de Communication Idéale ?

Frédéric DESPREZ

LIP ENS Lyon

INRIA Rhône-Alpes

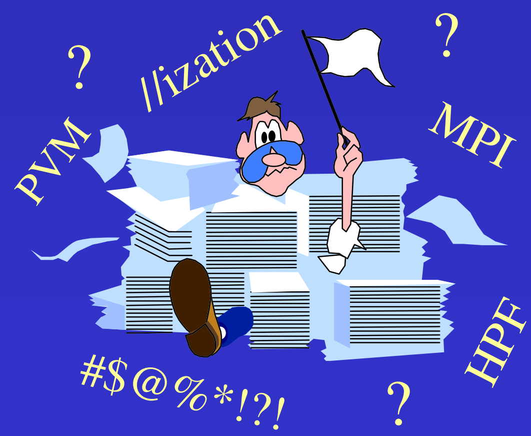


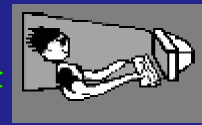
Plan de l'exposé

- Introduction
- MPI, MPI-2 et MPI R/T
- PVM vs MPI
- Conclusions et perspectives

Comment programme-t-on une machine parallèle?

- Sans le savoir (SMP, VSM, bibliothèques //, ...)!
- Compilateur + directives (HPF, ...).
- Avec une bibliothèque de communication.
- Avec des threads.
- Un mélange de tout çà.





APPLICATIONS

Outils de Parallélisation

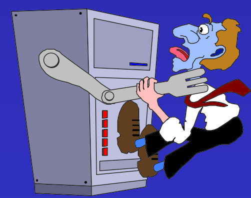
Compilateurs HPF-like

Threads

Bibliothèques

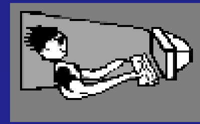
VSM

Bibliothèques de communication

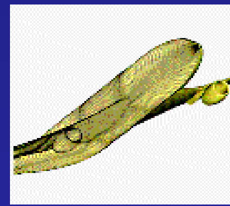


Appels systèmes de bas niveau

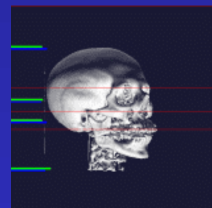
MATERIEL

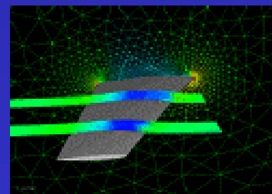


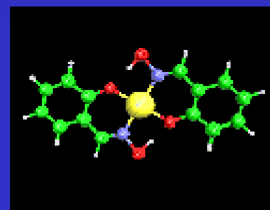
Quelques questions

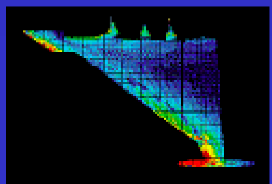


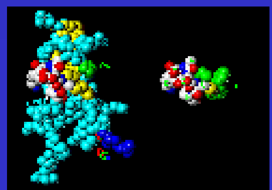
- sur l'application cible
  - numérique, simulation, ...
  - image, vidéo, sons, ...
  - bancaires, aide à la décision, ...
  - temps réel,
  - gestion, bases de données, ...
  - développement d'outils, d'OS, de bibliothèques, ...

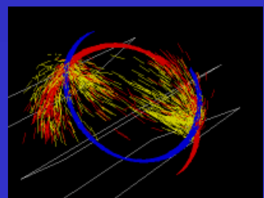


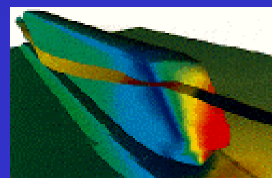




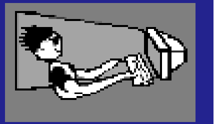




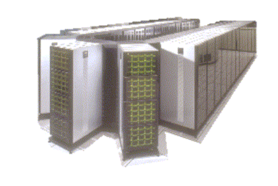




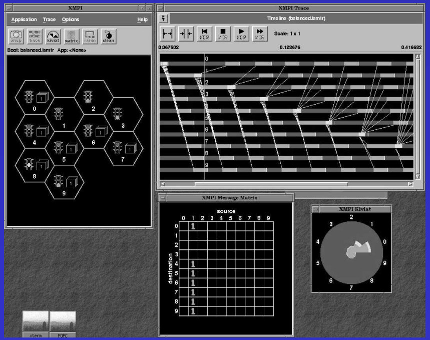
Quelques questions, suite.



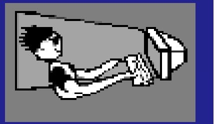
- sur la (les) machines?
  - SM, DM, UMA, NUMA, COMA, ...
  - DM, VSM, ...
  - Hétérogénéité, clusters de SMP, ...



- Quel(s) environnement(s)?
  - UNIX, Windows NT, ...
  - Bibliothèques numériques, imagerie, ...
  - C, C++, F77/90/95, HPF, lisp, ...
  - CORBA, ...

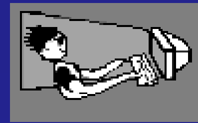


Pourquoi utiliser le passage de messages?



- Le moyen le plus performant de programmer les machines parallèles à mémoire distribuée.
- Bien compris (mais pas forcément bien utilisé).
- Il n'y a pas de compilateur paralléliseur efficace pour les *vraies* applications.
- Certaines applications se codent plus facilement de cette manière.
- Il faut développer des outils construits sur des bibliothèques performantes (et portables).
- C'est un moyen portable!

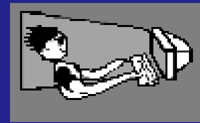
Mais c'est dur !!!




MPI
MPI-2
MPI/RT

Qu'est-ce que MPI ?

- Message Passing Interface !
- La spécification d'une bibliothèque de passage de messages,
- pour les machines parallèles, les clusters de stations de travail (hétérogènes ou pas).
- Conçue pour le développement sûr des bibliothèques et des programmes.
- Conçue en 2 ans par un consortium international (par des groupes de travail et par e-mail).



Pourquoi n'ont-ils pas choisi une bibliothèque existante?

- Chaque vendeur de machine parallèle avait sa propre bibliothèque (parfois efficace mais pas portable)!
- Des bibliothèques portables (principalement des projets de recherche)
  - Incomplètes.
  - Parfois dépendantes de l'application cible.
  - Peu de support vendeurs.
  - Pas tellement efficaces, si vraiment portables.

Quelles étaient leurs motivations ?

- De concevoir une bibliothèque avec de nombreuses fonctionnalités intéressantes:
  - Communications bas et haut niveau.
  - Portabilité.
  - Modularité (pour le développement de bibliothèques).
  - Supportant l'hétérogénéité.
  - Avec Groupes et des topologies de processus.
- Elle doit rester efficace sur la plupart des machines parallèles.

Histoire de MPI-1

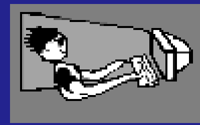
- Démarre en Avril 1992 (style HPF).
- Groupes de travail toutes les 6 semaines.
- Draft final MPI-1 en Mai 94.
- Quelques corrections après (MPI-1.1).
- Groupes de travail et forums e-mail.

