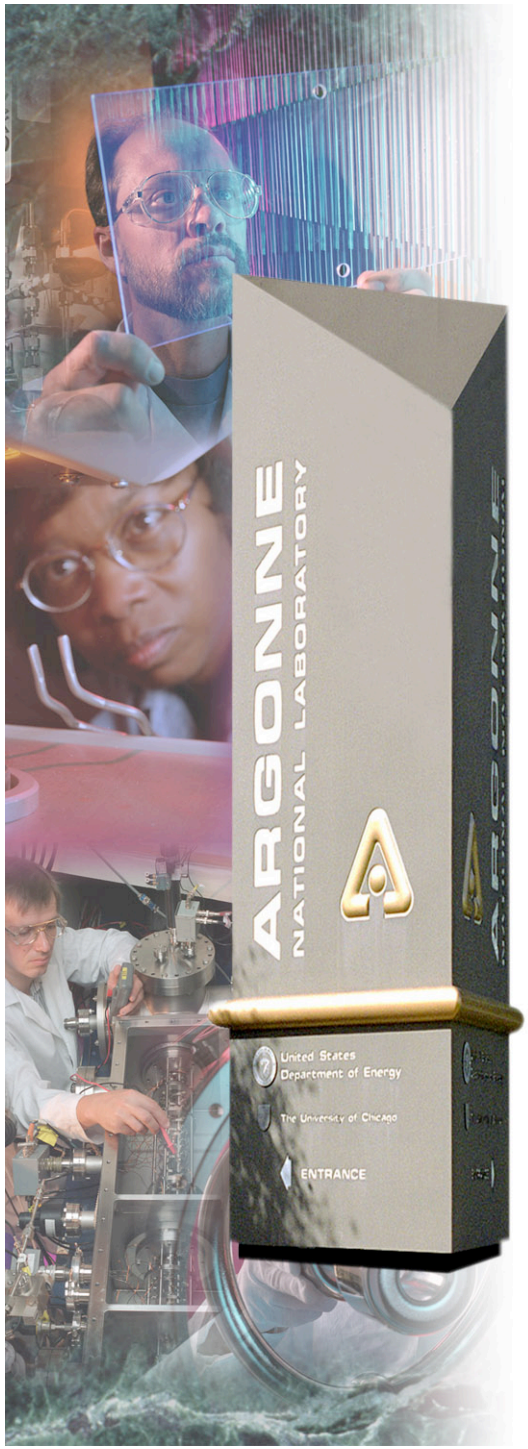


MPICH2:Nemesis : a high-performance MPI-2 implementation

Guillaume Mercier

www-unix.mcs.anl.gov/~mercierg

Argonne National Laboratory

A U.S. Department of Energy
Office of Science Laboratory
Operated by The University of Chicago

Office of Science
U.S. Department of Energy

Plan de l'exposé

- **Contexte et motivations**
- **MPICH2-Nemesis**
  - Vue d'ensemble
  - Communications par mémoire partagée
  - Communications réseaux
  - Optimisations
- **Résultats**
  - Mesures point-à-point
  - Applications "réelles"
- **Conclusion et perspectives**

CONTEXTE ET MOTIVATIONS

Contexte

- **MPI 1 :**
  - 1993-1994 : diffusion du standard
  - Peu après, beaucoup d'implémentations libres disponibles
    - *LAM/MPI*
    - *CHIMP*
    - *MPICH*
- **MPI-2 :**
  - 1997 : apparition du standard MPI-2
  - 2005-2006 : premières implémentations libres disponibles
    - *MPICH2 (fin 2004)*
    - *Open MPI (fin 2005)*
- **La compétition débute maintenant !**
  - Peu de résultats et comparaisons disponibles

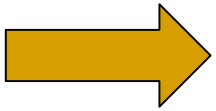
Motivations et objectifs

- **Analyse de l'architecture de MPICH2**
  - Amélioration du design
  - Amélioration des performances
- **Fournir une plate-forme d'expérimentation et d'analyse de MPI**
  - Performances maximales possibles
  - Surcoût sur l'implémentation de diverses fonctionnalités (ex `MPI_ANY_SOURCE`, multithreading, multi-méthode, etc)
- **Obtenir une implémentation de MPI**
  - Performante et extensible
  - Avec un faible surcoût
  - Vers une version «light» de MPI
- **Infirmer l'affirmation de Cray :**
  - « Un Send/Recv prend *au minimum* 1400 instructions »
  - Quel est le véritable nombre ?

Le «pari» perdu de Cray

- **Communications par mémoire partagée**
- **Utilisation de `MPI_COMM_WORLD`**
- **Envoi/réception d'un `double` (programme de type ping-pong)**
- **Temps de polling non comptabilisé**

	Instr # (Send)	Instr # (Recv)	TOTAL
MPICH2:shm	446	1233	1679
MPICH-GM	584	823	1407
LAM/MPI	550	599	1149
Open MPI	809	1979	2788



