De l'interaction des communications et de l'ordonnancement de threads au sein des grappes de machines multi-cœurs

François Trahay

Sous la direction de Raymond Namyst et Alexandre Denis

Équipe-projet Runtime

LaBRI

UNIVERSITÉ BORDEAUX 1 Sciences Technologies

INRIA FUTURS

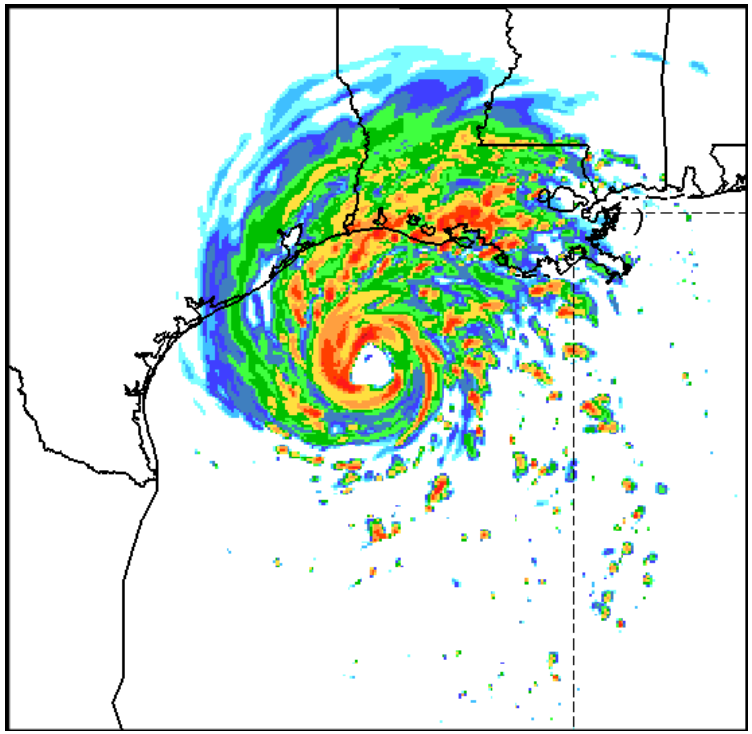
Financé par ANR

Plan

- Contexte de travail
- Problématiques
- Proposition
- Évaluation
- Conclusion et perspectives

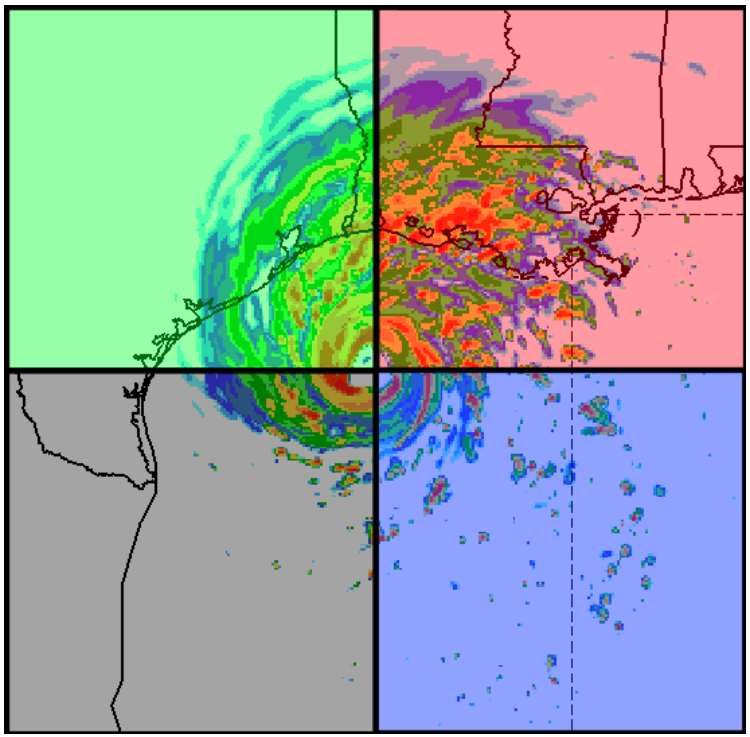
Exigences du calcul scientifique

- Simulations numériques
  - Exemples :
    - Météorologie, climatologie
    - Sismologie, astrophysique, etc.
  - Beaucoup de calculs
  - Manipulation de masses de données
- Accroissement des besoins de puissance de calcul
  - Aller toujours plus vite
  - Traiter des problèmes plus gourmands en ressources



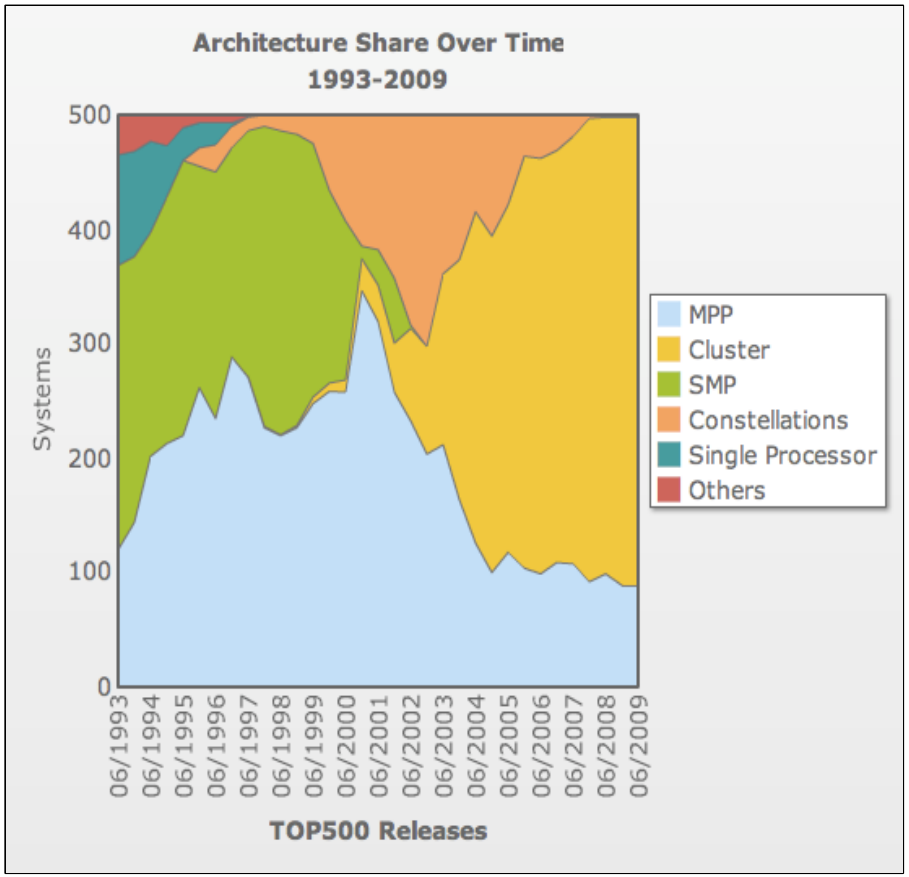
Comment traiter des problèmes toujours plus gourmands ?

- Parallélisation des algorithmes
  - Répartition du travail sur plusieurs processeurs
  - Chaque processeur traite un sous-ensemble du problème
  - Communication entre les processeurs pour mettre en commun les contributions



Plates-formes d'expérimentation

- Supercalculateurs
  - Machines constituées de matériel spécifique
  - Performances élevées
  - Coût important
- Grappes de calcul
  - Dès les années 1990
  - Ensemble de machines “classiques”
  - Interconnectées par un réseau rapide
  - Rapport performance/coût intéressant

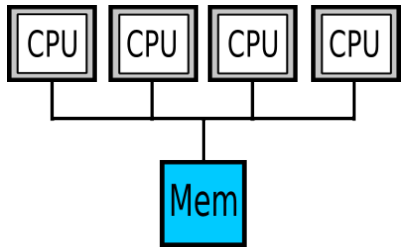


Évolutions des grappes
Les réseaux hautes performances

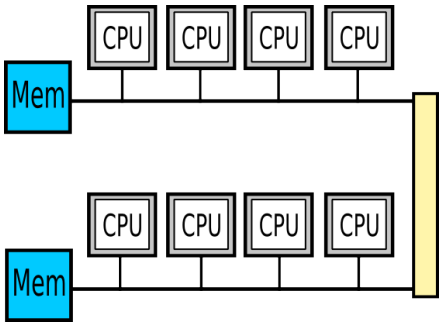
- Nombreuses technologies
  - InfiniBand, Myrinet, Ethernet
  - Faible latence (~ 1 µs) et bande-passante élevée (~ 40 Gbit/s)
  - Interfaces de communication spécifiques
    - Verbs/InfiniBand, MX/Myrinet, TCP/Ethernet
- MPI : Interface de communication standard
  - Abstraction des interfaces de bas niveau
- Nombreuses implémentations
  - Spécifiques à une technologie : BlueGene MPI, MVAPICH, MPICH2-MX,...
  - Génériques : MPICH2, OpenMPI, …
- Performances brutes proches des performances du matériel

Évolution des nœuds de grappes

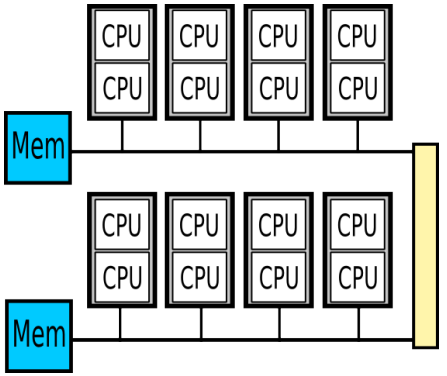
- Multi-processeurs symétriques (SMP)
  - Passage à l'échelle problématique



- Machines NUMA
  - Accès à la mémoire non-uniformes

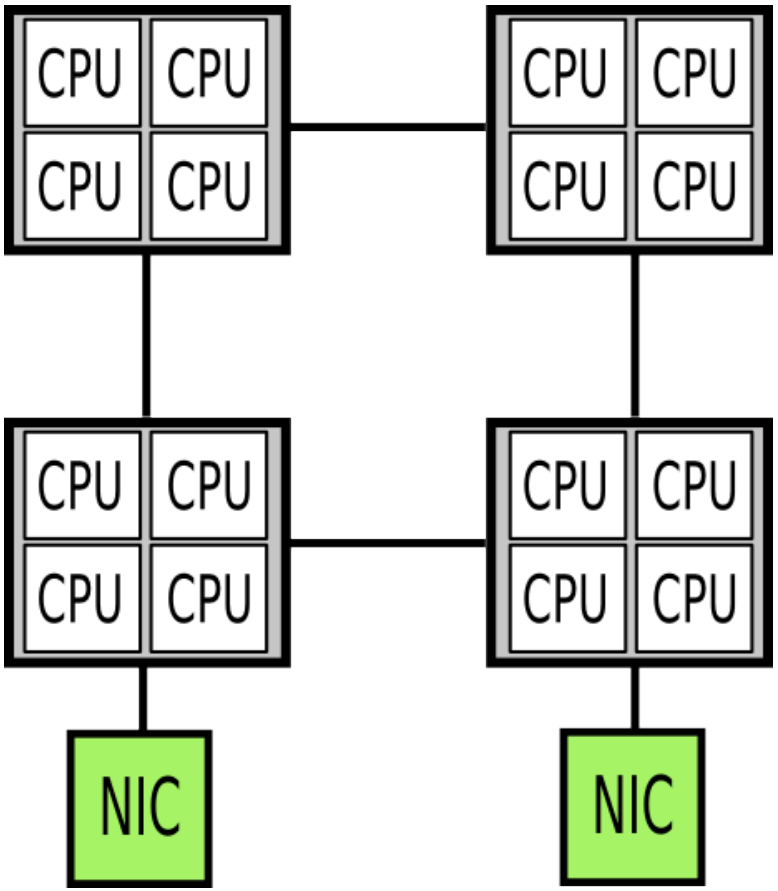


- Machines NUMA multi-cœurs
  - Augmentation du nombre de cœurs



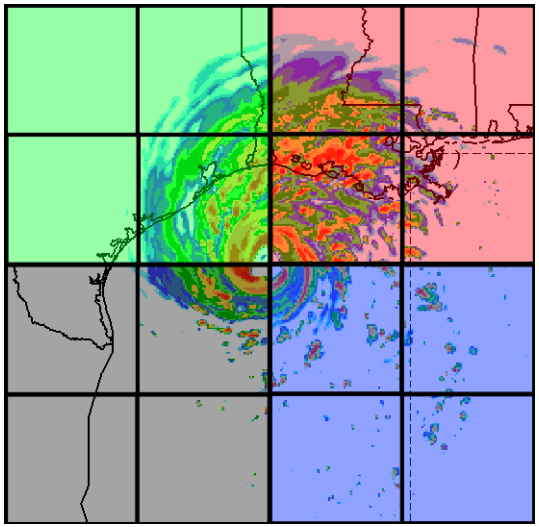
Évolution des nœuds de calcul
Les défis à relever