Institutions pour MPI-1

Companies		Laboratories		Universities	
ARCO	Convex	ANL	GMD	UC Santa Barbara	Syracuse U.
Cray Research	IBM	LANL	LLNL	Michigan State U.	Oregon Grad. Inst.
Intel	KAI	NOAA	NSF	U. of New Mexico	Mississippi State U.
Meiko	NAG	ORNL	PNL	U. of Southampton	U. of Colorado
nCUBE	ParaSoft	Sandia	SDSC	Yale U.	U. of Tennessee
Shell	TMC	SRC		U. of Maryland	Western Michigan U.
				U. of Edinburgh	Cornell U.
				Rice U.	U. of San Francisco

Partis de bibliothèques existantes

- Ils ont pris les meilleures fonctionnalités de quelques bibliothèques éprouvées
  - PVM et PICL
  - P4, Zipcode,
  - CHIMP, Parmacs, Express
  - Intel NX
  - développements d'IBM
- Pas de fonctionnalités vraiment nouvelles.



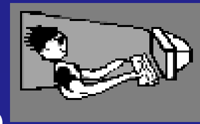
Qu’est-ce qu’il y a dans MPI-1 ?

- Communications point-à-point .
  - support pour les buffers structurés et les types dérivés.
  - plusieurs modes de communication.
  - support pour l’hétérogénéité.
- Routines de communication collectives.
  - Toutes les routines dont vous réviez.
  - Opérations pré-définies et utilisateur.
  - Communication dans les sous-groupes de processus.
  - Communications dans des topologies.

Qu'est-ce qu'il y a dans MPI-1 , suite ?

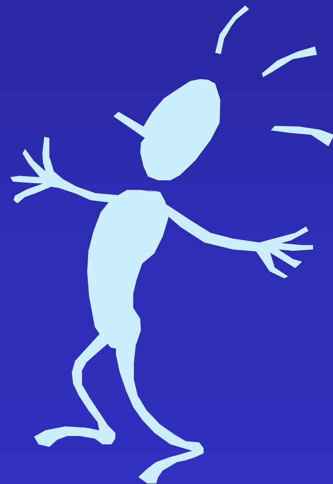
- Support pour les groupes de processus.
- Mécanismes pour séparer les messages de diverses applications (pour le développement de bibliothèques sures).
- Interfaces C et Fortran 77.
- Une interface de profiling.

Rien n'est dit à propos de l'implémentation. MPI spécifie juste l'interface avec les routines.

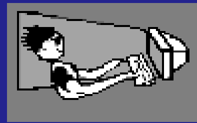


Qu’est-ce qu’il n’y a pas dans MPI-1 ?

- Opérations de mémoire partagée (put, get).
- Messages actifs.
- Gestion de processus.
- Support pour les threads.
- Support pour le debug.
- Entrees/Sorties parallèles.



Certaines de ces fonctionnalités ont été définies dans MPI-2 !

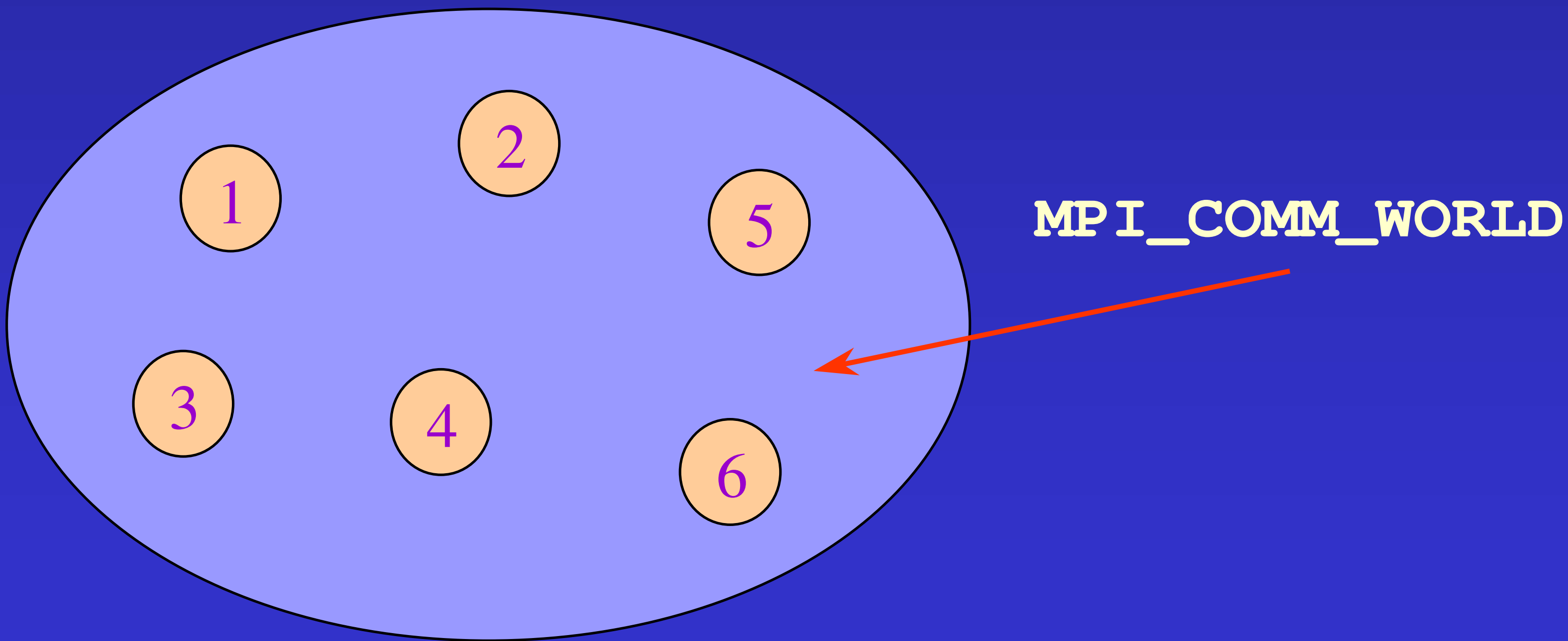


Processus et Groupes ds MPI-1

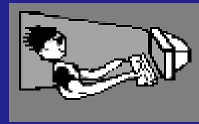
- MPI-1 suppose que les processus ont été démarrés (aucune routine pour les démarrer ou les arrêter) - style de programmation SPMD.
- Chaque processus a son propre flot de contrôle et son propre espace d'adressage.
- MPI supporte les groupes de processus dynamiques, mais l'appartenance est statique
  - Les groupes peuvent être créés et détruits.
  - Les groupes peuvent se recouvrir.
- Le placement des processus sur les processeurs ne fait pas partie de MPI.

MPI_COMM_WORLD

- Chaque processus possède un rang unique dans la machine virtuelle.
- Le communicateur **`MPI_COMM_WORLD`** contient tous les processus démarrés (statique).



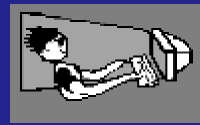
Communicateurs, suite.



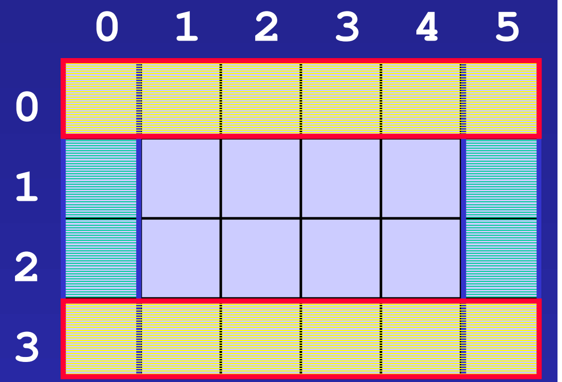
- Un ensemble de processus sensés se connaitre.
- Chaque communication MPI a lieu par rapport à un communicateur.
- Définit l'étendue d'une communication (il contient les informations de *groupe* et de *contexte*).
- **`MPI_COMM_WORLD`** est un communicateur prédéfinit.
- Les processus peuvent appartenir à plusieurs communicateurs (avec un rang différent pour chaque communicateur).
- Il définit un espace de communication sûr.