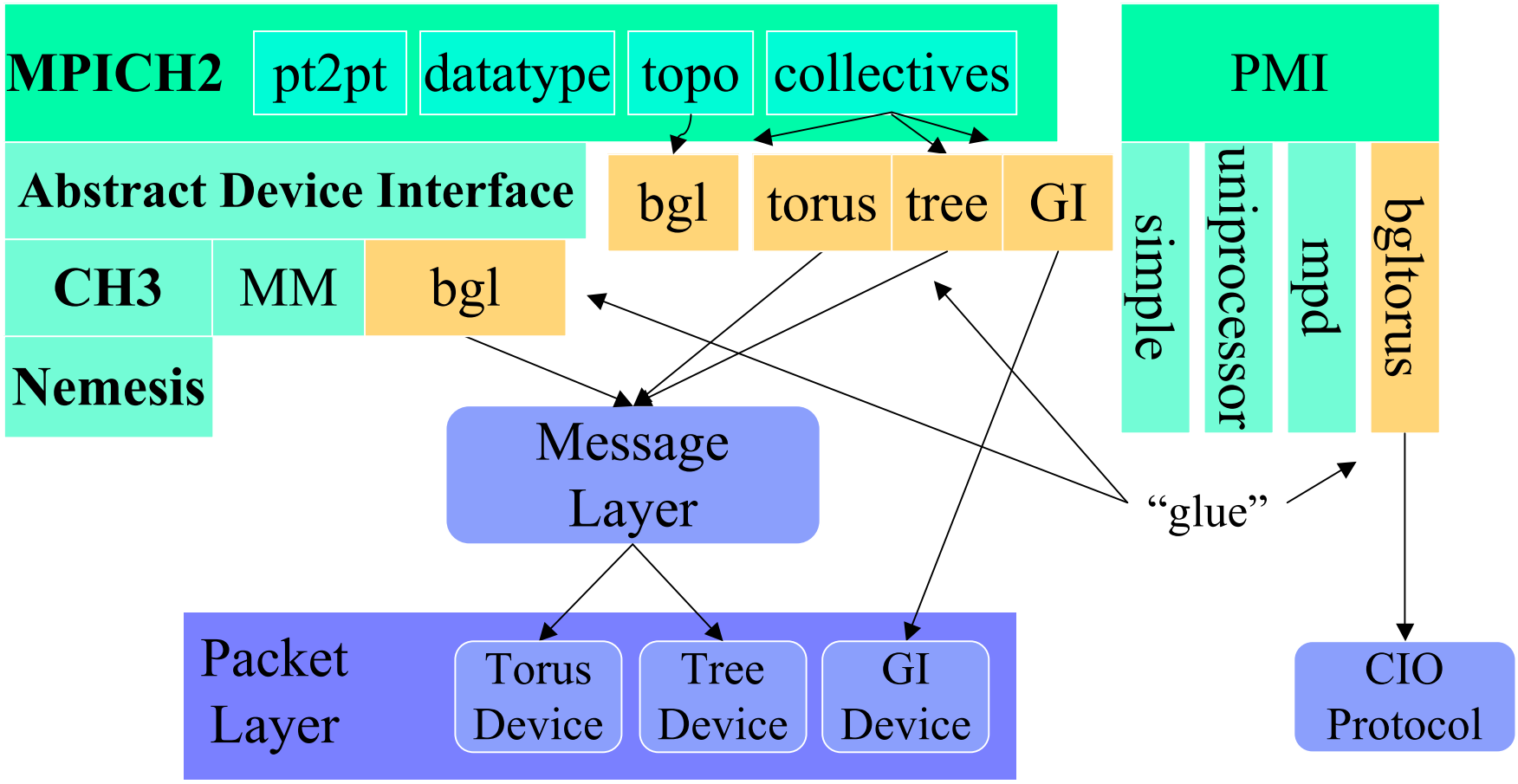
Confusion entre standard et implémentation !

ARCHITECTURE DE NEMESIS

Architecture de MPICH2:Nemesis

- **Un nouveau "Channel" pour MPICH2**
  - Utilise l'ADI3
  - Approche incrémentale : "Channel" puis "Device"
- **Priorité aux transferts utilisant la mémoire partagée**
  - Mémoire partagée : "first-class citizen"
  - Chemin critique le plus court possible dans ce cas
    - *Peu importe si le chemin critique des communications réseaux est plus long*
- **Les communications réseaux utilisent l'interface "mémoire partagée" plutôt que l'inverse**
  - Approche inverse de ce qui se fait habituellement (d'abord le réseau puis la mémoire partagée)
- **Nativement multi-méthode**

IBM BlueGene/L MPI Software Architecture



(slide based on one provided by IBM)

Principes de Nemesis

- **La mémoire partagée est le cas général**
  - Système basé sur des *Lock-free queues*
    - *Faible latence*
    - *Design extensible*
- **Multi-méthode**
  - Nouveaux réseaux faciles à ajouter (modules)
  - 5 fonctions de base à écrire
    - *init, finalize, send, poll_send, poll_recv*
  - Interface optionnelle pour RMA et opérations collectives (pour des performances améliorées)
    - *Suit l'approche standard de développement dans MPICH : un premier portage facile puis réglages de performance*

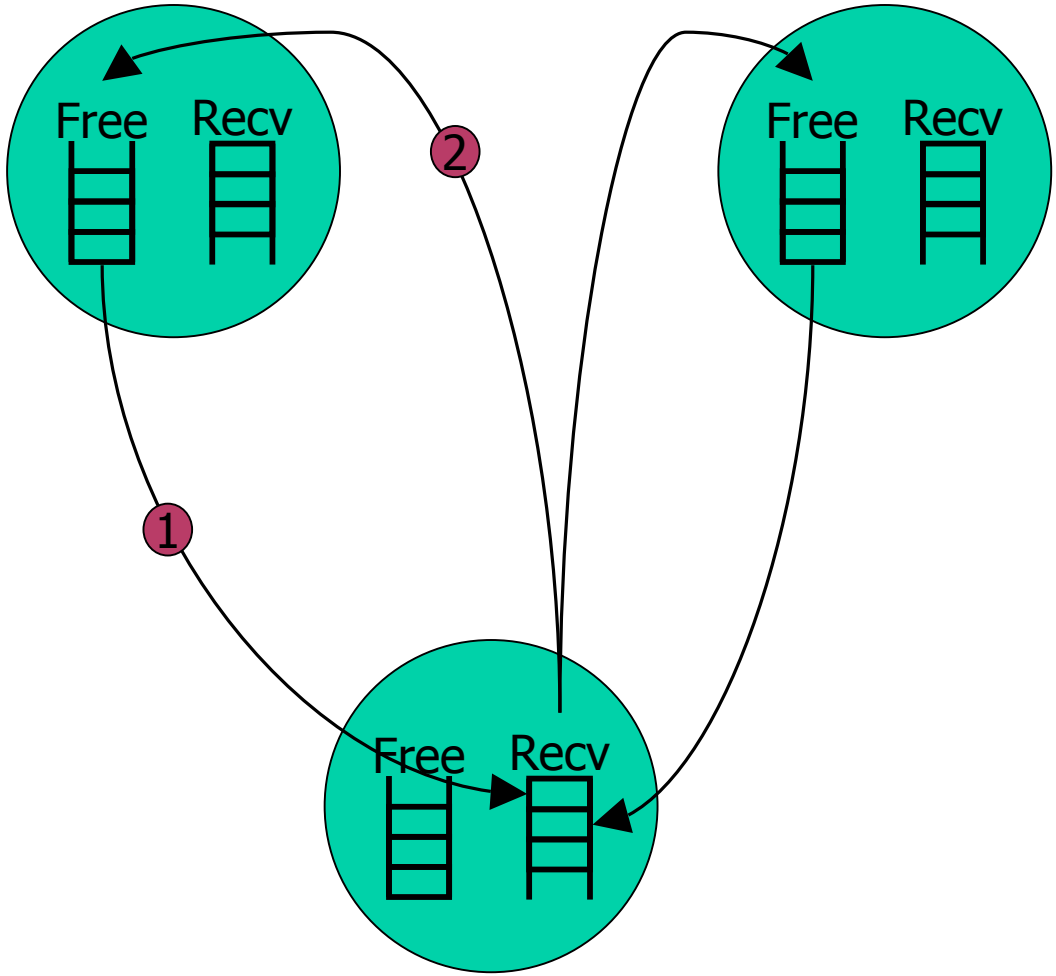
Communications avec Nemesis

- **Chaque processus possède plusieurs queues localisées en mémoire partagée qui traitent la plupart des communications:**
  - Une *Free Queue*
  - Une *Recv Queue*
- **Chaque processus possède aussi une paire "Fast Boxes" pour chaque processus de destination**
  - Utilisé pour les petits messages
  - Meilleure latence possible
  - Compte d'instructions le plus faible
- **Chaque processus possède aussi des modules réseaux pour les autres communications**
  - Ces modules utilisent les *Free et Recv Queues* du processus