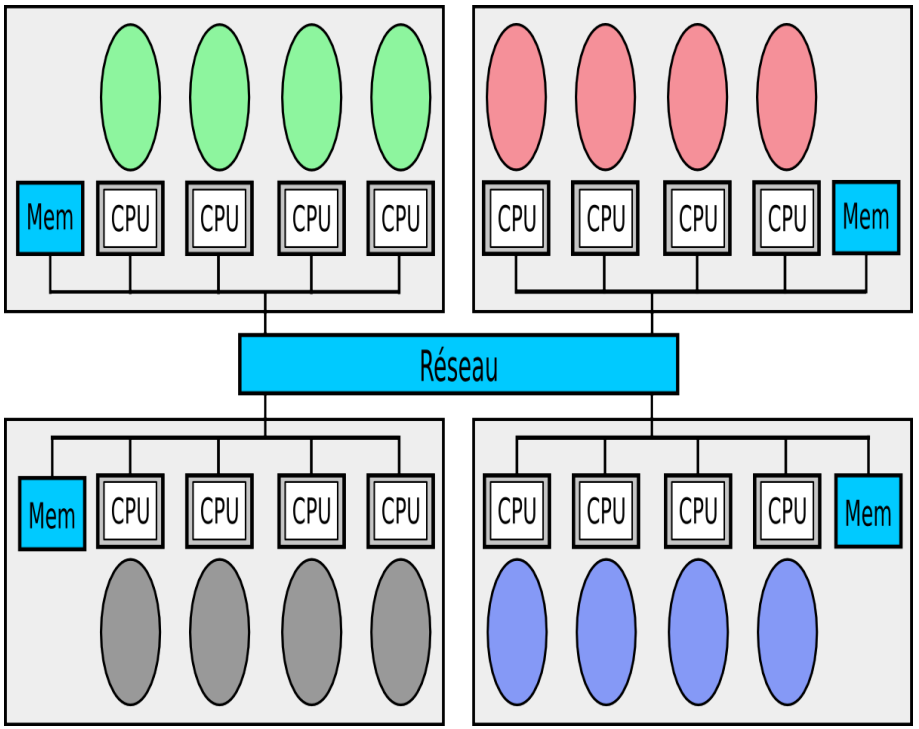
- De plus en plus de cœurs par nœud de calcul
  - Il y a 5 ans : < 4 cœurs par nœud
  - Aujourd'hui : > 16 cœurs par nœud
  - Demain : > 100 cœurs par nœud ?
- Le nombre de cartes réseaux augmente peu
  - Partage des ressources de communication



Exploitation d'une grappe de calcul
L'approche classique

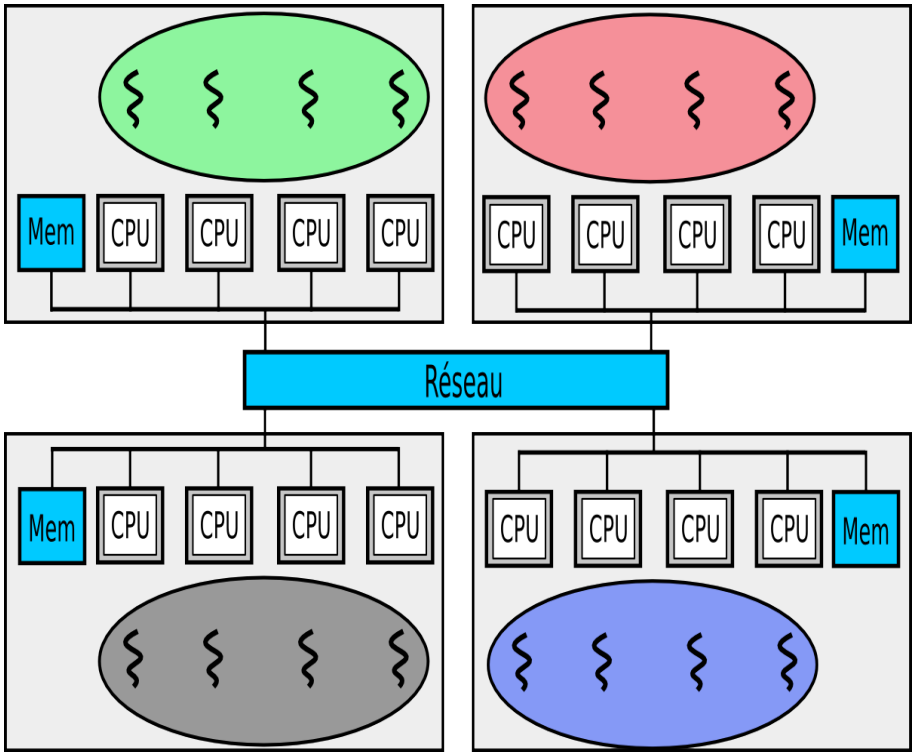
- Approche classique : un processus MPI par cœur
  - Intra-nœud : Mémoire partagée
  - Inter-nœud : Réseau
- Problème de passage à l'échelle
  - Consommation mémoire
  - Répartition de la charge
  - Accès aux réseaux

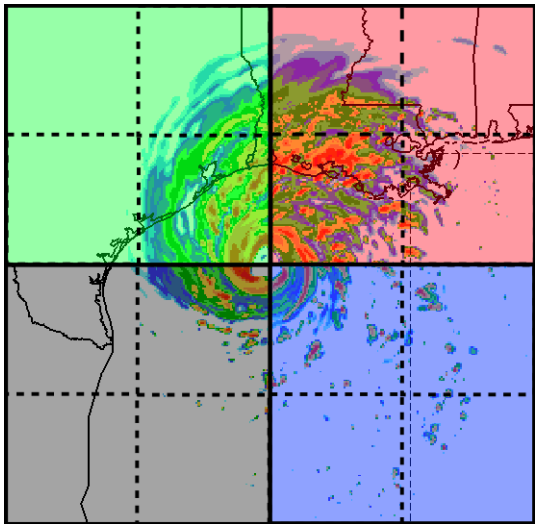




Exploitation d'une grappe de calcul
L'approche hybride

- Approche hybride : un processus MPI par nœud
  - Threads (PThreads, OpenMP, TBB, ...)
  - Communication par le réseau
- Avantages :
  - Synchronisation intra-nœud rapide
  - Vision globale des communications d'un nœud





Plan

- Contexte de travail
- Problématiques
  - Accès concurrents
  - Recouvrement des communications par du calcul
- Proposition
- Évaluation
- Conclusion et perspectives

Accès concurrents

- Développement du modèle MPI + threads
- Problème : la bibliothèque de communication doit supporter les accès concurrents