Bufferisation Classique de Messages

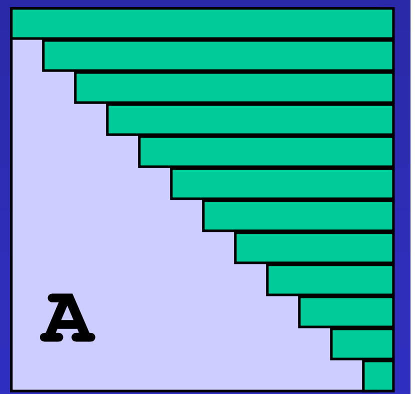
- Dans les bibliothèques de bas niveau
  - On envoie des zones contigues de données.
  - On a besoin d'un pré-emballage des données.
- Cela simplifie le processus d'optimisation.
- mais
  - Cela empêche les communications dans les machines hétérogènes ou les réseaux de stations.
  - Le pré-emballage est parfois difficile.

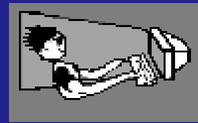


Bufferisation MPI



- Vous pouvez avoir
  - Des tableaux de types élémentaires.
  - Des zones contigues de données.
  - Des blocs de types avec saut.
  - Des tableaux indexés de blocs de types.
  - Des structures générales.
- Les types sont construits récursivement.
- Vous pouvez avoir des types de données dépendants de l'application.





Communications MPI simples

```
mpi_send( OUT start_of_buff,
          IN count,
          IN data_type,
          IN dest_rank,
          IN tag,
          IN communicator,
          OUT error_code)
```

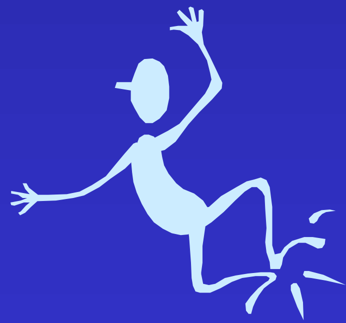
```
mpi_recv( OUT start_of_buff,
          IN count,
          IN data_type,
          IN source_rank,
          IN tag,
          IN communicator,
          OUT return_status,
          OUT error_code)
```

source_rank et tag peuvent être des jokers (MPI_ANY_RANK et MPI_ANY_TAG)

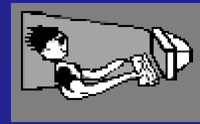
Sous-ensemble MPI-1

- Il suffit de ces 6 routines pour écrire des programmes MPI (simples).

```
MPI_Init(...)
MPI_Comm_size(...)
MPI_Comm_rank(...)
MPI_Send(...)
MPI_Recv(...)
MPI_Finalize()
```



MPI est simple!

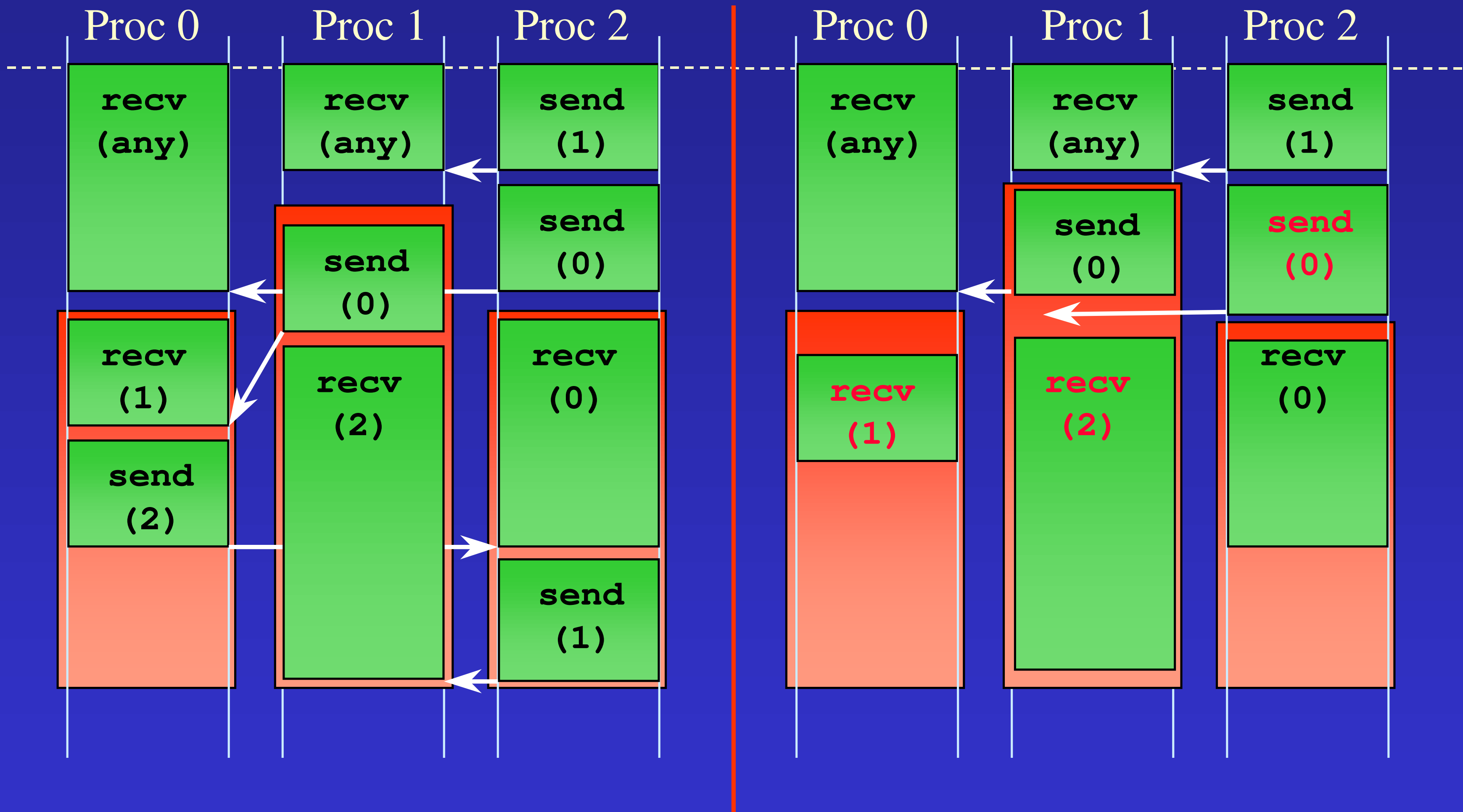


Un mot sur les communications sécurisées.

- La sécurité des communications est importante!
  - Avec des bibliothèques.
  - Avec des programmes *exotiques*.
- MPI fournit des moyens pour séparer les messages.
  - Contextes de communication (dans les communicateurs).

    → peut être vu comme des *étiquettes système*.
  - Informations sur les groupes.
  - Etiquettes de messages.
- **`MPI_COMM_WORLD`** contient le premier contexte et le groupe de tous les processus démarrés.
- MPI Garantie l'ordre des messages.