Lock-Free Queues

- **Faible latence**
  - Pas de verrous
    - *Utilise des instructions atomiques comme compare-and-swap et swap*
  - Implementation simple
    - *Enqueue: 6 instructions, 1 L2 cache miss*
    - *Dequeue: 11 instructions, 1-2 L2 cache misses*
  - Le moteur de progression n'a qu'une seule queue à examiner
- **Très extensible**
  - Chaque processue n'a besoin que de deux queues quel que soit le nombre de processus
    - *La Recv queue*
    - *La Free queue*
  - Une seule queue pour le polling
- **Le même mécanisme est utilisé pour les transferts réseaux**
  - On souhaite conserver la même interface quel que soit le transfert
  - Les messages reçus du réseau sont mis dans la *recv queue*

Emission/réception avec des Lock-Free Shared Queues

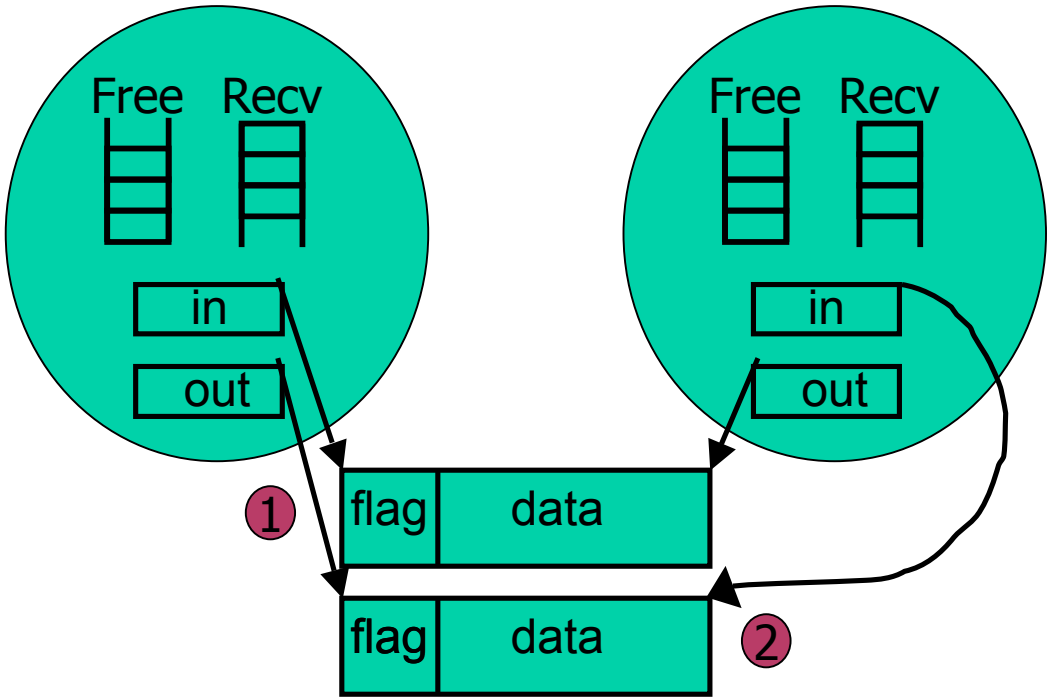


- **Le processus émetteur retire une entrée de sa propre free queue et la met dans la recv queue du destinataire**
- **Le processus récepteur enlève l'entrée de sa recv queue et la replace dans la free queue de l'émetteur**

Free Queue : locale à l'émetteur

- **Chaque processus possède une free queue qu'il utilise pour obtenir des entrées qui seront mises dans les recv queues des destinataires**
- **Un processus "dequeue", plusieurs "enqueue"**
- **Avantages et inconvénients :**
  - \+ Un processus ne peut pas être à court d'entrées à cause d'autres processus émetteurs
  - \+ Les entrées dans la free queue sont locales au processus émetteur (important pour les NUMA), donc remplir l'élément esr rapide
  - — Une distribution uniforme des entrées pour chaque processus peut conduire à des ressources perdues si des processus n'envoient pas de données
  - — Un processus lent à recevoir peut empêcher un autre processus d'envoyer des données