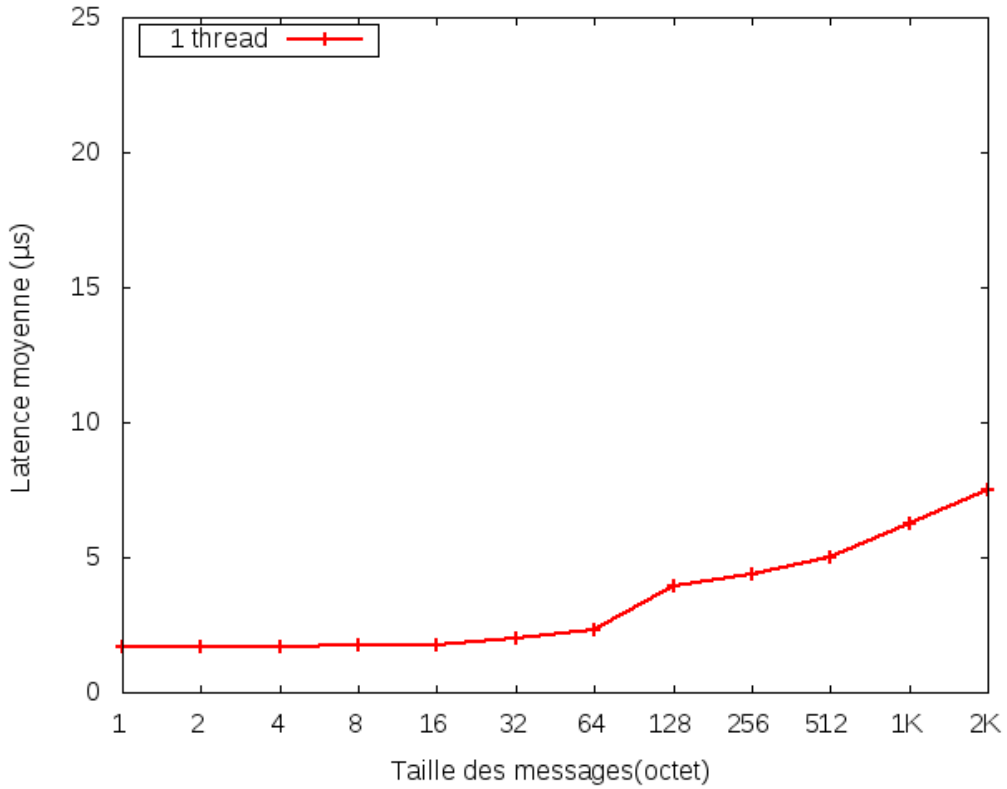
  → Arbitrage des accès au réseau

  → Arbitrage des accès aux structures de données

- Les implémentations MPI ne sont pas prêtes

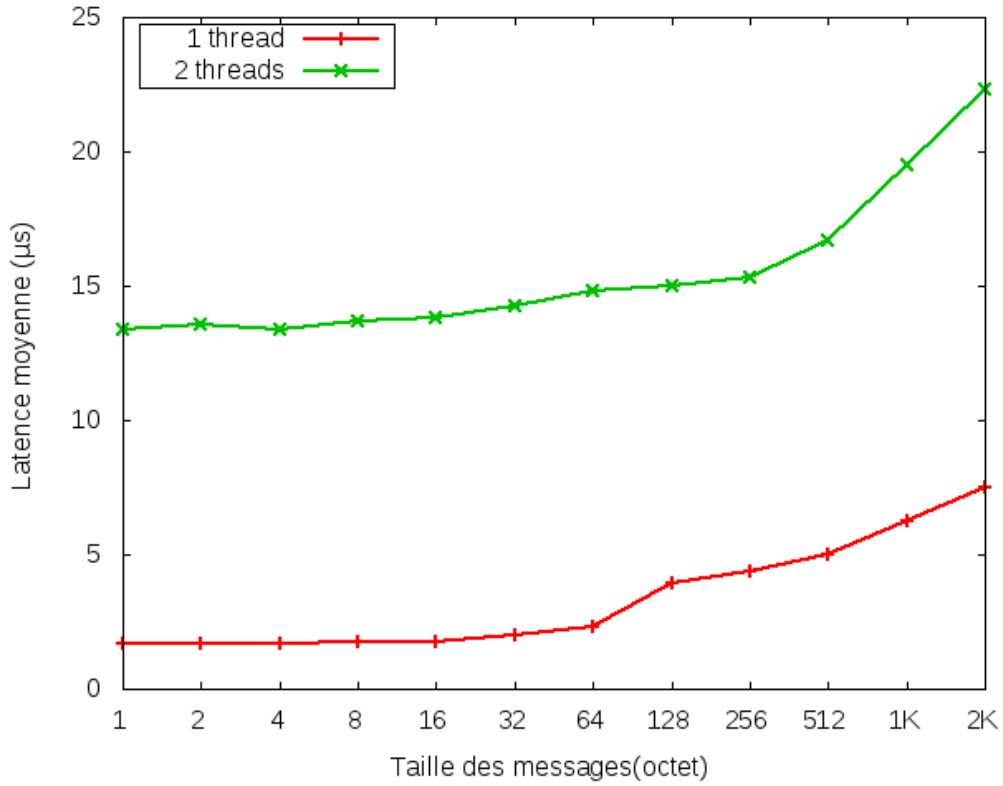
Accès concurrents exemple

- Ping-pong classique avec MPI
  - Temps de transfert moyen entre deux machines



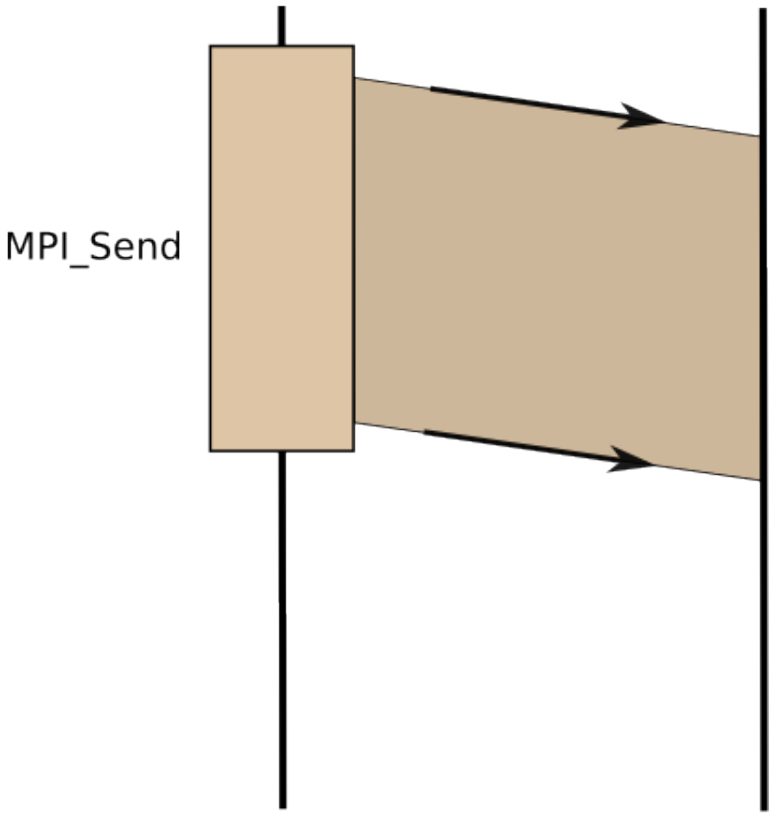
Accès concurrents exemple

- Ping-pong classique avec MPI
  - Temps de transfert moyen entre deux machines



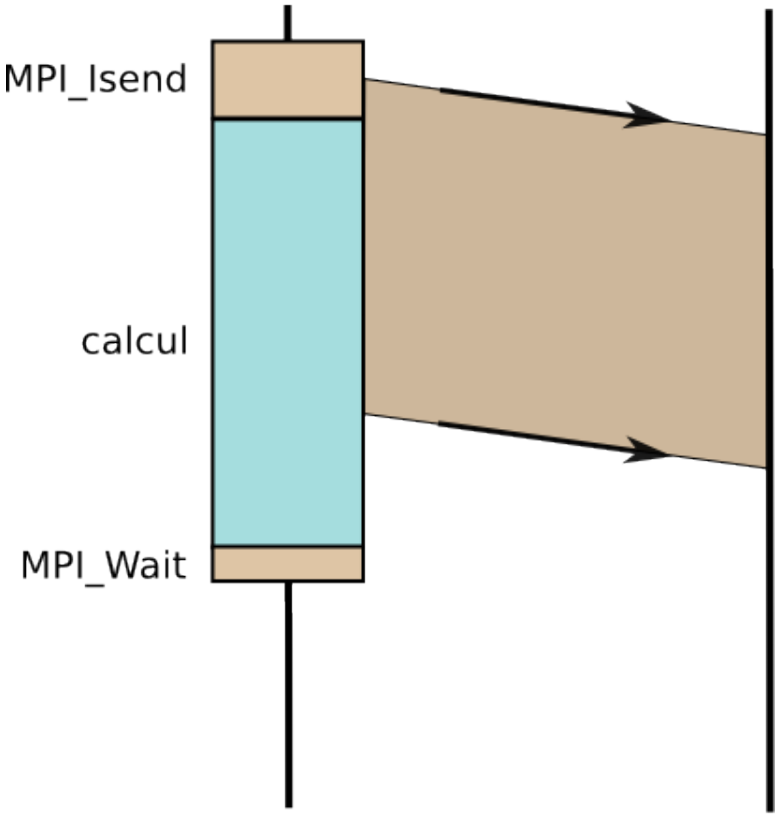
Recouvrement des communications par du calcul

- Primitives de communications non-bloquantes
  - Initialisation de la communication
  - Communication en arrière-plan
  - Possibilité de “cacher” le coût des communications



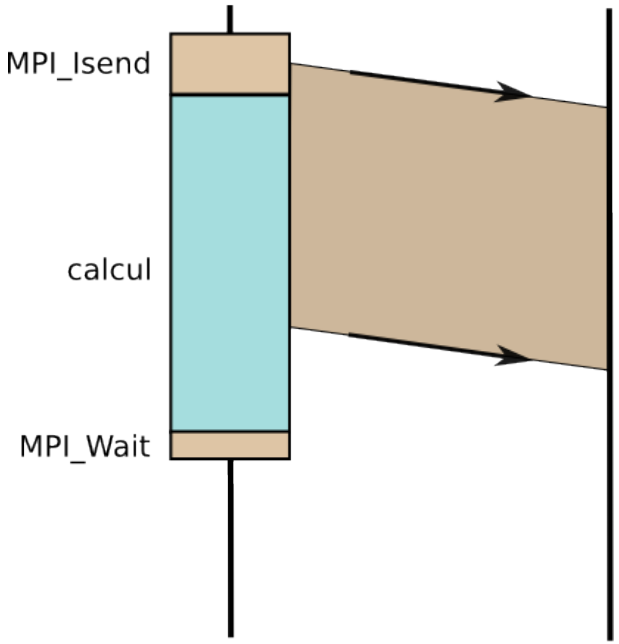
Recouvrement des communications par du calcul

- Primitives de communications non-bloquantes
  - Initialisation de la communication
  - Communication en arrière-plan
  - Possibilité de “cacher” le coût des communications

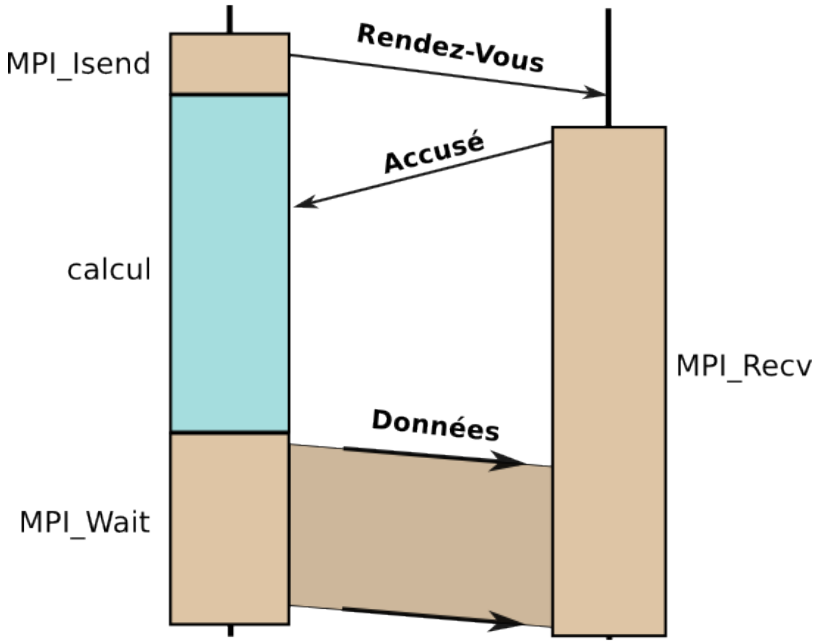


Recouvrement des communications par du calcul

- Primitives de communications non-bloquantes
  - Initialisation de la communication
  - Communication en arrière-plan
  - Possibilité de “cacher” le coût des communications



Comportement attendu

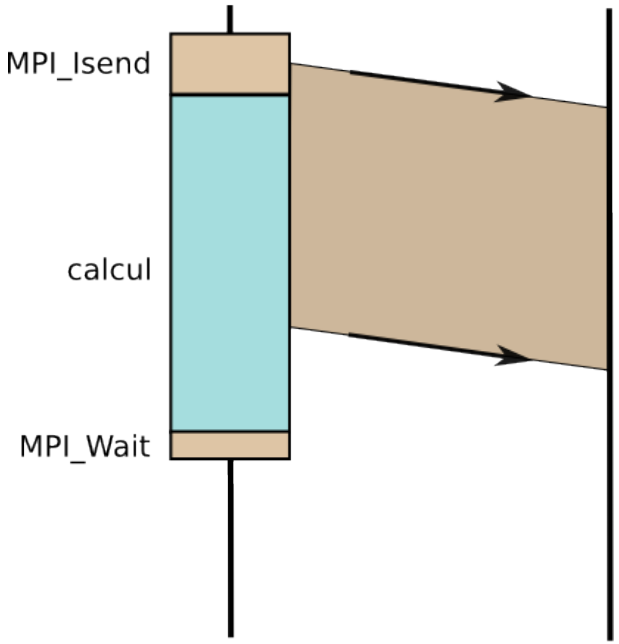


Comportement obtenu

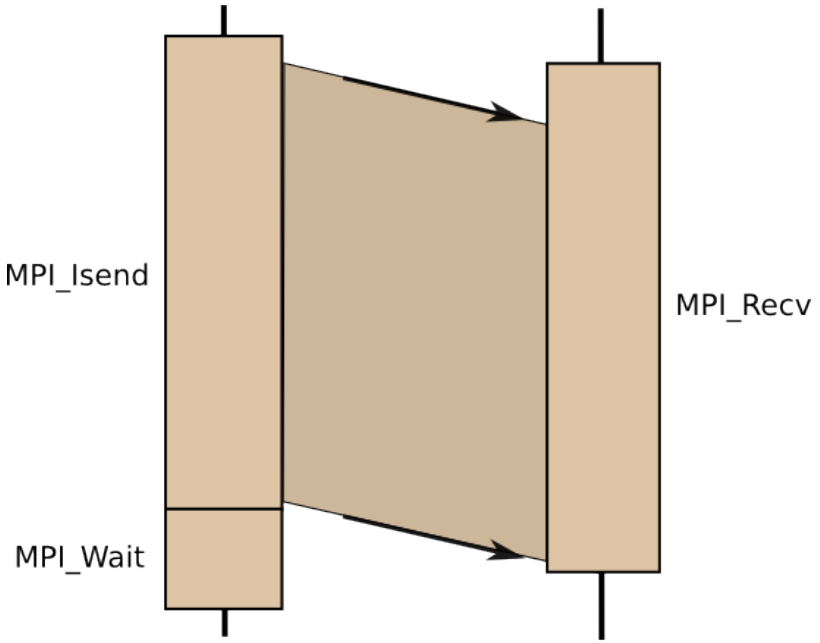
Protocole de Rendez-vous

Recouvrement des communications par du calcul

- Primitives de communications non-bloquantes
  - Initialisation de la communication
  - Communication en arrière-plan
  - Possibilité de “cacher” le coût des communications



Comportement attendu



Comportement obtenu

Transfert du message vers la carte réseau

Problèmes des implémentations actuelles

- Implémentations orientées MPI “classique”
  - Conception suivant la philosophie du MPI “classique” (ie. sans thread)

    → Support des threads *a posteriori*
- Objectif principal de ces implémentations :
  - Avoir les meilleures performances brutes
- Les applications pallient les insuffisances
  - Threads de communication
  - Accès à la bibliothèque en exclusion mutuelle

    → Report des problèmes sur les couches supérieures

Les défis à relever

- Progression des communications
  - Cacher le coût des communications
- Support des accès concurrents
  - Support des applications multi-threadées
- Parallélisation des traitements
  - Profiter des cœurs disponibles

Plan

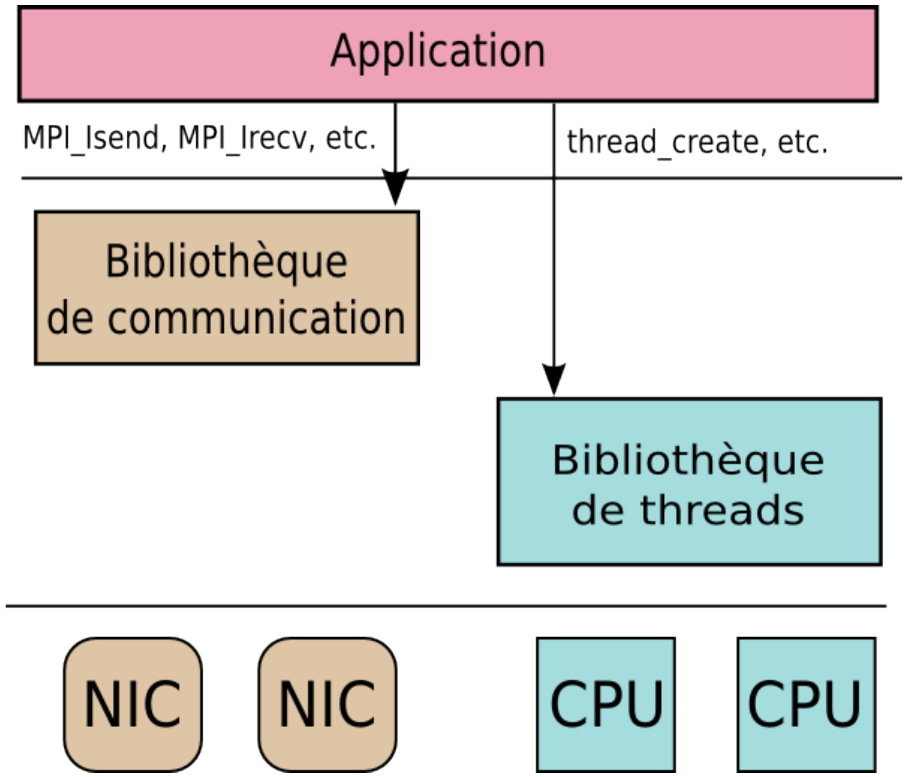
- Contexte de travail
- Problématiques
- Proposition
- Évaluation
- Conclusion et perspectives