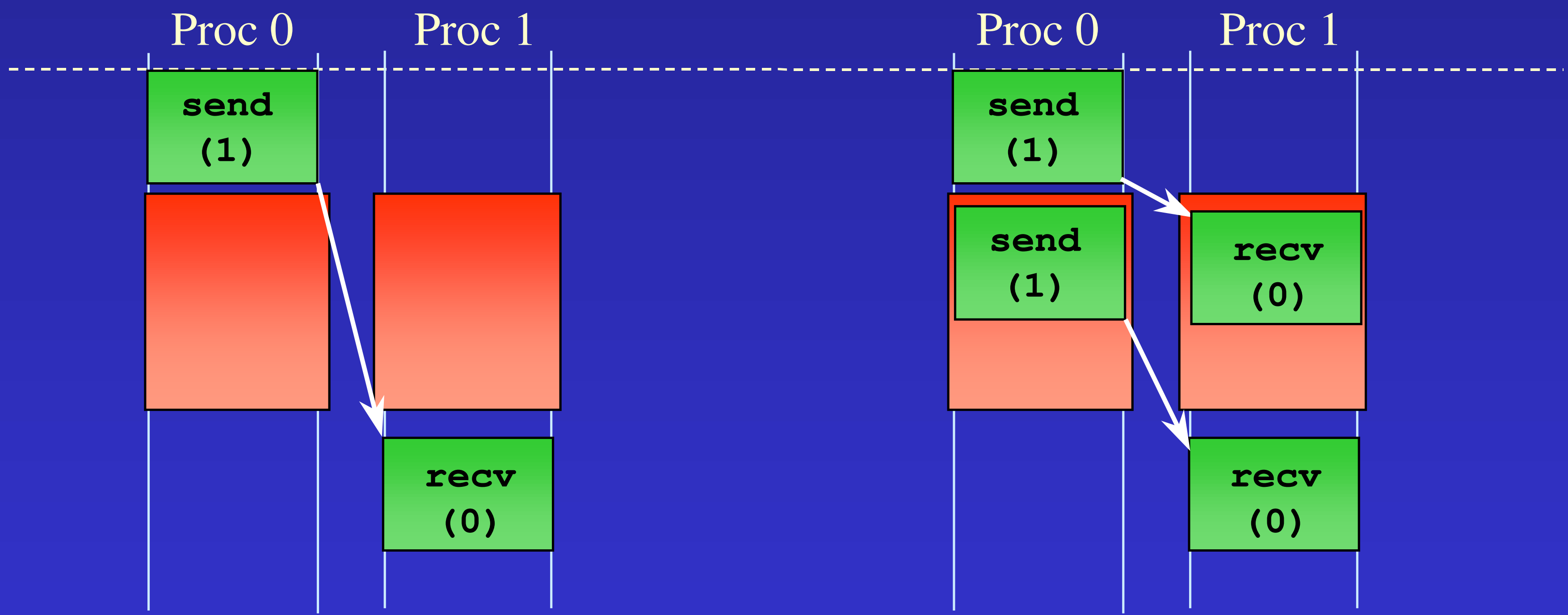
Appels de routine asynchrones



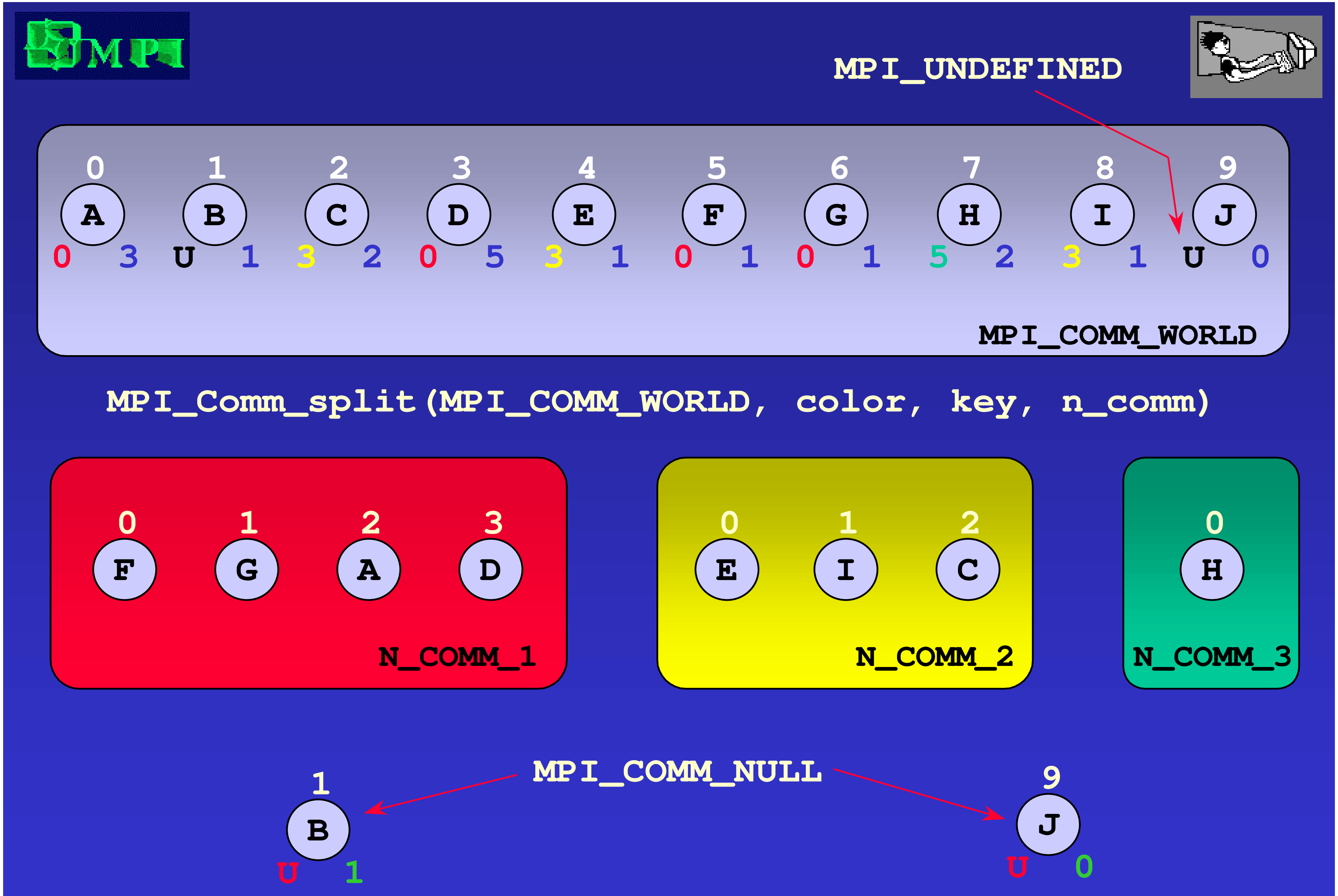
BLOCAGE !

Appels de routines synchrones

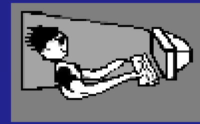
Est-ce que la synchronisation est la solution ultime?

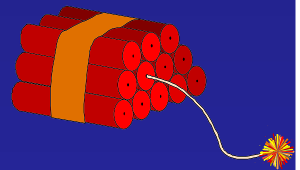


BLOCAGE !

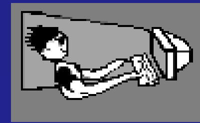

MPI_UNDEFINED
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J
0 3 U 1 3 2 0 5 3 1 0 1 0 1 5 2 3 1 U 0
MPI_COMM_WORLD
MPI_Comm_split(MPI_COMM_WORLD, color, key, n_comm)
0 1 2 3
F G A D
N_COMM_1
0 1 2
E I C
N_COMM_2
0
H
N_COMM_3
MPI_COMM_NULL
1
B
U 1
9
J
U 0

Pour les développeurs de bibliothèques!