Emission/Réception avec les Fast Boxes

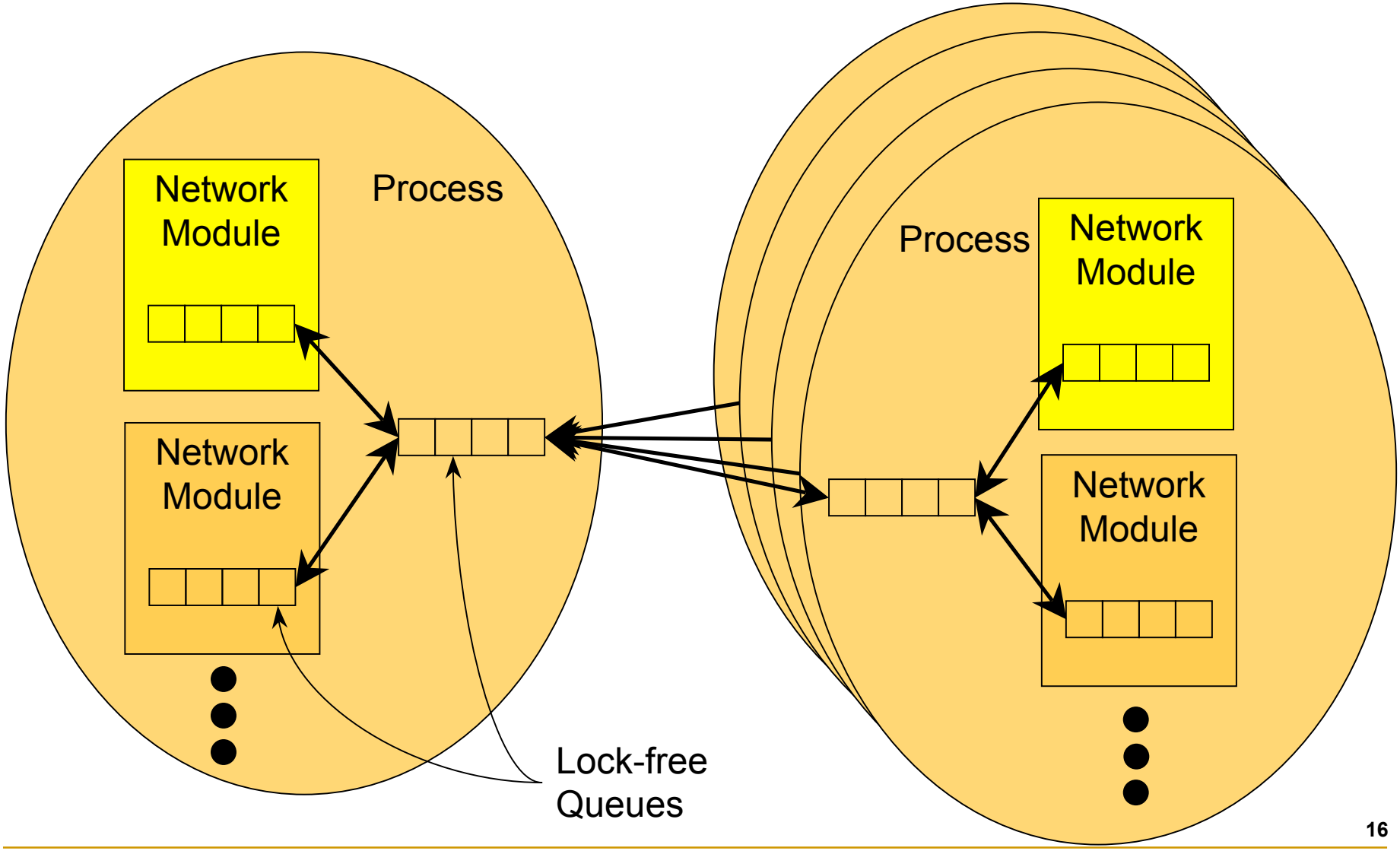


Fast Boxes en mémoire partagée

- **Le processus émetteur copie le message dans la fastbox et met à jour un flag**
- **Le processus récepteur vérifie le flag, copie le message dans le buffer utilisateur et met à jour le flag**

Un coup de canif pour l'extensibilité !

Modules réseaux et Shared Queues



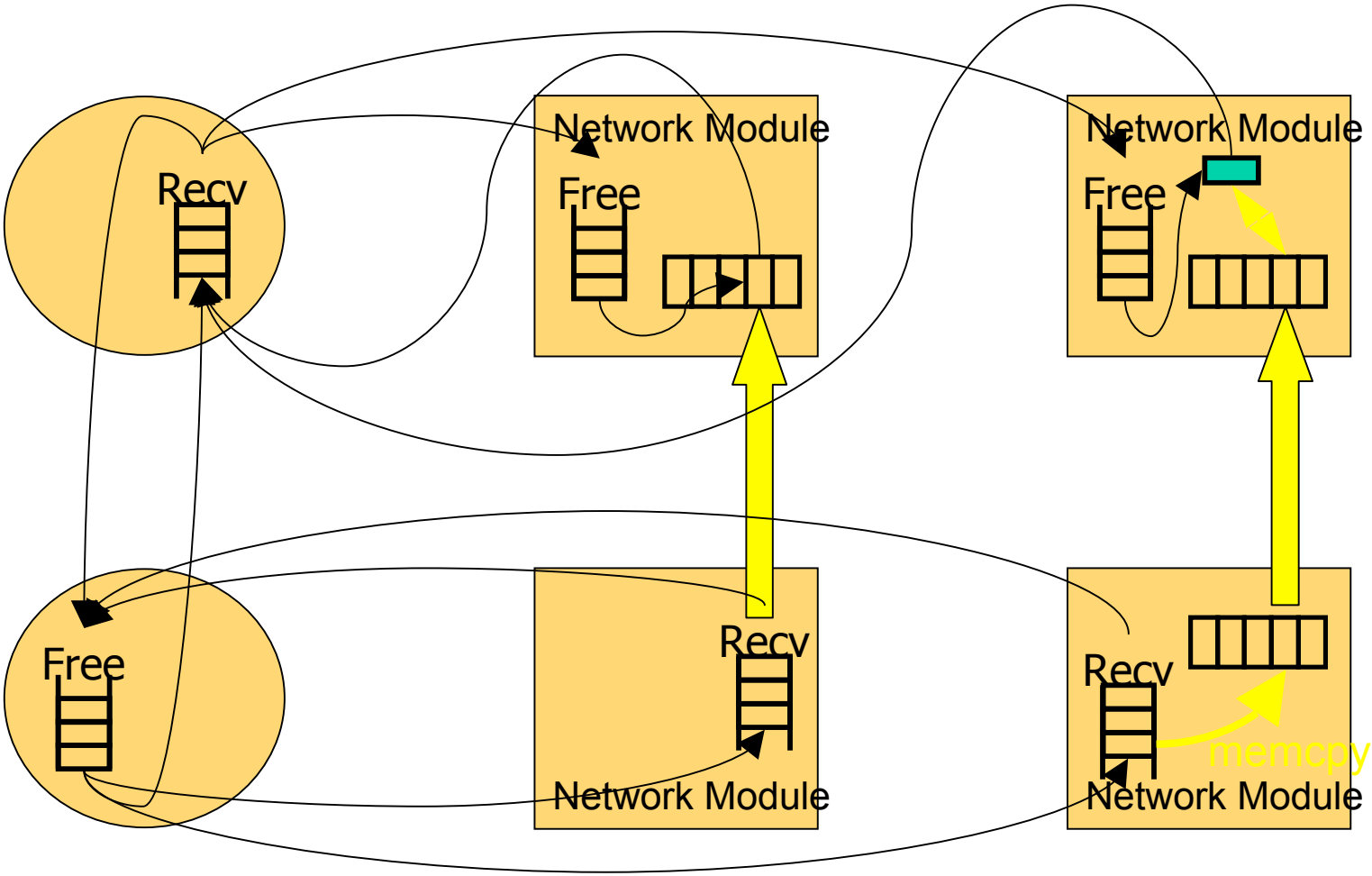
Réception Multi-méthode

- **Un module réseau reçoit un message et le poste dans la recv queue**
- **Où le module réseau peut-il prendre l'élément ?**
  - Cet élément doit être en mémoire partagée parce que d'autres processus SMP doivent y poster des entrées
  - Chaque module réseau possède sa propre *free queue*
- **Comment éviter des recopies ?**
  - Il faut enregister tous les éléments
    - *Les réseaux doivent être capables d'enregister de la mémoire partagée*
    - *Les données ne sont pas copiées : on met dans la queue le paquet reçu*

Réseaux Multiples : problèmes avec le Multi-méthode

- **Pour que les modules réseaux puissent envoyer/recevoir directement depuis/dans les éléments des queues, ces derniers doivent pouvoir être enregistrables par chaque module**
  - Est-ce que plusieurs réseaux sont capables d'enregister la même zone mémoire ?
    - *Peut-être pas*
- **Solution:**
  - Un seul réseau est autorisé à le faire
  - Les autres doivent copier
  - Hiérarchie implicite des communications
- **Motivation:**
  - Un seul réseau rapide dans la grappe
  - Les communications inter-grappes ne seront pas trop affectées par ces copies
  - Copies uniquement du côté émetteur