Vers une collaboration entre communications et threads

- Penser au multi-threading dès la conception
  - Support des accès concurrents
  - Progression des communications
- Avoir une vision globale de l'état d'un nœud
  - Charge des processeurs
  - État des cartes réseau

Proposition
Vue d'ensemble

- Pile logicielle typique MPI
  - Bibliothèque de communications pour le réseau
  - Bibliothèque de threads pour les CPU
- Pas d'interaction

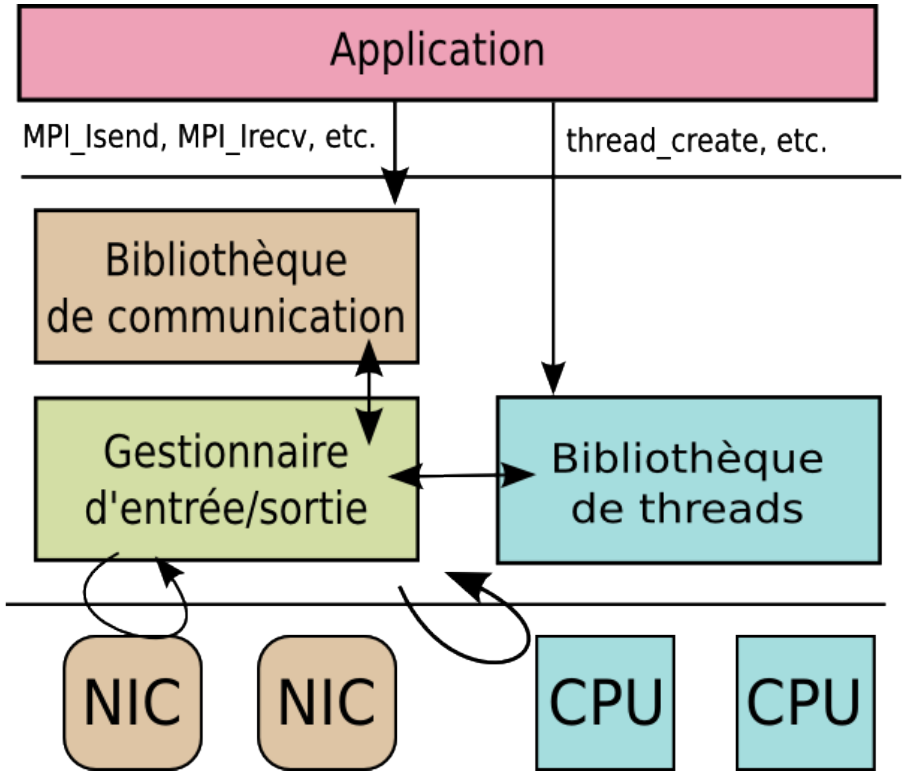


Proposition
Vue d'ensemble

Collaboration entre les bibliothèques

- Vision globale des entrées/sorties
- Vision globale de la charge

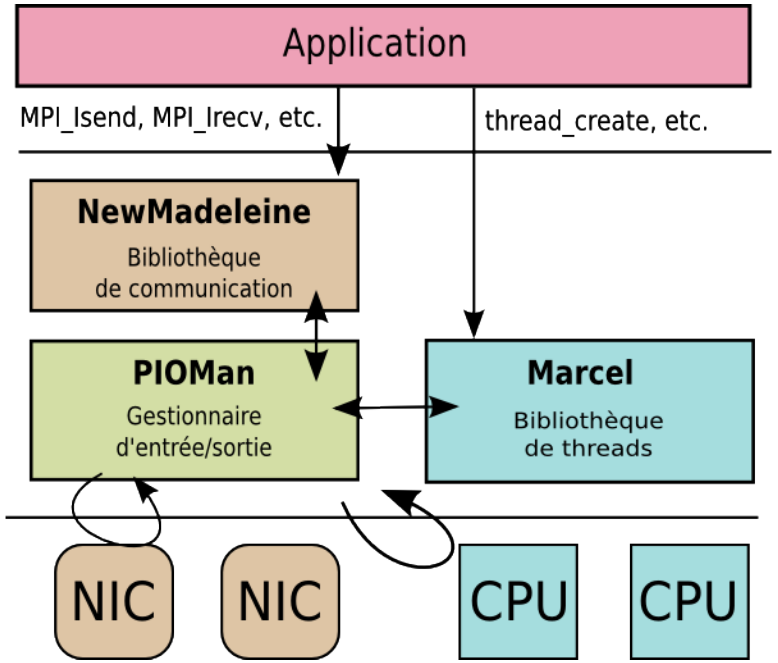
→ Exploiter intelligement ces informations



Implémentation
La suite logicielle PM2

- NewMadeleine : bibliothèque de communication [E. Brunet 08]
  - Optimisations sur les flux de communication
- Marcel : bibliothèque de threads

  [Namyst 97, Danjean 04, Thibault 07]
  - Threads de niveau utilisateur
- PIOMan : gestionnaire d'entrée/sortie
  - 6000 lignes de code
  - Fournit un service de détection des événements
  - Répartit les traitements sur les cœurs disponibles

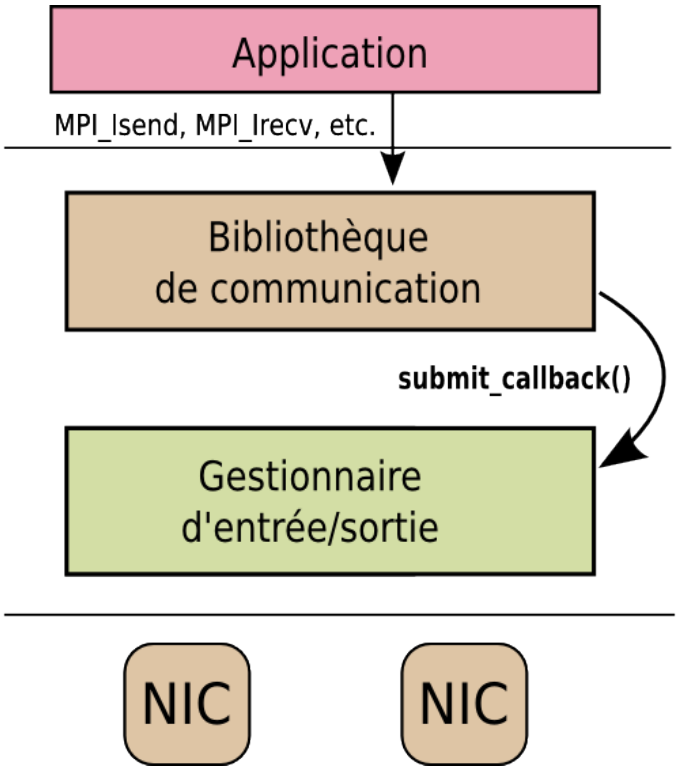


Proposition

- Progression des communications
- Support des accès concurrents
- Parallélisation des traitements

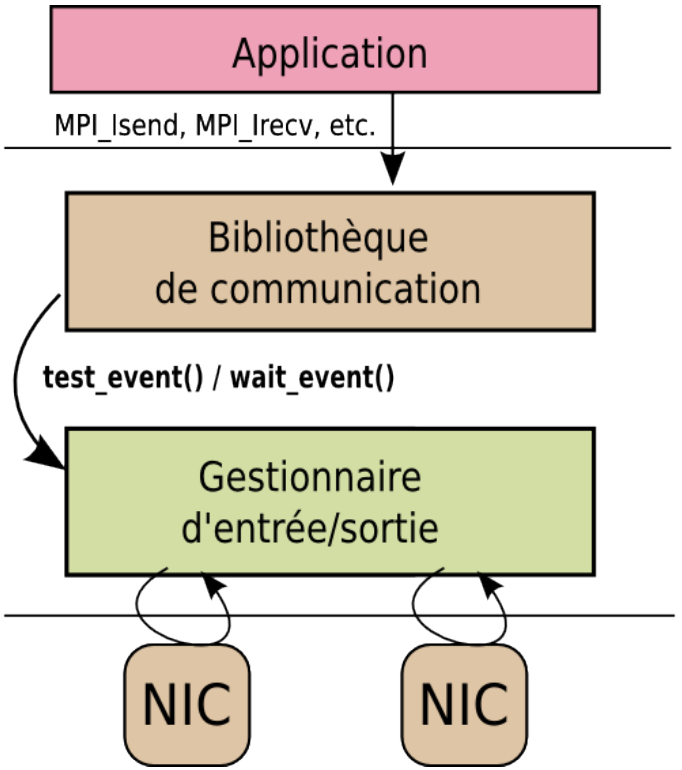
Détection des événements réseau

- La bibliothèque de communication fournit des fonctions permettant
  - de scruter le réseau X
  - d'attendre un événement sur le réseau X



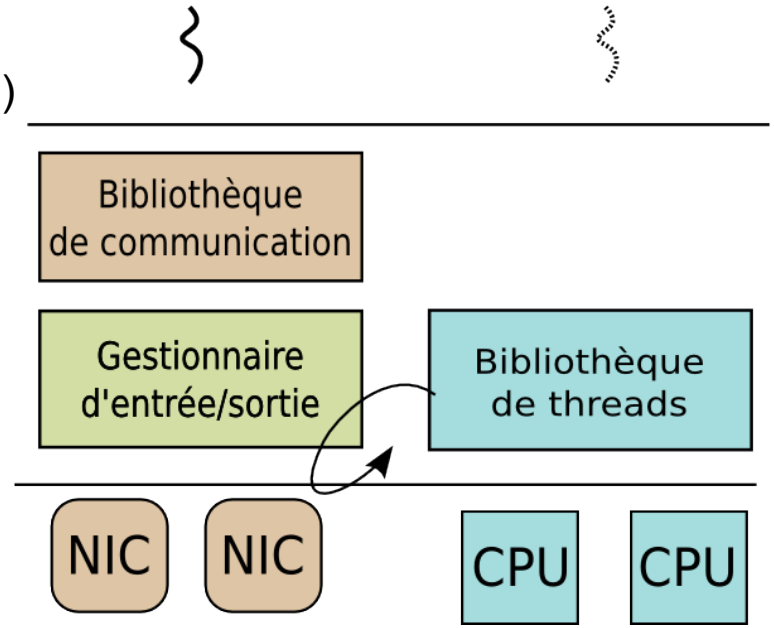
Détection des événements réseau

- La bibliothèque de communication fournit des fonctions permettant
  - de scruter le réseau X
  - d'attendre un événement sur le réseau X
- Délégation de l'attente d'événement
  - Non-bloquant
  - Bloquant
- Utilisation des fonctions pour détecter les événements



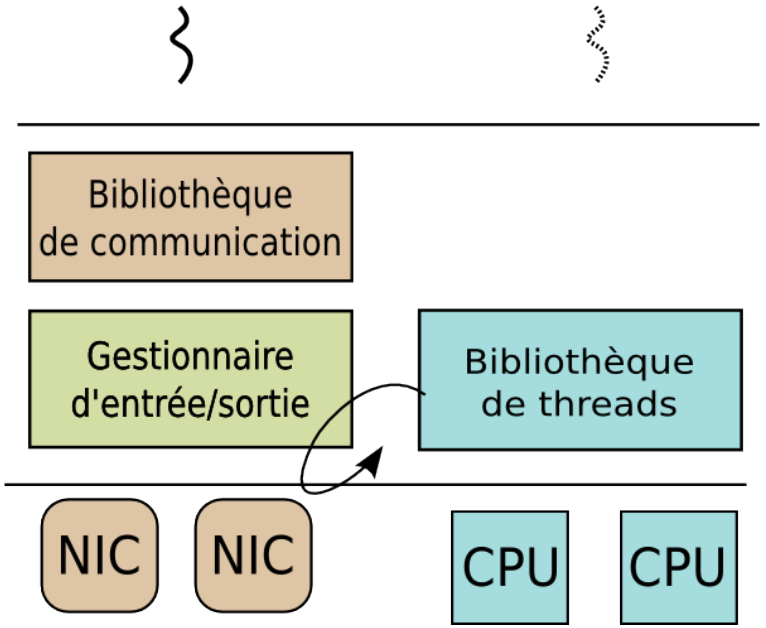
Collaboration avec l'ordonnanceur de threads