- On peut attacher des informations à un ou plusieurs communicateurs en utilisant les *attributs*.
  - initialisation des paramètres d'une bibliothèque.
  - des communicateurs additionnels.
  - ....
- Les attributs sont référencés par *keyval* (créée (de manière interne) par **`MPI_Keyval_create`**).
  - **`MPI_Attr_put`** attacher un attribut à un communicateur.
  - **`MPI_Attr_get`** permet d'accéder aux valeurs.

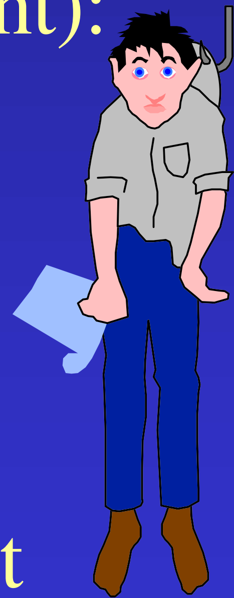


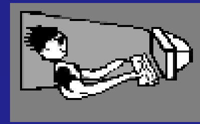
Modes de Communication

- Quoi ? On ne dit pas juste

  le message A va de X à Y ?

- Dans MPI vous avez un plus grand contrôle sur la transmission avec 4 modes (envoi uniquement):
  - Synchronous send
  - Buffered send
  - Standard send
  - Ready send
- Mais vous n’avez pas forcément à savoir tout ça !





Modes de Communication, suite.

- Supposez qu'un envoi bloquant s'est terminé. Est-ce que ça nous dit qque chose sur la terminaison de la communication? Non! En fait, cela dépend!
- Si vous voulez des performances, il faut éviter la bufferisation système

  Mais alors un envoi bloquant est vraiment bloquant (jusqu'à ce qu'une réception soit démarrée).
- Si vous ne voulez pas de ce rendez-vous, alors il faut avoir une certaine bufferisation

  Mais vous perdez des performances !

Modes de Communication, suite.

- Mode **Standard** `MPI_Send/MPI_Isend`

  L'envoi ne terminera pas tant que le buffer d'envoi est plein (aucune supposition sur le côté receveur).

- Mode **Bufferisé** `MPI_Bsend/MPI_Ibsend`

  Un buffer utilisateur est fournit et l'envoi se terminera aprés qu'il soit vidé (ou copié dans un buffer système).

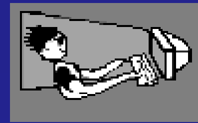
- Mode **Synchrone** `MPI_Ssend/MPI_Issend`

  L'envoi ne se termine pas tant que le message n'est pas arrivé.

- Mode **Ready** `MPI_Rsend/MPI_Irsend`

  L'utilisateur assure qu'une réception a démarré!!!

  → Simplifie le protocole, mais peut être source d'erreur.



Envoi Synchrone et Bufferisé

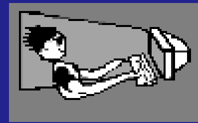
- Envoi synchrone
  - Evite la bufferisation système.
    Utilisez `SSend` si vous savez que un recv sera démarré en même temps.
  - Sinon, l'émetteur attendra un bon moment.
- Envoi bufferisé
  - Bufferisation mais on peut recouvrir les calculs et les communications simplement.
  - Gestion de vos propres buffers (`MPI_Buffer_attach`).
- Compromis à trouver!

Autres Modes

- Envoi Standard
  - Le mode «*les performances de communications ne sont pas si importantes*» or «*j'ai confiance en mon implémentation de MPI*».
  - Dépendant de l'implémentation (peut être implémenté avec les envois synchrone ou bufferisé).
- Envoi Ready
  - Le mode «*je sais ce que je fais*».
  - Potentiellement dangereux!
  - Vous pouvez obtenir de très bonnes performances.

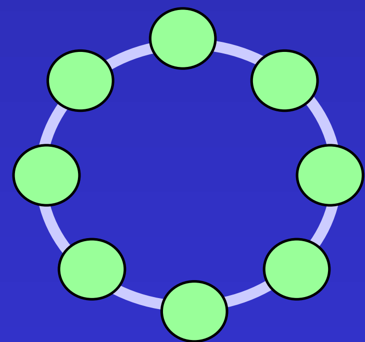
Communications Non-bloquantes

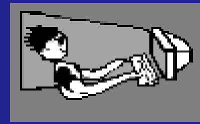
- Cela ne signifie pas asynchrones!
- Peuvent être très efficaces
  - si vous avez assez de calculs à recouvrir.
  - si votre machine posède un co-processor de com.
- Parfois il y a un surcoût système
  - bufferisation supplémentaire.
  - system polling ou interruptions.
- N'utilisez pas `ssend` and `irecv` (utilisez `issend`)!
- Il y a un compromis à trouver!



Autres fonctionnalités de Comm.

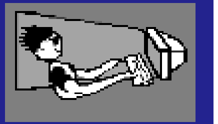
- Communications persistentes
  communications à 2 étapes (Init et Start)
  - L'appel Init prend la liste d'arguments et crée une requête (**`MPI_Send(_Recv)_init`**).
  - L'appel Start utilise la requête et démarre la communication (**`MPI_Start(_all)`**).
  - Un seul init pour plusieur starts.
- Communication sendrecv
  `MPI_Sendrecv(_replace)`





Communications Collectives

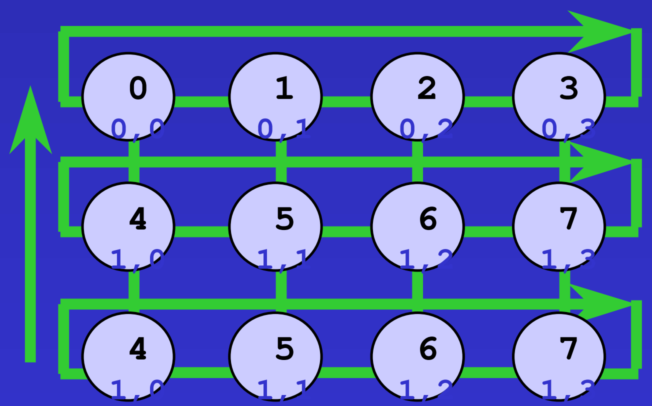
- Transmettre des données entre tous les processus spécifiés par un communicateur.
  - Barrière de synchronisation
  - Diffusion
  - Scatter/gather
  - All-to-all
  - All-gather
  - Réductions (SUM, MAX, MIN, ..., fct utilisateur).
  - Préfixe/scan.



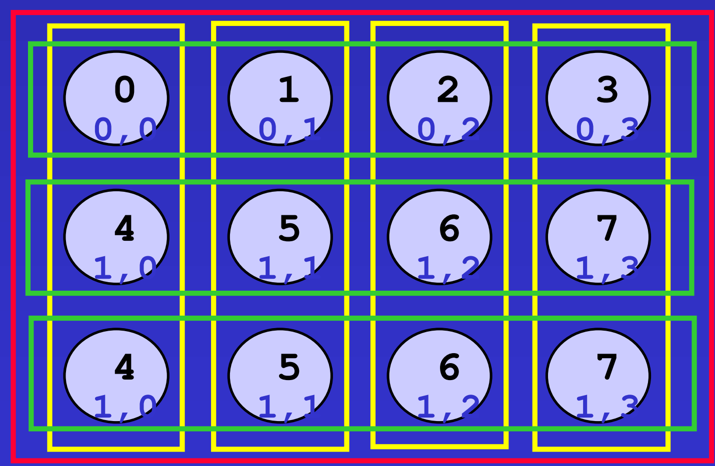
- **MPI_Bcast**
  MPI_Reduce_Scatter
  MPI_Gather
  MPI_Scatter
  MPI_Allgather
  MPI_Alltoall
  MPI_Allreduce
  MPI_Reduce
  MPI_Scan
  MPI_Gatherv
  MPI_Scatterv
  MPI_Allgatherv
  MPI_Alltoallv
- Les versions **All** laissent le résultat sur tous les processus.
- Les versions **v** permettent de communiquer des morceaux de vecteurs.
- Les routines **MPI_*reduce*** et **MPI_Scan** autorisent l'emploi de fonctions utilisateur.