Réseaux multiples



Optimisations

- **Communications réseaux : *direct-send***
  - Évite une séquence *enqueue-dequeue*
  - Déroge à la règle “interface unique”
- **Utilisation de l’*inlining***
  - *gcc-dependent*
  - Diminution du compte d’instruction
  - “Applatissement” de la structure logicielle
    - Compromis entre maintenance et performances
    - Transformations sources à sources ?
    - Uniquement au niveau de Nemesis pour le moment
- **Refonte du moteur de progression des communications**
  - Travail au niveau de l’ADI3
  - Préfigure la version “Device” de Nemesis
  - Méthode de développement identique (analyse et instrumentation)

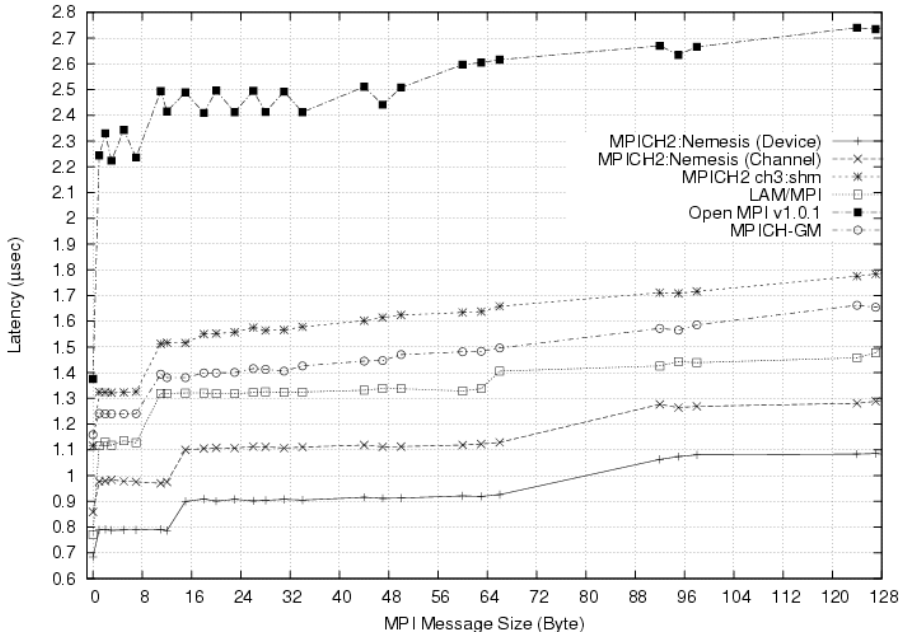
RESULTATS

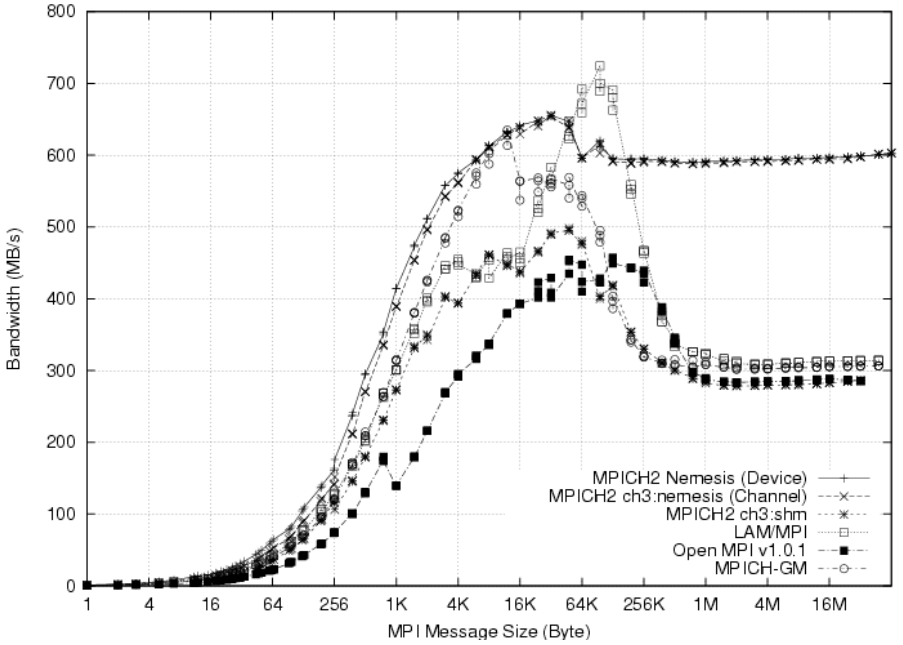
Résultats

- **Tests points à point**
  - Latence et débit
    - *Mémoire partagée*
    - *Myrinet/GM*
    - *Gigabit Ethernet/TCP*
  - Comparaisons avec :
    - *Channels existants dans MPICH2 (shm, sock)*
    - *LAM/MPI*
    - *Open MPI*
    - *MPICH-GM*
    - *YAMPII*
- **Applications**
  - HPL
  - NAS Parallel Benchmarks
  - Comparaisons avec
    - *Open MPI (GM)*
    - *MPICH-GM*
- **Compte d'instructions**
  - Mémoire partagée uniquement

Performances sur processeurs 32-bits

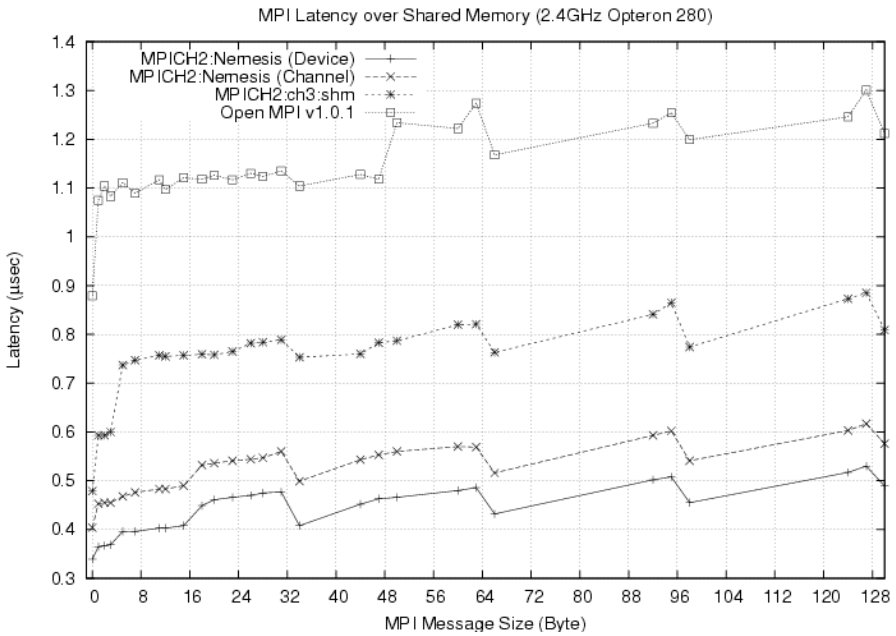
- **Communications utilisant la mémoire partagée**
- **Processeurs Intel Pentium 4 Xeon, 2 GHz**
- **Programme de test Netpipe :**
  - Latence minimale : 0,68 µs
  - Débit maximal : 650 Mo/s