- L'ordonnanceur de threads
  - Fournit des informations (charge des processeurs, etc.)
  - Donne la main au gestionnaire d'entrée/sortie à certains points-clés
    - Lors d'un changement de contexte
    - Lors de la réception d'un signal d'horloge
    - Lorsqu'un cœur est inutilisé



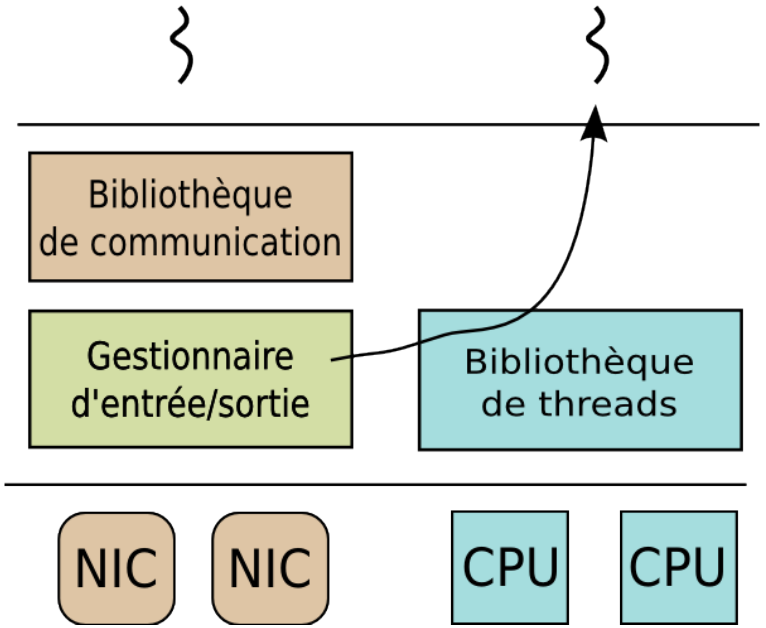
Collaboration avec l'ordonnanceur de threads

- Le gestionnaire d'entrée/sortie
  - Utilise les callbacks en fonction de ces informations
    - Systèmes chargés : appels bloquants
    - Systèmes sous-utilisés : scrutation
  - Informe l'ordonnanceur des threads débloqués



Collaboration avec l'ordonnanceur de threads

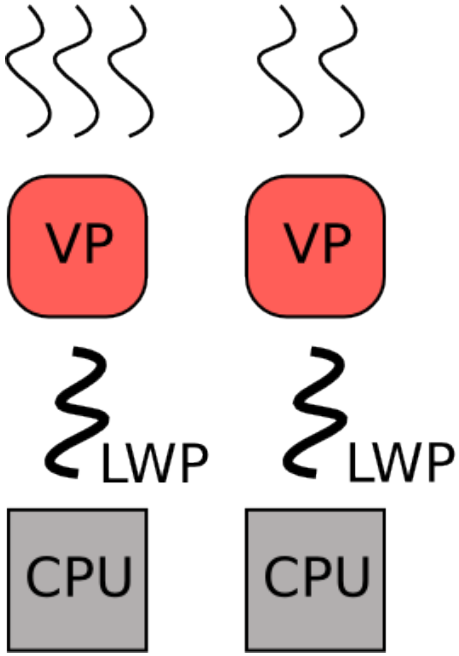
- Le gestionnaire d'entrée/sortie
  - Utilise les callbacks en fonction de ces informations
    - Systèmes chargés : appels bloquants
    - Systèmes sous-utilisés : scrutation
  - Informe l'ordonnanceur des threads débloqués



Gestion des appels bloquants

- Bibliothèque de threads de niveau utilisateur
  - Un thread noyau (LWP, *LightWeight Process*) par cœur
  - Plusieurs threads utilisateur par LWP

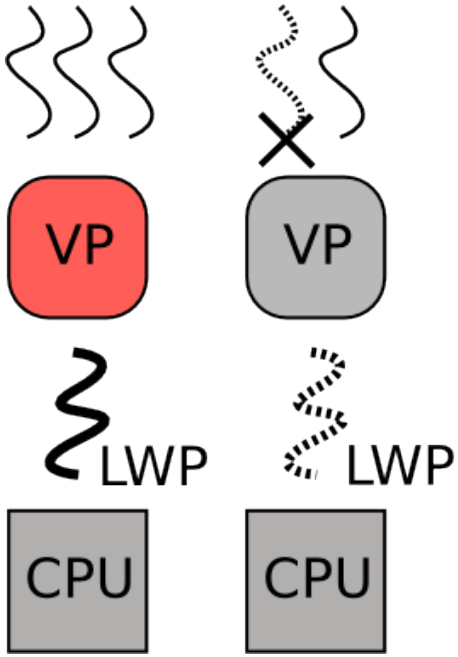
→ Que se passe-t-il lors d'un appel bloquant ?



Gestion des appels bloquants

- Bibliothèque de threads de niveau utilisateur
  - Un thread noyau (LWP, *LightWeight Process*) par cœur
  - Plusieurs threads utilisateur par LWP

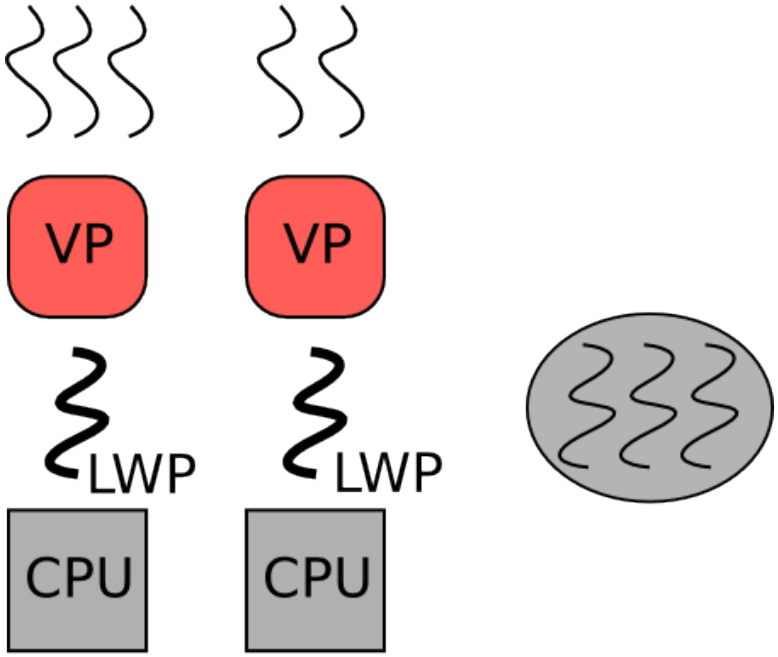
→ Que se passe-t-il lors d'un appel bloquant ?



Gestion des appels bloquants

- Bibliothèque de threads de niveau utilisateur
  - Un thread noyau (LWP, *LightWeight Process*) par cœur
  - Plusieurs threads utilisateur par LWP

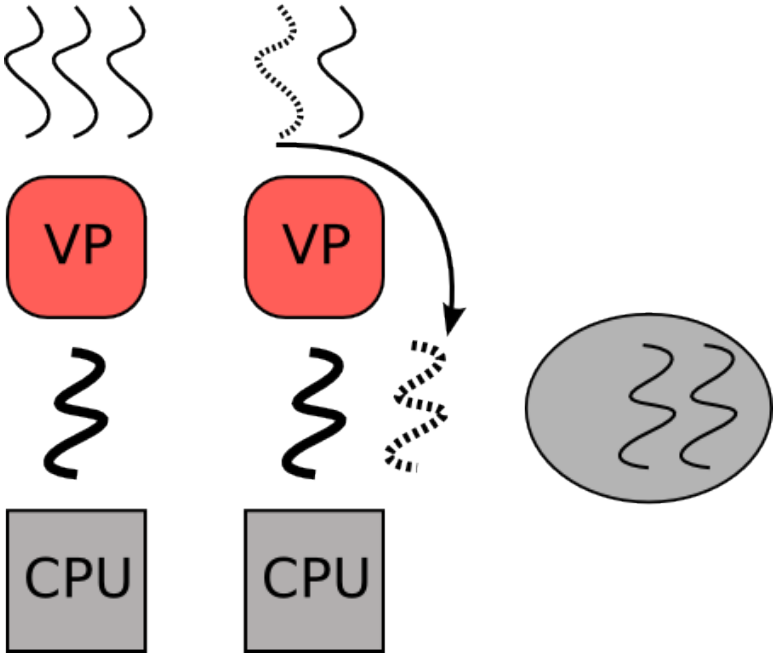
  → Que se passe-t-il lors d'un appel bloquant ?
- Réserve de LWP supplémentaires



Gestion des appels bloquants

- Bibliothèque de threads de niveau utilisateur
  - Un thread noyau (LWP, *LightWeight Process*) par cœur
  - Plusieurs threads utilisateur par LWP

  → Que se passe-t-il lors d'un appel bloquant ?
- Réserve de LWP supplémentaires
- Exportation des appels bloquants

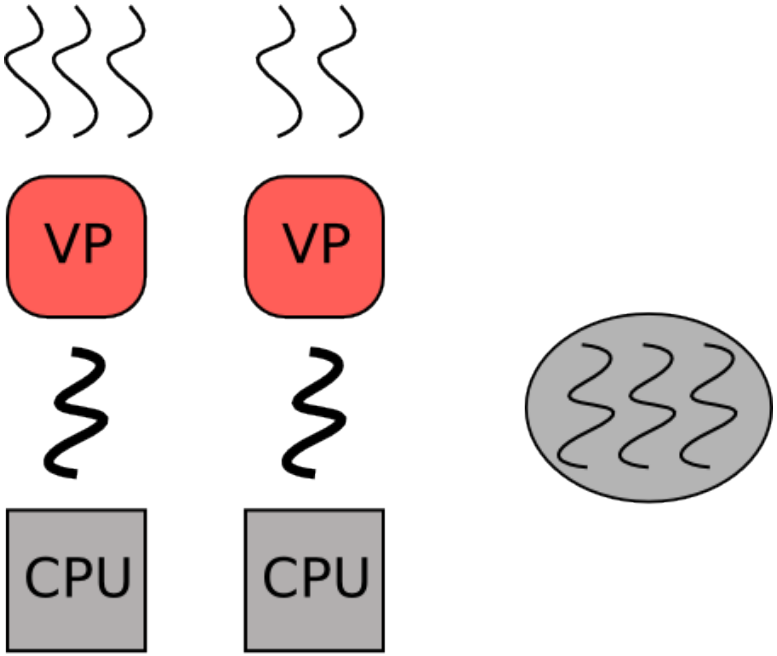


Gestion des appels bloquants

- Bibliothèque de threads de niveau utilisateur
  - Un thread noyau (LWP, *LightWeight Process*) par cœur
  - Plusieurs threads utilisateur par LWP

  → Que se passe-t-il lors d'un appel bloquant ?
- Réserve de LWP supplémentaires
- Exportation des appels bloquants
- Par rapport aux *Scheduler Activations* [Anderson 92]
  - Connaissance des appels bloquants *a priori*
  - Nombre borné d'appels bloquant simultanés

    → Temps de réaction constant



Collaboration avec l'ordonnanceur de threads

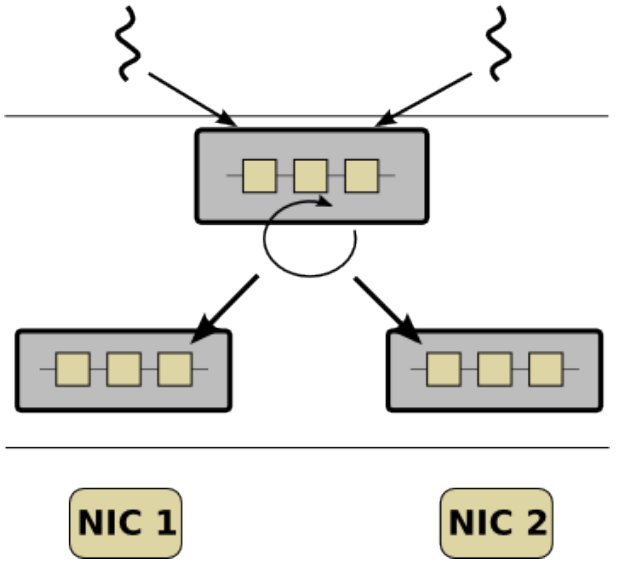
- Avantages :
  - Détection des événements réseau en temps constant
    - Réponse à une demande de rendez-vous
  - Ne perturbe pas le calcul
  - Profite des cœurs inactifs
    - Progression des communications en arrière-plan

Proposition

- Progression des communications
- Support des accès concurrents
- Parallélisation des traitements