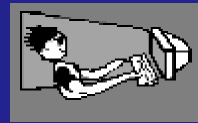
Topologies virtuelles

- Routines pour structurer les ensembles de processus.
- Pour répondre à cette simple question: *Où sont mes voisins dans cette topologie?*
- Peut aider les implémentations de MPI à arranger les processus sur les processeurs.
- Topologies cartésiennes ou graphes généraux.



New_Comm
Comm_Row
Comm_Col





MPI trop gros ou trop petit?

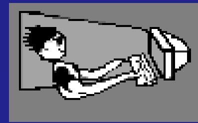
- MPI est petit (6 fonctions sont nécessaires).

  `MPI_INIT, MPI_Comm_size, MPI_Comm_rank, MPI_Send,`

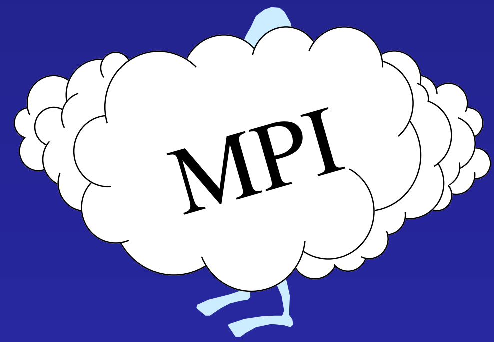
  `MPI_Recv, MPI_Finalize.`

- MPI-1 est gros (128 fonctions).
  - Avoir beaucoup de fonctionnalités implique un grand nombre de fonctions.
  - Pour avoir des performances et simplifier la programmation.
- MPI est de la bonne taille
  - Un petit nombre de concepts.
  - flexibilité, robustesse, efficacité, modularité, et utilité.
  - Vous n'avez pas besoin de tout savoir!

MPI-2



MPI-2



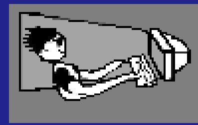
- **Cool, des nouvelles fonctions !!!**
- **Quand?**

  Démarré en Mai 1995.
- **Pourquoi?**

  Pour étendre MPI-1 et faire quelques corrections.
- **Comment?**

  Différents sous-comités (gestion dynamique de processus, communications *one-sided*, comm. collectives étendues, interfaces externes, interfaces C++ et F90) et un autre forum pour MPI I/O.
- **Dernier document:** 18 Juillet 1997.

Gestion Dynamique de Processus



- MPI rejoint (pratiquement) PVM!
- **`MPI_Comm_spawn`** démarre quelques processus et établit les communications (communication collective sur **`comm`**)

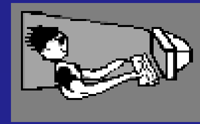
```
MPI_Comm_spawn(command, argv, maxp, info, root,
               comm, i_comm, error_codes);
```

- **`MPI_Comm_spawn_multiple`** démarre plusieurs exécutables.

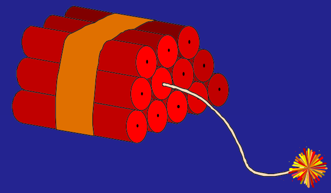
```
MPI_Comm_spawn(count, array_of_commands,
               array_of_argv, array_of_maxp,
               array_of_info, root, comm, i_comm,
              error_codes);
```

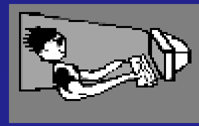
- Il est conseillé de démarrer tous les processus en même temps.

L'objet **`info`**



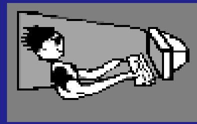
- Ensemble de paires (**`key, value`**) (string).
- Plusieurs clés réservées
  - **`host`** hostname (dépendant de l'implémentation).
  - **`arch`** nom de l'architecture (dep. de l'impl.)
  - **`wdir`** répertoire de travail des processus.
  - **`path`** Rangement des exécutables.
  - **`file`** nom d'un fichier qui contient des informations additionnelles (dep. de l'impl.)
  - **`soft`** valeurs permises pour le nombre de processus démarrés (**`0:N`**, **`2:10000:2`**, **`2:10:2,7`**).





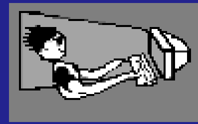
MPI_Spawn n'est pas PVM_SPAWN

- En MPI, tous les processus d'une application doivent être démarrés à la fois.
- Quand on utilise **`MPI_Spawn`**, les performances ne seront pas forcément obtenues.
- Problèmes de portabilité pour cette fonction.



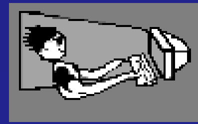
Communications *One-sided*

- Ajout de communications put et get au paradigme de rendez-vous traditionnel.
  - Le transfert de données nécessite un appel de bibliothèque dans un seul processus.
  - meilleures performances sur certains systèmes (support matériel, machines à mémoire partagée).
  - Plus simple pour certaines applications (schémas de communication irréguliers ou dynamiques, compilateurs data-parallèles, ...).
- Quand même un modèle à mémoire distribuée.



Interfaces Externes

- Requêtes généralisées.
- Décodage des types.
- MPI et threads.
  - Tous les appels MPI sont *thread safe*.
  - Les appels MPI bloquants bloqueront uniquement le thread appelant.
  - **`MPI_INIT_THREAD, MPI_QUERY_THREAD`**, ...
- *Caching* sur les communicateurs, les types et les fenêtres.
- ...

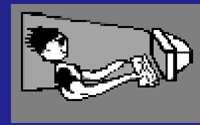


Interfaces de Langages et Interopérabilité

- Interface C++ : ensemble de classes et interfaces de langage.
- Interfaces Fortran 90

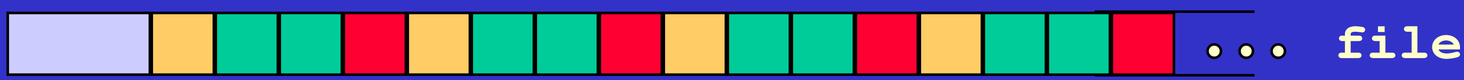
```
call MPI_Recv(a(1:100:2), MPI_REAL, ...)
```

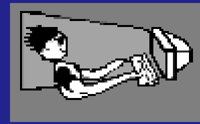
- Mélange de langages (C, C++, Fortran, F90)
  - fonctions f2c-like pour les objets opaques (Fortran vers C)
  - Fonctions C++ pour l'interoperabilité inter-langages.
  - Communications inter-language possibles.