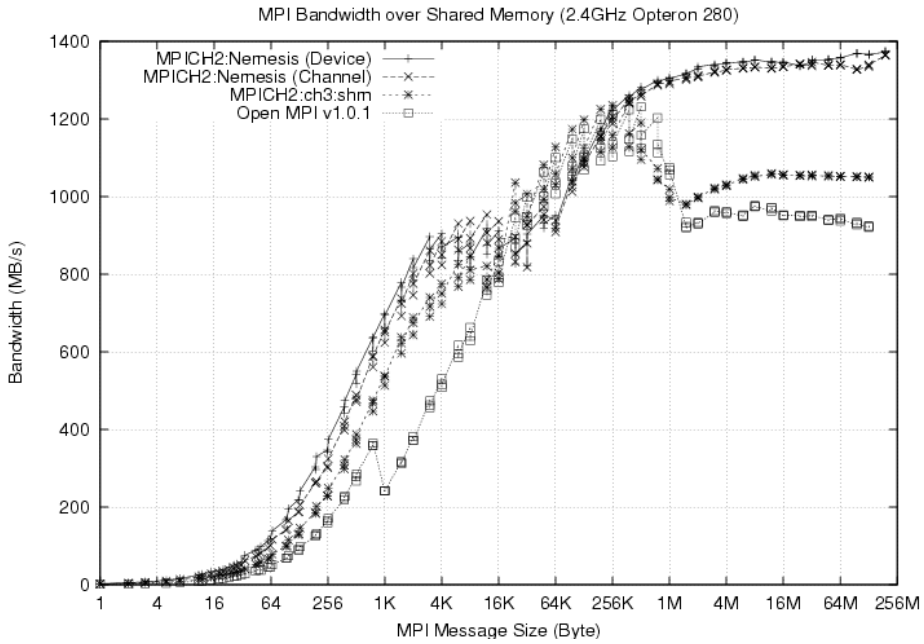




Performances sur processeurs 64-bits

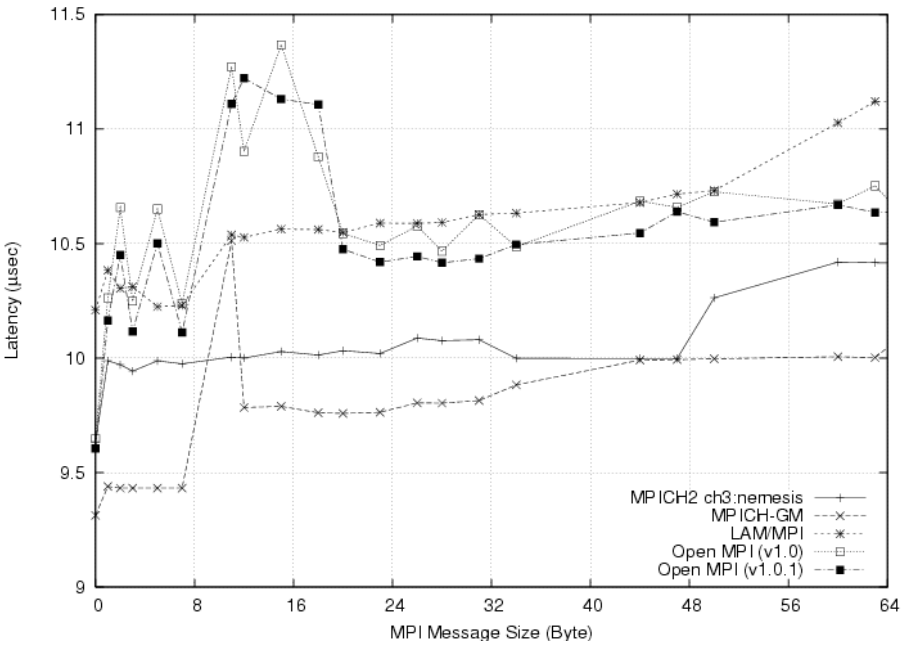
- **Communications utilisant la mémoire partagée**
- **Processeurs AMD Opteron 280, 2,4 GHz, dual core**
- **Programme de test Netpipe :**
  - Latence minimale : 0,34 µs
  - Débit maximal : 1,37 Go/s