Gestion des attentes concurrentes

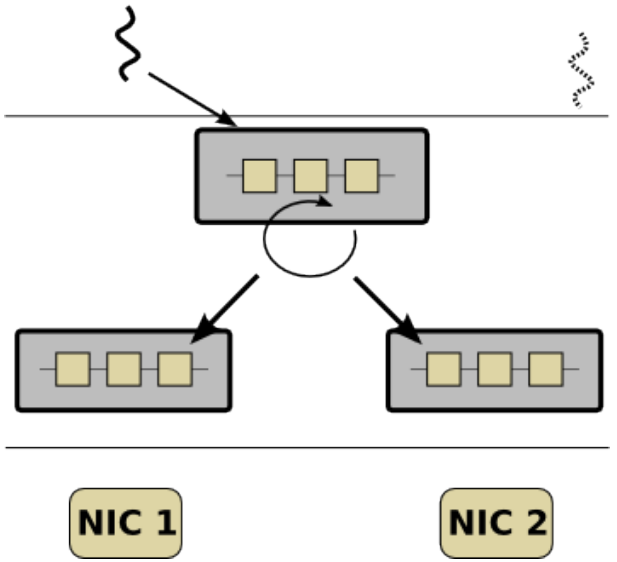
- Verrouillage de la bibliothèque
  - Gros grain ou grain fin
  - Assure l'intégrité des structures de données
  - Arbitrage des accès réseaux
- Attentes concurrentes
  - Plusieurs threads appellent MPI_Wait
  - Implémentation de MPI_Wait : attente active
    - Contention lors de l'accès aux cartes réseau
    - Gaspillage des ressources



Gestion des attentes concurrentes

- Attente passive :
  - Attente sur une primitive bloquante
  - Possibilité d'ordonnancer des threads
  - Surcoût important (changement de contexte)

- Mécanisme d'attente mixte
  - Attente active puis attente passive
  - Faible surcoût
  - Attentes concurrentes efficaces
  - Possibilité d'ordonnancer des threads



Proposition

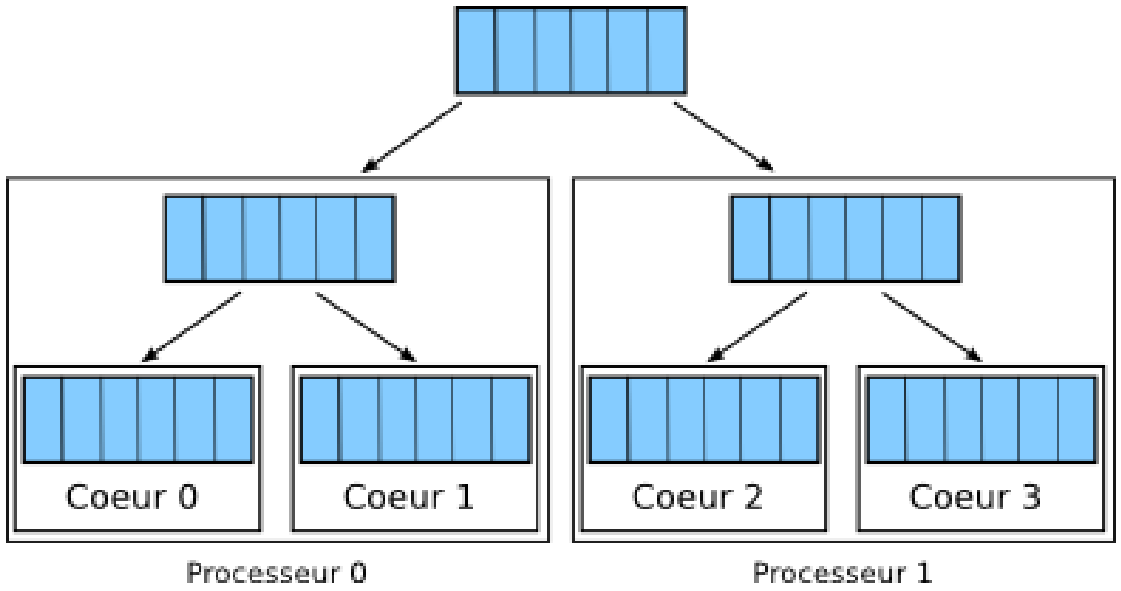
- Progression des communications
- Support des accès concurrents
- Parallélisation des traitements

Parallélisation des traitements

- Collaboration avec l'ordonnanceur permet
  - D'exploiter les cœurs inutilisés
  - De répartir les traitements sur l'ensemble des cœurs
- Exemples d'utilisation :
  - Progression des rendez-vous en arrière-plan
  - Traitement coûteux des communications (copie mémoire, enregistrement, …)
- Problèmes :
  - Machines fortement hiérarchiques
    - Placement des données important
  - Grand nombre de cœurs
    - Passage à l'échelle

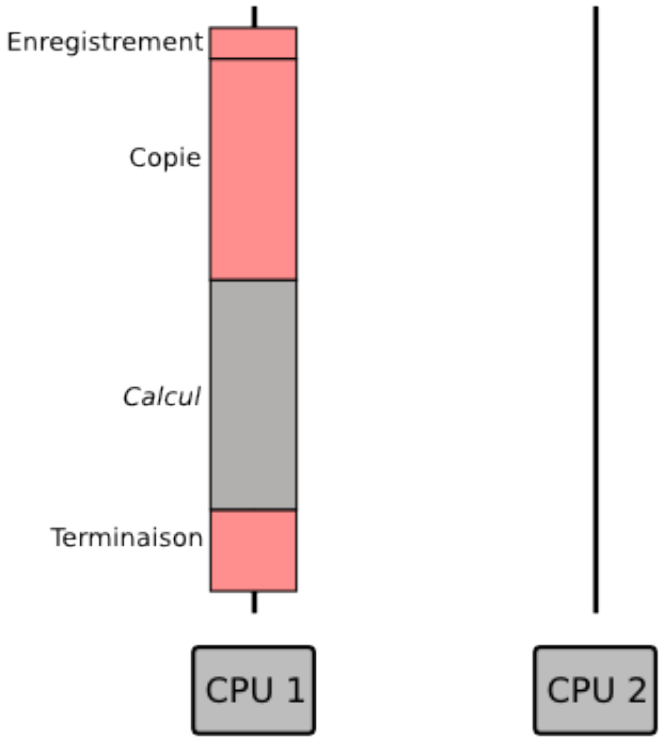
Mécanisme d'exportation de tâches

- Hiérarchie de listes de tâches
  - Appliquée à la topologie
  - Réduit la contention sur les verrous
  - Conserve la localité des données
  - Similaire aux mécanismes implantés dans Marcel



Décomposition des traitements

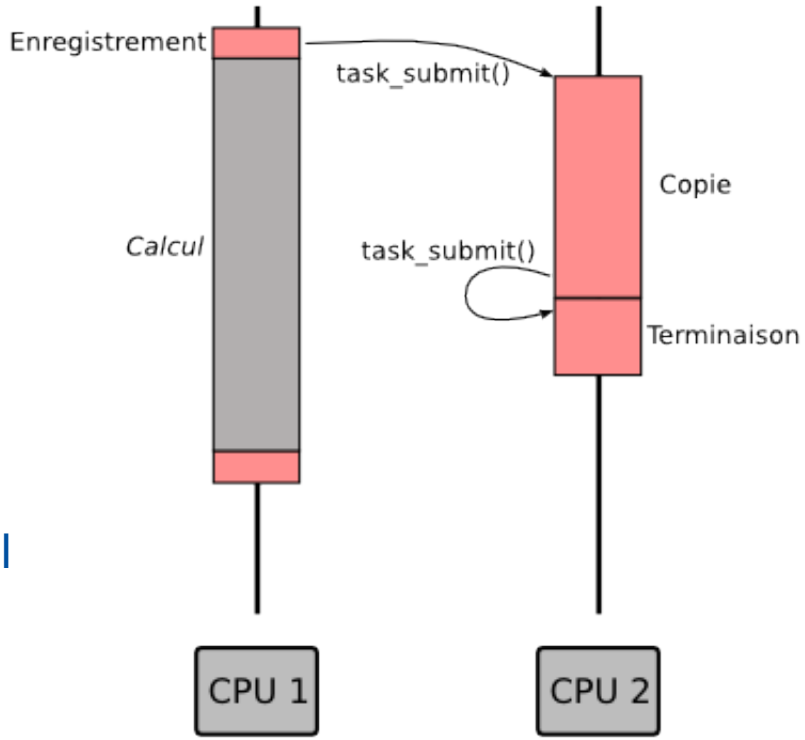
- Déroulement d'une communication :
  - Enregistrement de la requête
    - Destinataire, adresse du message, ...
  - Transfert des données vers la carte
  - Détection de la terminaison



Décomposition des traitements

- Déroulement d'une communication :
  - Enregistrement de la requête
    - Destinataire, adresse du message, ...
  - Transfert des données vers la carte
  - Détection de la terminaison
- Décomposition des traitements en tâches
  - Transfert des données
  - Terminaison

→ Recouvrement des communications et du calcul

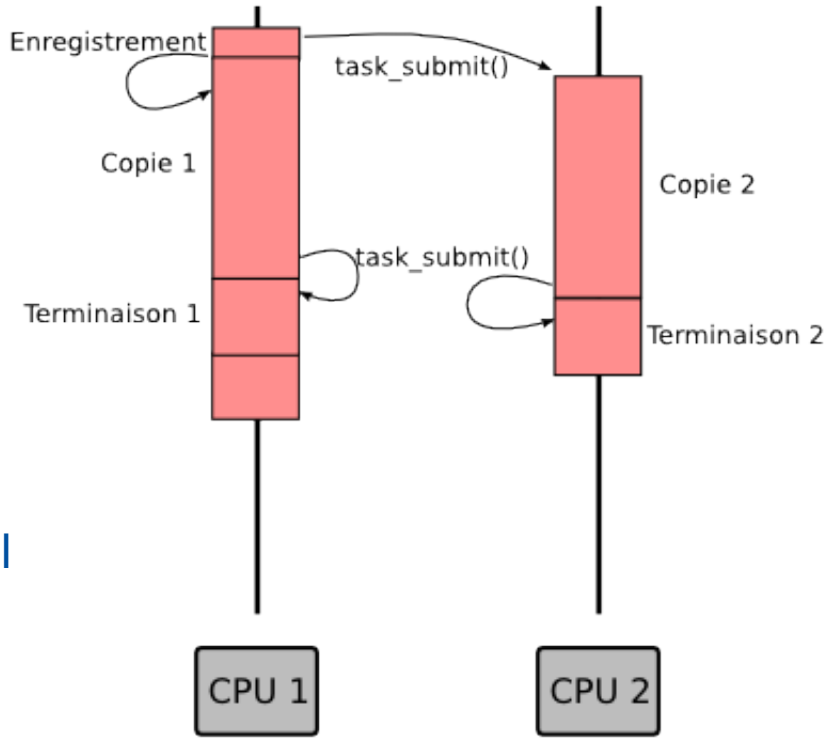


Décomposition des traitements

- Déroulement d'une communication :
  - Enregistrement de la requête
    - Destinataire, adresse du message, ...
  - Transfert des données vers la carte
  - Détection de la terminaison
- Décomposition des traitements en tâches
  - Transfert des données
  - Terminaison

→ Recouvrement des communications et du calcul

→ Envois simultanés sur deux réseaux



Utilisation de PIOMan dans MPICH2/NewMadeleine

- MPICH2/NewMadeleine
  - Gestion des communications réseau par NewMadeleine
  - Gestion des communications en mémoire partagée par Nemesis

- PIOMan comme moteur de progression dans Nemesis
  - Remplacement de l'attente active par une attente mixte

    → Progression des communications en arrière-plan

    → Attentes concurrentes efficaces
- Mécanisme léger de détection des événements
  - Système de boîte aux lettres
  - Pas de verrouillage
    - Surcoût faible

Évaluation

Plate-forme d'expérimentation

- Grappe Joe
  - 2 nœuds
  - Quadri-cœur Xeon X5460 (3.16 GHz)
  - 4 Go de mémoire
  - Carte Myri-10G (pilote MX 1.2.7)
  - Carte InfiniBand ConnectX MT25418 (pilote OFED 1.2)

- Grappe Borderline
  - Grid5000
  - 10 nœuds
  - 4 Processeurs bi-cœurs Opteron 8218 (2.6 GHz)
  - 32 Go de mémoire
  - Carte Myri-10G (pilote MX 1.2.7)
  - Carte InfiniBand ConnectX MT25418 (pilote OFED 1.2)