MPI I/O

- Les E/S parallèles sont un point crucial pour le développement d'applications parallèles.
  Goulot d'étranglement pour certaines applications.
- Les E/S parallèles signifient un accès fichier concurrent à partir de processus multiples dans un programme parallèle.
- MPI I/O peut être considéré commes les E/S Unix plus de nombreuses autres fonctionnalités.
- Démarré à l'IBM T.J.Watson Research Center en 93.





MPI/RT

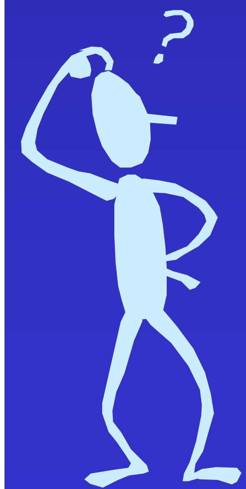
- Fait partie du J.o.D. MPI (Journal of Development).
- Démarré en Avril 95.
- But

  Fournir un environnement de programmation standardisé pour le temps réel.

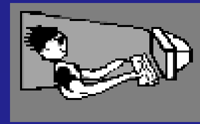
`http://www.cs.msstate.edu/mpirt/`

- MPI a de bonnes chances de devenir un standard, qui remplacera les bibliothèques existantes (dans la plupart des cas).
- Les principaux vendeurs de machines ont déjà des implémentations efficaces de MPI sur leur machines.
- Des implémentations efficaces existent également dans le domaine public (LAM, MPICH, ...).
- Conçue par des personnes intéressées par la programmation d'applications parallèles (chercheurs, vendeurs de machines parallèles, développeurs d'applications, ...).

VS

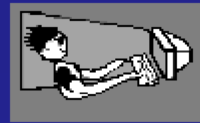



EuroPVM
95
LIP
ENS-Lyon
France
13TH-15
Sept 1995
PVM
MPI
MPI
IBM



Vous vous rappelez de PVM?

- Projet de recherche démarré en 89.
- Standard de fait très utilisé.
- A suscité de nombreuses vocations.
- Notion de Machine Parallèle Virtuelle.
- Interface simple.
- Processus et groupes dynamiques.
- Communications point-à-point et collectives.
- Tolérance aux pannes.
- A évolué continuellement au long des années.



Portabilité vs Intéropérabilité

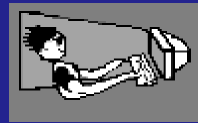
- PVM et MPI produisent des programmes portables (i.e. qui peuvent être compilés et exécutés sur des architectures différentes).
- PVM permet à des machines différentes de coopérer (avec leurs propres implémentation de PVM).

  → Nécessite une conversion → surcoût
- MPI ne le requiert pas (pour les performances).
- Aucune des implémentations de MPI ne peut coopérer.
- L'intéropérabilité au niveau langage fait partie de MPI-2.

Contrôle de Processus et de Ressources

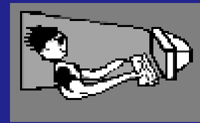
- PVM possède un ensemble riche de routines pour le contrôle de processus.
- MPI a définit quelques routines dans MPI-2.
- PVM est dynamique par nature.
  - facilite la gestion de ressources.
  - tolérance aux pannes.
- MPI est statique (mais autorise un comportement *dynamique*).
- Fonctionnalités dépendantes des implémentations dans MPI.
- MPI permet la définition de topologies.

Tolérance aux Pannes



- Problème critique pour les applications aux temps d'exécutions trés longs.
- PVM possède un schéma de notification basic. Les tâches sont prévenues quand le statut de la machine virtuelle change.
- Les tâches et les hôtes MPI sont considérés statiques.
- Pas de mécanisme pour se rétablir de la mort d'un processus en MPI.
- MPI RT s'occupe du problème de tolérance aux pannes.

Contextes de Communication



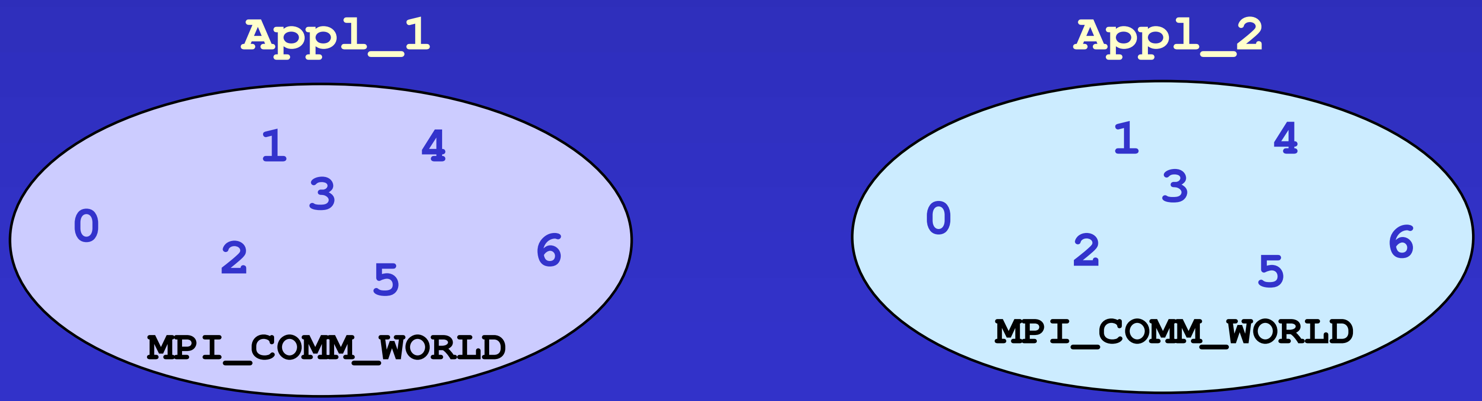
- Contextes dans MPI et dans PVM 3.4.
- Les contextes statiques de MPI impliquent l'utilisation d'inter-communicateurs.
- Si une tâche plante en MPI, `MPI_COMM_WORLD` est invalide et le comportement du prog. dépent de l'impl.
- En PVM, contextes uniques générés par le démon.
- PVM n'a pas de contextes en paramètres de `pvm_send/recv`.
- PVM possède un contexte système, qui peut être créé par le programme. Cela peut poser des problèmes dans les programmes multi-threads.