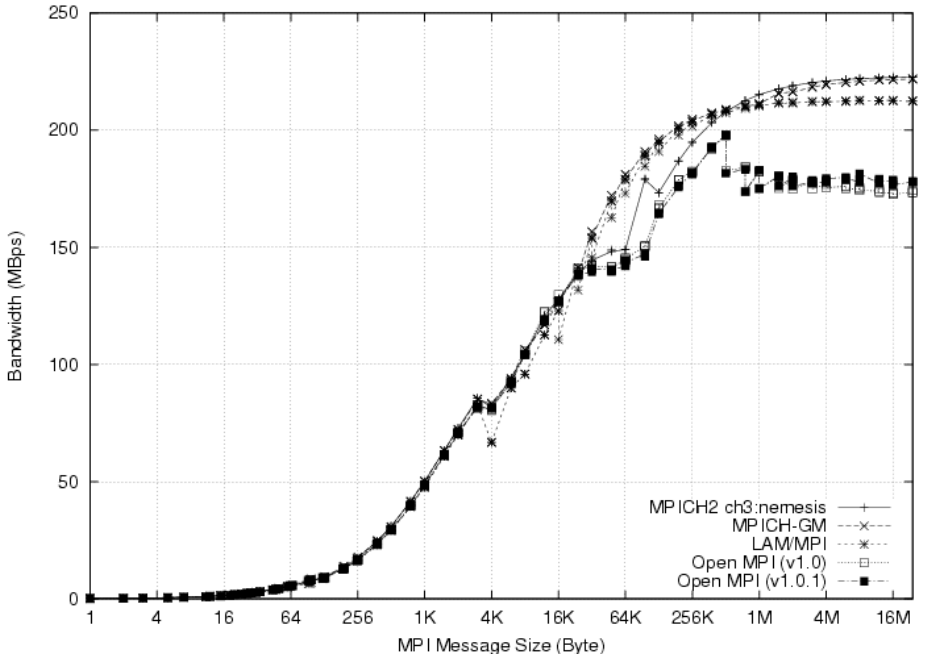




Performances sur processeurs 32-bits

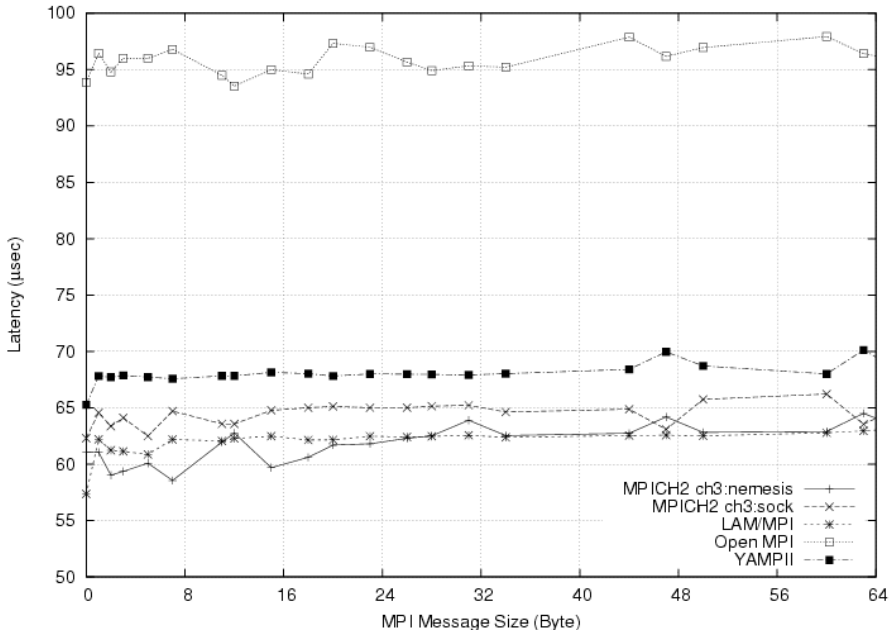
- **Communications utilisant Myrinet/GM**
- **Processeurs Intel Pentium 4 Xeon, 2 GHz**
- **Programme de test Netpipe**