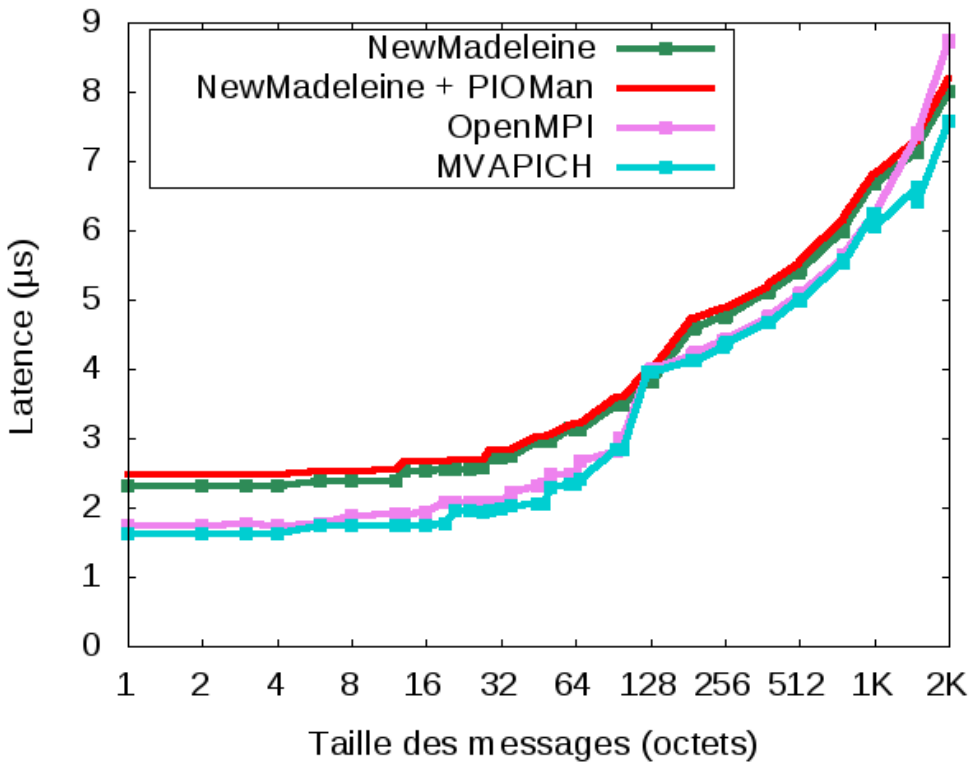
- **MPICH2** [MX, SHMem] [Argonne National Lab]
  - Implémentation générique
  - Utilisation du *channel* Nemesis/MX
  - Version 1.1.1

- **OpenMPI** [MX, IB, SHMem]
  - Implémentation générique
  - Utilisation du *BTL* MX et du *BTL* OpenIB
  - Version 1.3.3

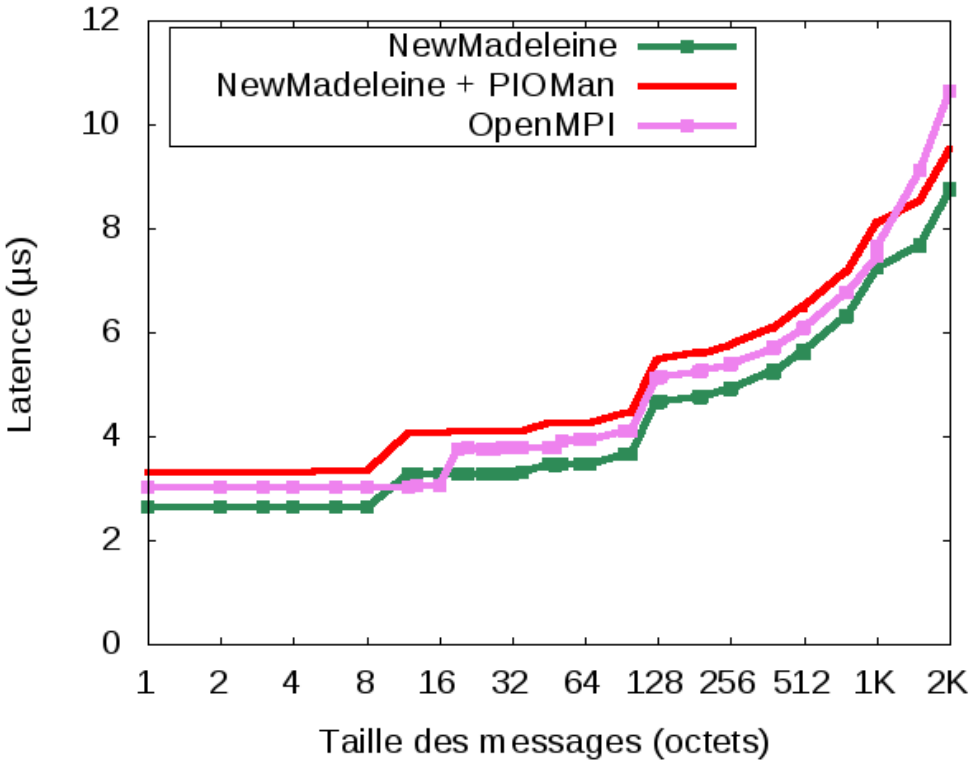
- **MVAPICH2** [IB, SHMem] [Ohio State Univ.]
  - Implémentation de MPI pour InfiniBand
  - Version 1.4 rc1

Micro-Benchmarks
PIOMan

- InfiniBand
  - NewMadeleine : 2,05 µs
  - PIOMan : 2,47 µs (+ 400 ns)

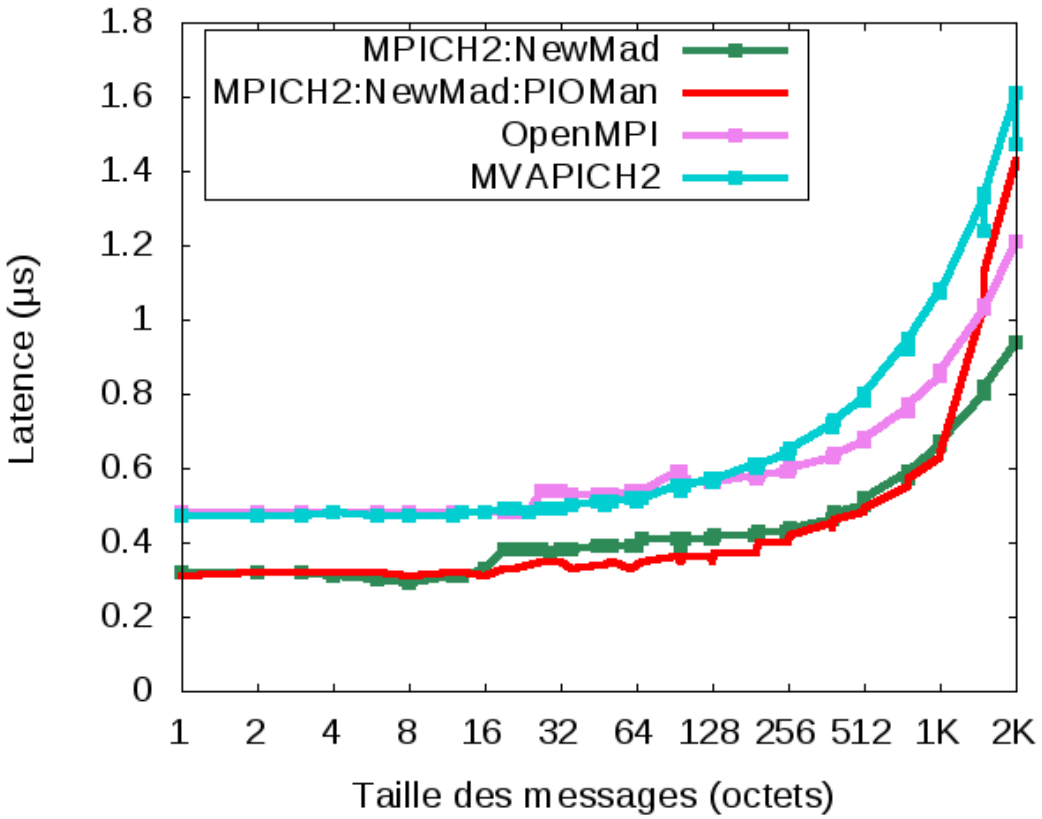


- Myrinet
  - NewMadeleine : 2,62 µs
  - PIOMan : 3,31 µs (+ 700 ns)



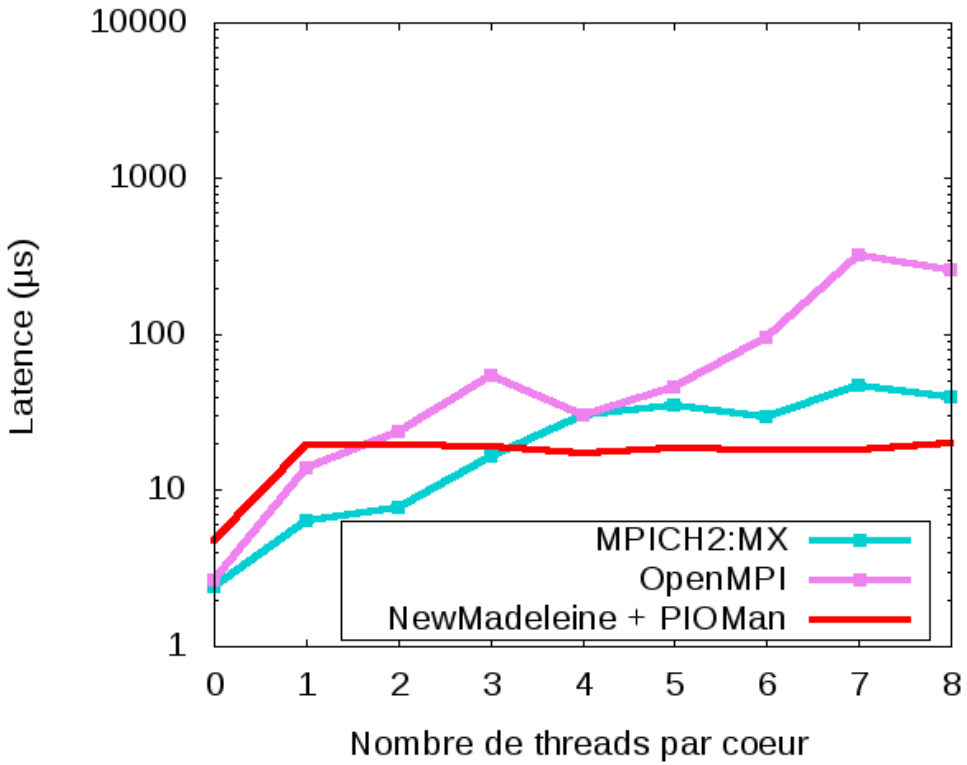
Micro-Benchmarks
Mémoire partagée

- Impact de PIOMan : < 50 ns



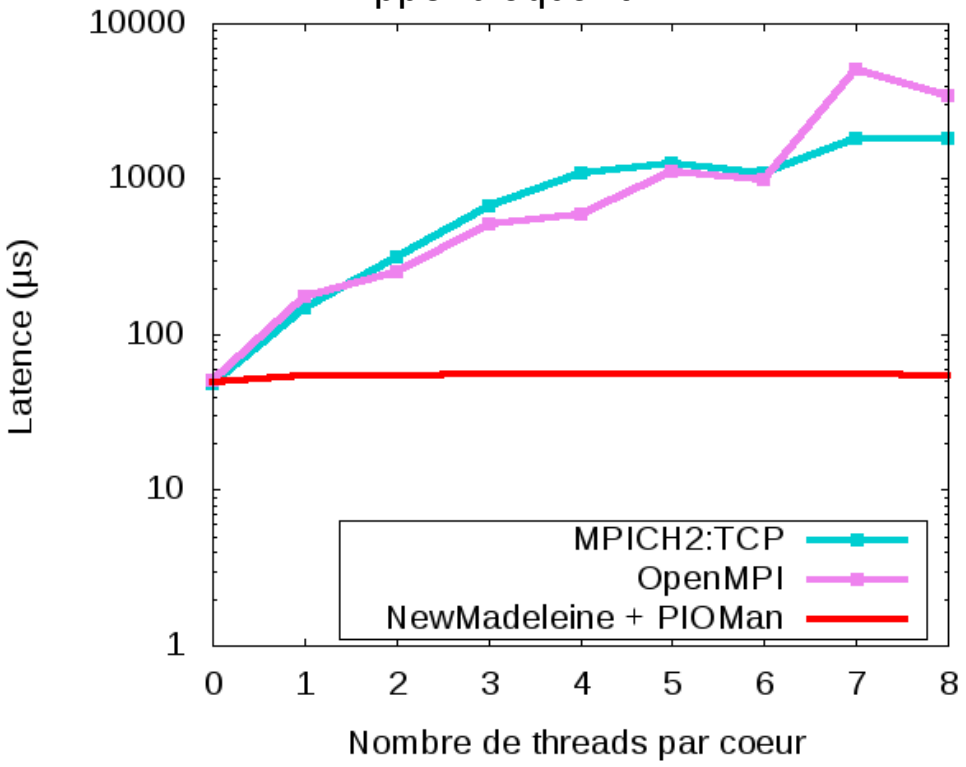
Micro-Benchmarks
Réactivité

- Test de latence sur des machines chargées
  - Ping pong classique
  - N threads de calcul par cœur