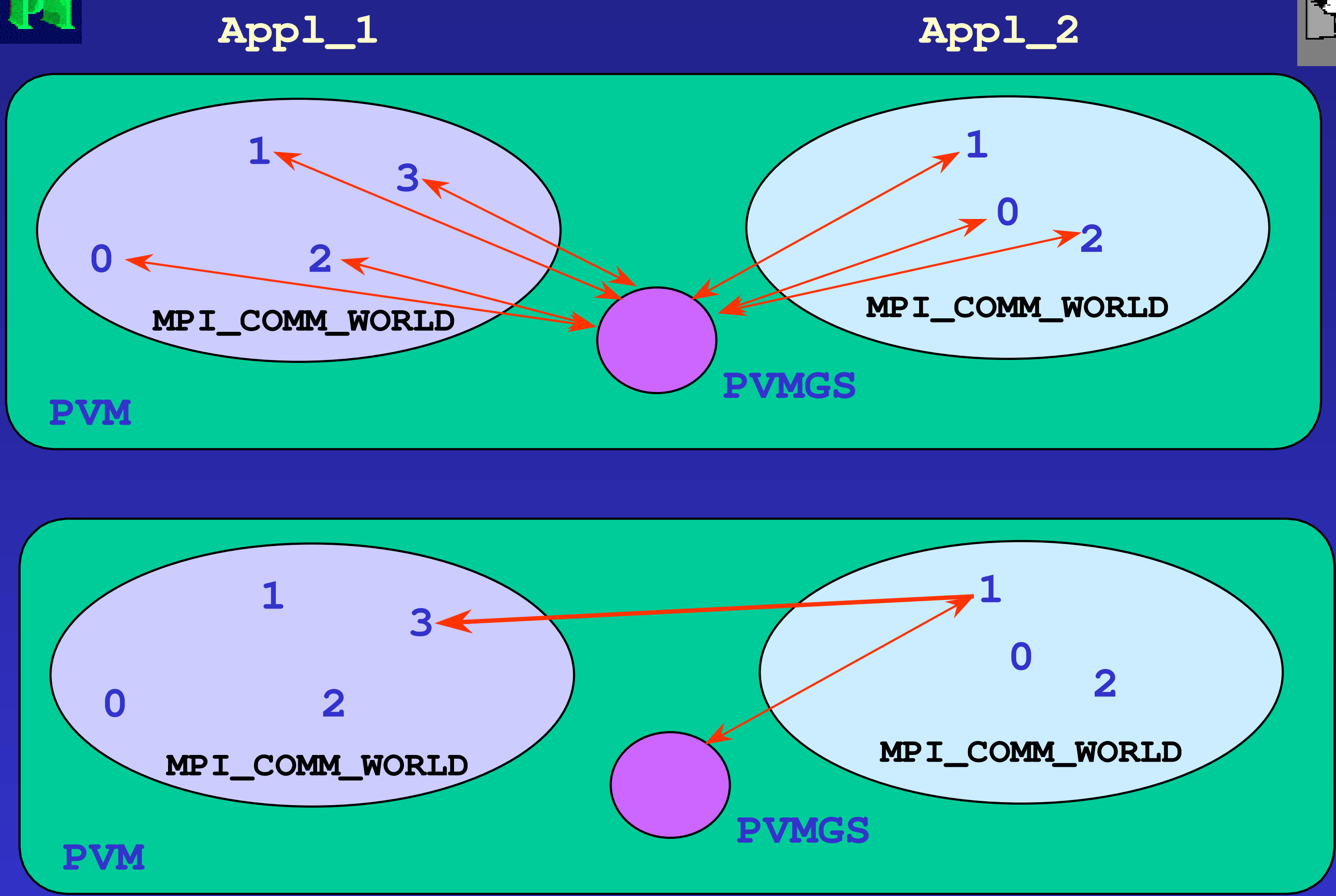
Alors, PVM ou MPI?

- Pour l'instant, les deux!
- Pour une application qui doit s'exécuter sur une machine parallèle homogène, **utilisez MPI** (plus rapide, ensemble de routines plus riche).
- Vous avez besoin d'intéropérabilité, de tolérance aux pannes, de dynamicité simple, **utilisez PVM**!
- PVMPI?

PVMPI

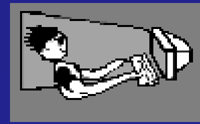
- L'intéropérabilité entre différentes implémentations de MPI peut être cruciale.
- PVMPI permet de connecter 2 applications à travers PVM.
- Les groupes de processus s'enregistrent via un service de nommage global.





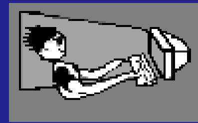

```
pvm_send(pvm_gettid(«Appl_1»,3), tag);
```

CONCLUSION



Conclusion

- Le développement de MPI-1 a suivi sa définition (voir par exemple MPICH).
- L'implémentation de MPI-2 sera plus lente.
  - Le démarrage de processus existe (LAM).
  - De nombreuses implémentation de MPI I/O.
- De nombreux projets ont démarré autour de MPI.
- Les bibliothèques parallèles et les outils commencent à être portés.



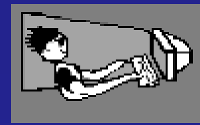
Conclusion, suite.

- MPI est trop gros ou trop petit!
- Il y a de nombreuses implémentations
  - de nombreuses versions du domaine public.
  - tous les vendeurs de machines suivent le *standard*.
- Vous pouvez obtenir de très bonnes performances en conservant la portabilité.
- Vous pouvez programmer à bas niveau ou à haut niveau.
- Utilisez MPI et développez des projets avec!

REFERENCES

Livres MPI

- **Using MPI: Portable Parallel Programming with the Message Passing Interface**, *W. Gropp, E. Lusk, and A. Skjellum*, The MIT Press, 1994.
  `http://www.mcs.anl.gov/mpi/usingmpi/`
- **MPI: The Complete Reference**, *M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, J. Dongarra*, The MIT Press, 1996.
  `http://www.netlib.org/utk/papers/mpi-book/mpi-book.html`
- **Parallel Programming with MPI**, *P.S. Pacheco*, Morgan Kaufman, 1997.
  `http://www.usfca.edu/mpi/`



Livres MPI/PVM, suite.

- **MPI-2: Extensions to the Message Passing Interface**, *Message Passing Interface Forum*, July 18, 1997.
- **Designing and Building Parallel Programs**, *I. Foster*, Addison Wesley, 1994.

  `http://www.mcs.anl.gov/dbpp/`
- **PVM: Parallel Virtual Machine**, *A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam*, The MIT Press, 1994.

  `http://www.netlib.org/pvm3/book/pvm-book.html`

MPI sur Internet

- Newsgroup

  `comp.parallel.mpi`
- MPI Forum (documents officiels)

  `http://www.mpi-forum.org/`
- LAM

  `http://www.osc.edu/lam.html`
- Mississippi State University

  `http://www.erc.msstate.edu/mpi/`
- Argonne National Lab.

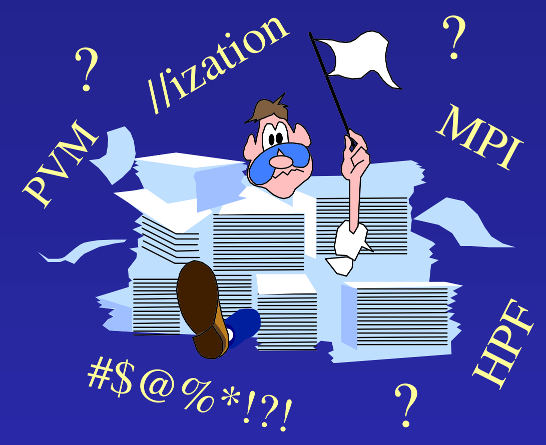
  `http://www.mcs.anl.gov/mpi/`
- Oak Ridge National Lab.

  `http://www.epm.ornl.gov/~walker/mpi/`

MPI/PVM sur Internet, suite

- MPI R/T
  http://www.cs.msstate.edu/mpirt/
- FAQ
  http://www.erc.msstate.edu/mpi/mpi-faq.html
- ROMIO
  http://www.mcs.anl.gov/home/thakur/romio/
- OOMPI
  http://www.cse.nd.edu/~lsc/research/oompi/
- Para++
  http://www.loria.fr/para++/
- Nexus
  http://www.mcs.anl.gov/nexus/
- PVM (et des liens sur les projets autour)
  http://www.epm.ornl.gov/pvm/pvm_home.html

Contact



Adresse

Frédéric DESPREZ

LIP ENS Lyon 46 Allée d'Italie

69364 Lyon Cedex 07

Tél 33 4 72 72 85 69 Fax 33 4 72 72 80 80

e-mail

Frederic.Desprez@inria.fr

URL

http://www.ens-lyon.fr/~desprez

Tutorial *Message Passing* (MPI/PVM/...)

http://www.ens-lyon.fr/~desprez/SCHEDULE/tutorials.html