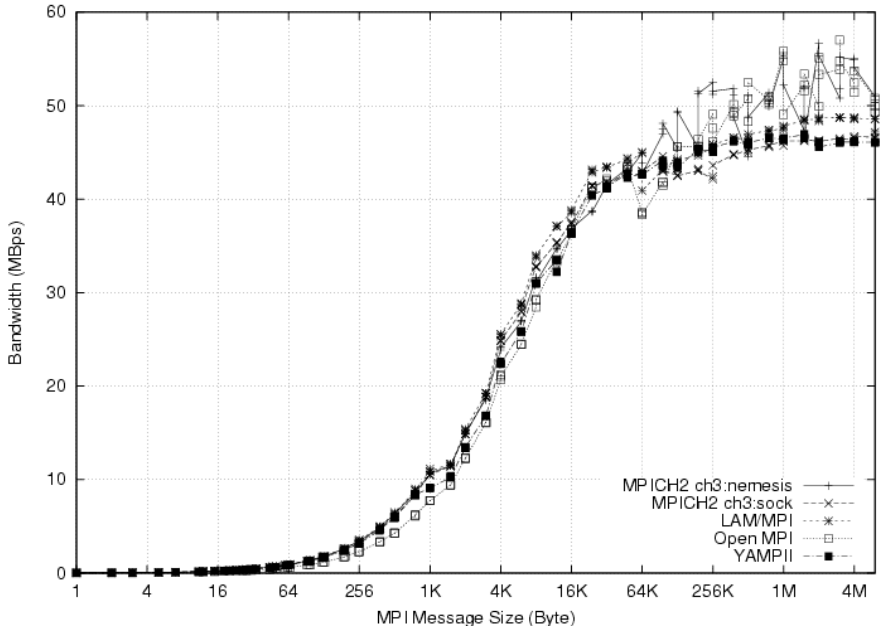




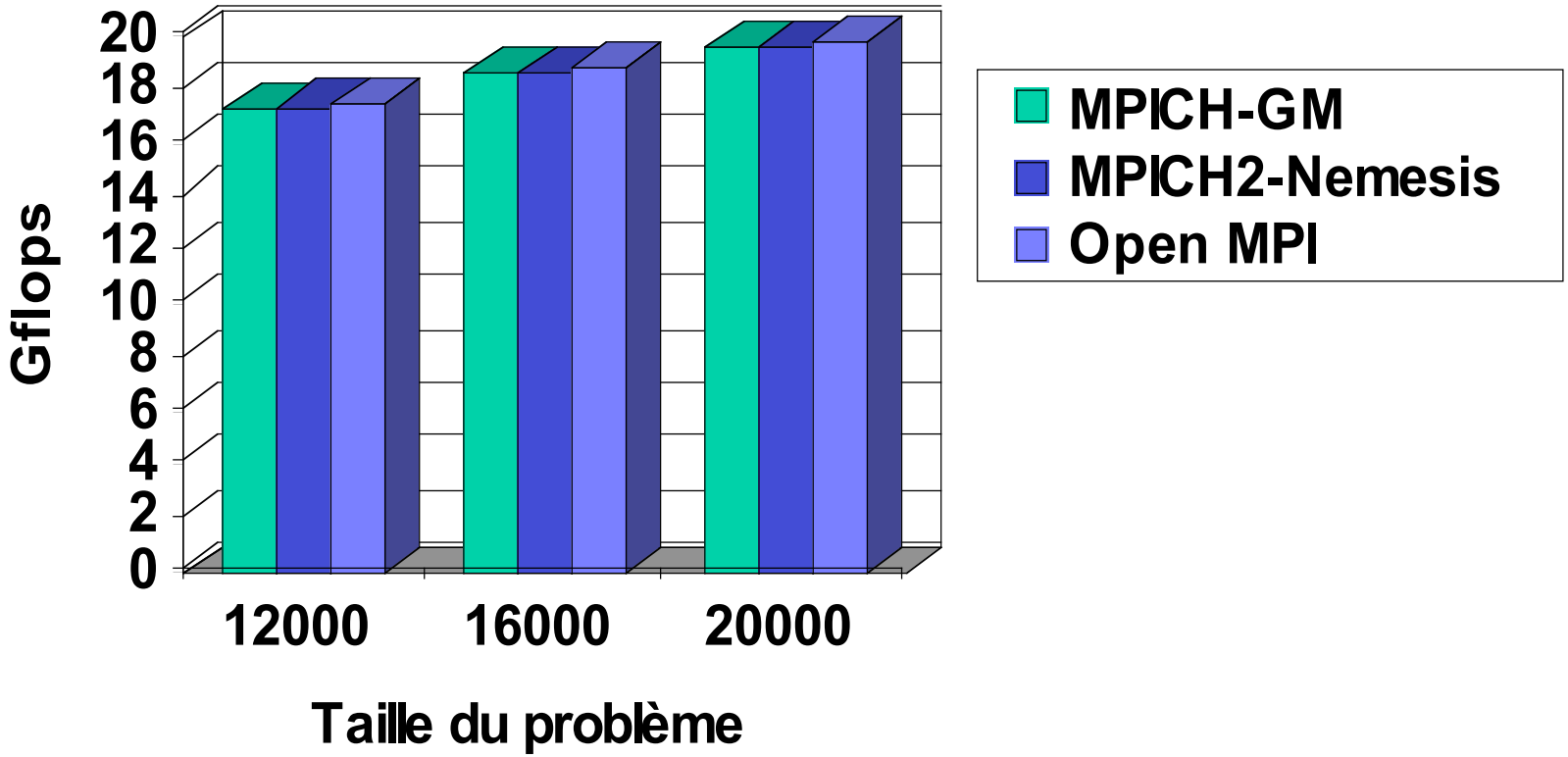
Performances sur processeurs 32-bits

- **Communications utilisant TCP/GigaBit Ethernet**
- **Processeurs Intel Pentium 4 Xeon, 2 GHz**
- **Programme de test Netpipe**



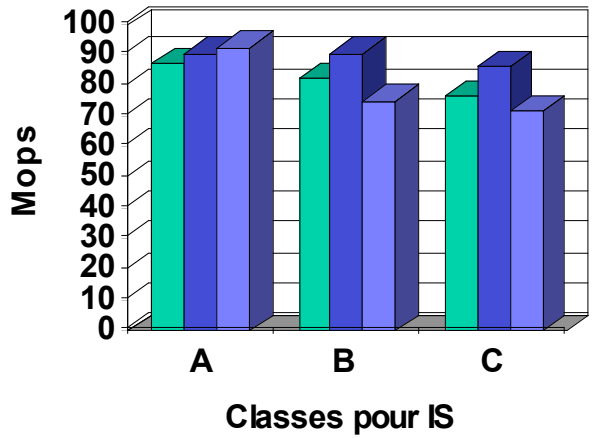


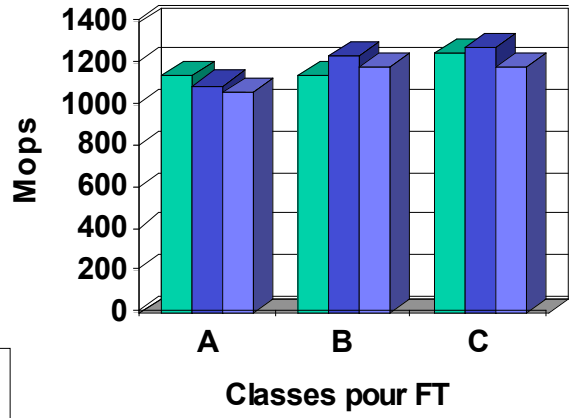
High-Performance Linpack

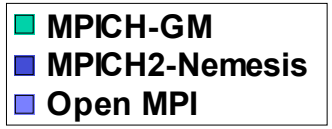


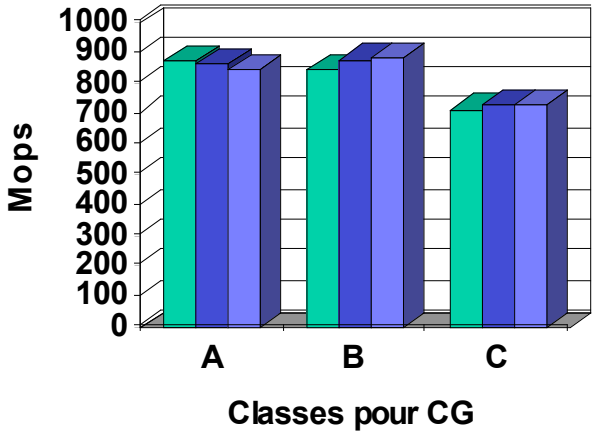
HPL sur 4 nœuds bi-processeurs – réseau Myrinet/GM

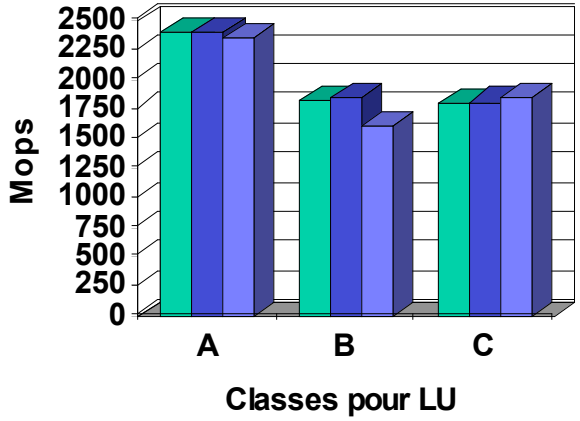
NAS Parallel Benchmarks











Compte d'instructions

- **Communications par mémoire partagée**
- **Utilisation de `MPI_COMM_WORLD`**
- **Envoi/réception d'un `double` (programme de type ping-pong)**
- **Temps de polling non comptabilisé**

	Instr # (Send)	Instr # (Recv)	TOTAL
MPICH2:shm	446	1233	1679
MPICH-GM	584	823	1407
LAM/MPI	550	599	1149
Open MPI	809	1979	2788
MPICH2:Nemesis (channel)	238	751	989
MPICH2:Nemesis (device)	238	250	488

Encore loin du minimum ! (~250 instructions)

Conclusion

- **Performances de Nemesis**
  - Channel MPICH2 le plus rapide
  - Meilleure latence actuelle dans MPI (mémoire partagée
  - Meilleur compte d'instruction actuel (faible surcoût)
- **Marge de progression « importante » pour le compte d'instruction**
- **Performances avec applications « réelles » du même niveau que la concurrence**
  - Léger avantage (à confirmer)

Perspectives

- **Diffusion de Nemesis avec MPICH2**
  - Channel «experimental» dans la prochaine release
  - Channel «par défaut» dans la release suivante
- **Implémenter de nouveaux modules réseaux**
  - MX
  - Quadrics
- **Implémenter la version Device**
  - Remplacement (partiel ? Total ?) de l'ADI3
- **Proposition de Nemesis dans le cadre de HPCS**
- **Amélioration des performances**
  - Nombre d'instructions encore trop important par rapport au nombre minimal possible (~250 instructions)
- **Tests avec d'autres applications**