Myrinet

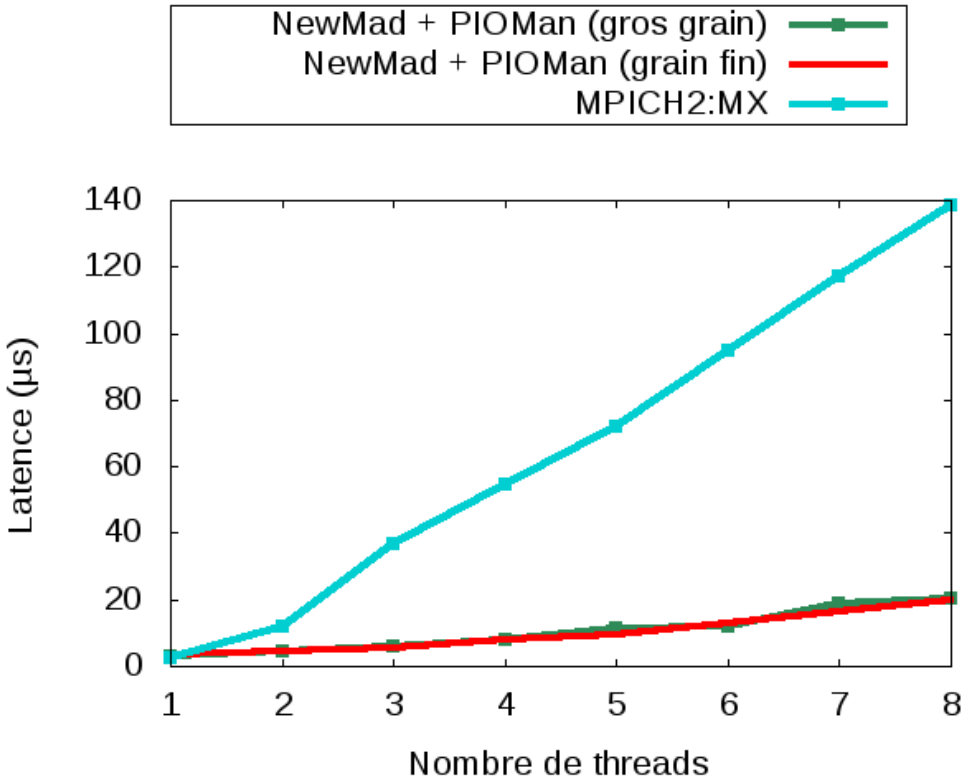
- 0 thread : machine sous-utilisée
  - Scrutation
- > 1 thread : machine surchargée
  - Appel bloquant



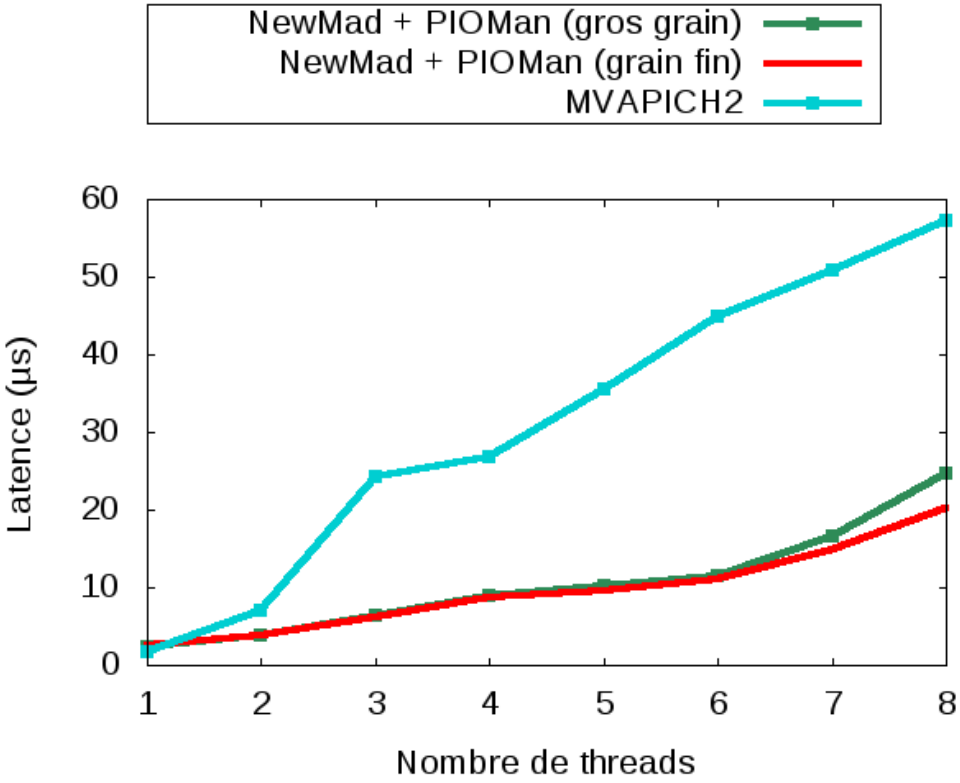
TCP/Ethernet

Micro-Benchmarks
Accès concurrents

- Ping-pong en parallèle
  - Mesure du temps de transfert moyen pour 8 octets



Myrinet

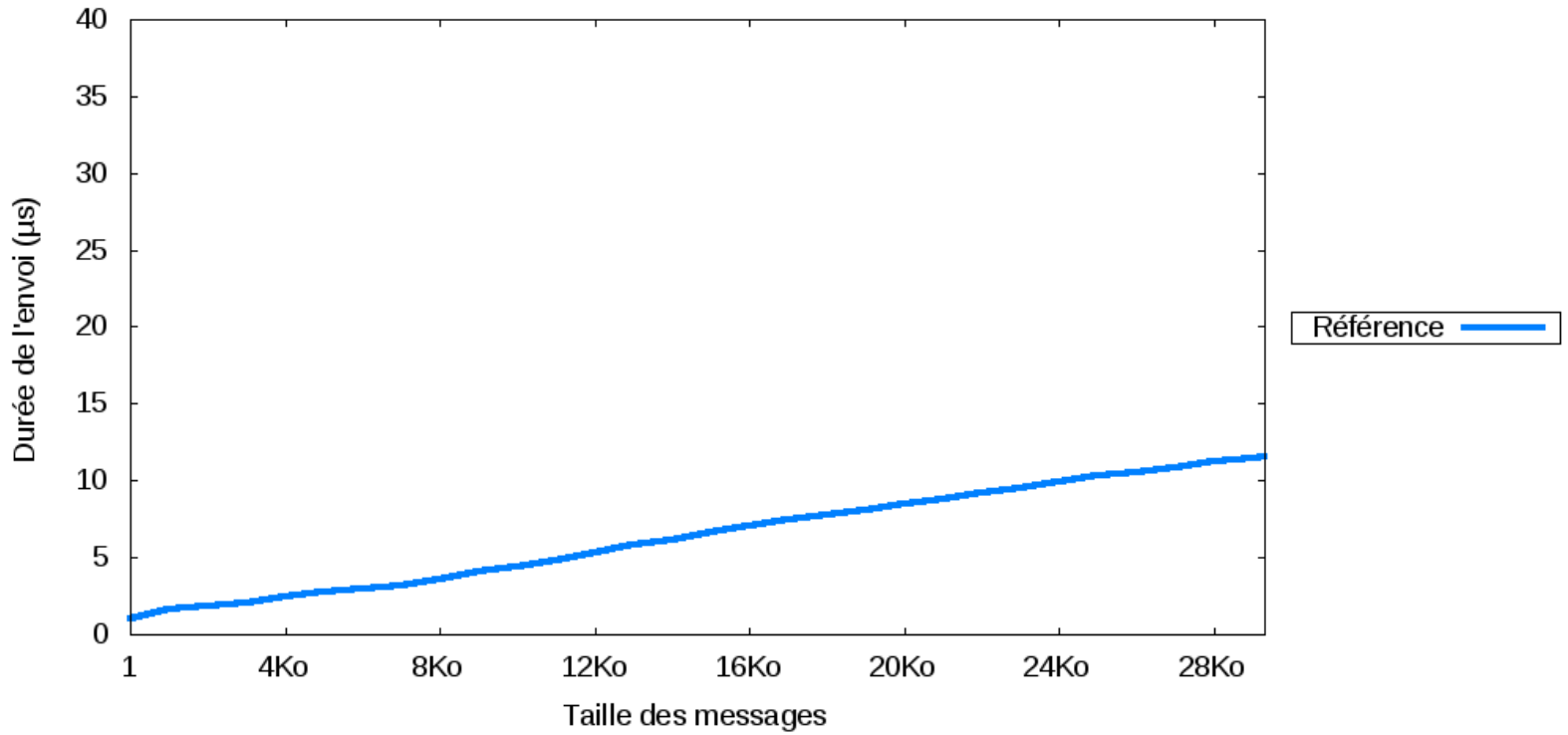


InfiniBand

Micro-Benchmarks

Parallélisation de l'envoi de message

```
MPI_Isend()
MPI_Wait()
```

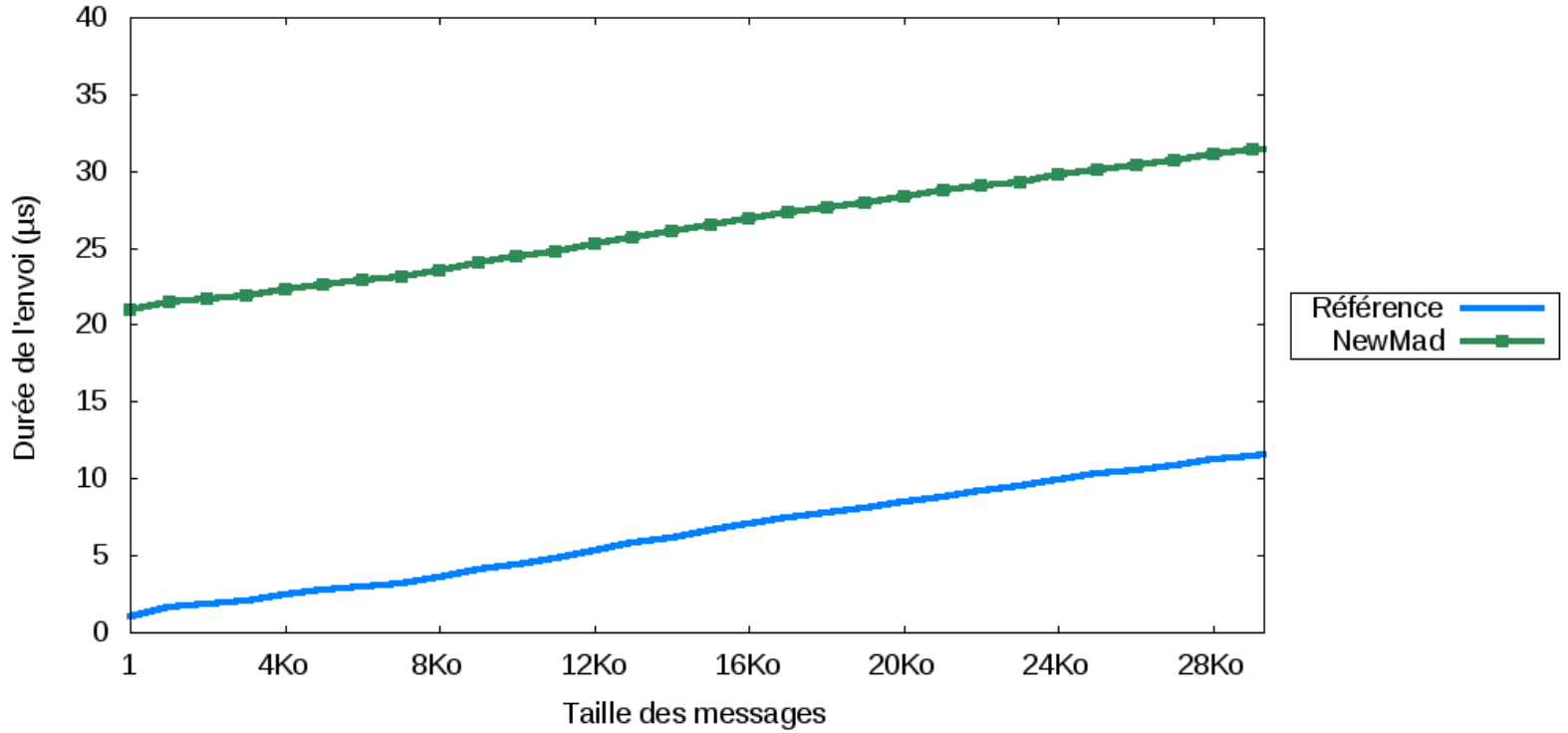


Myrinet

Micro-Benchmarks

Parallélisation de l'envoi de message

```
MPI_Isend()
Calcul(20 µs)
MPI_Wait()
```

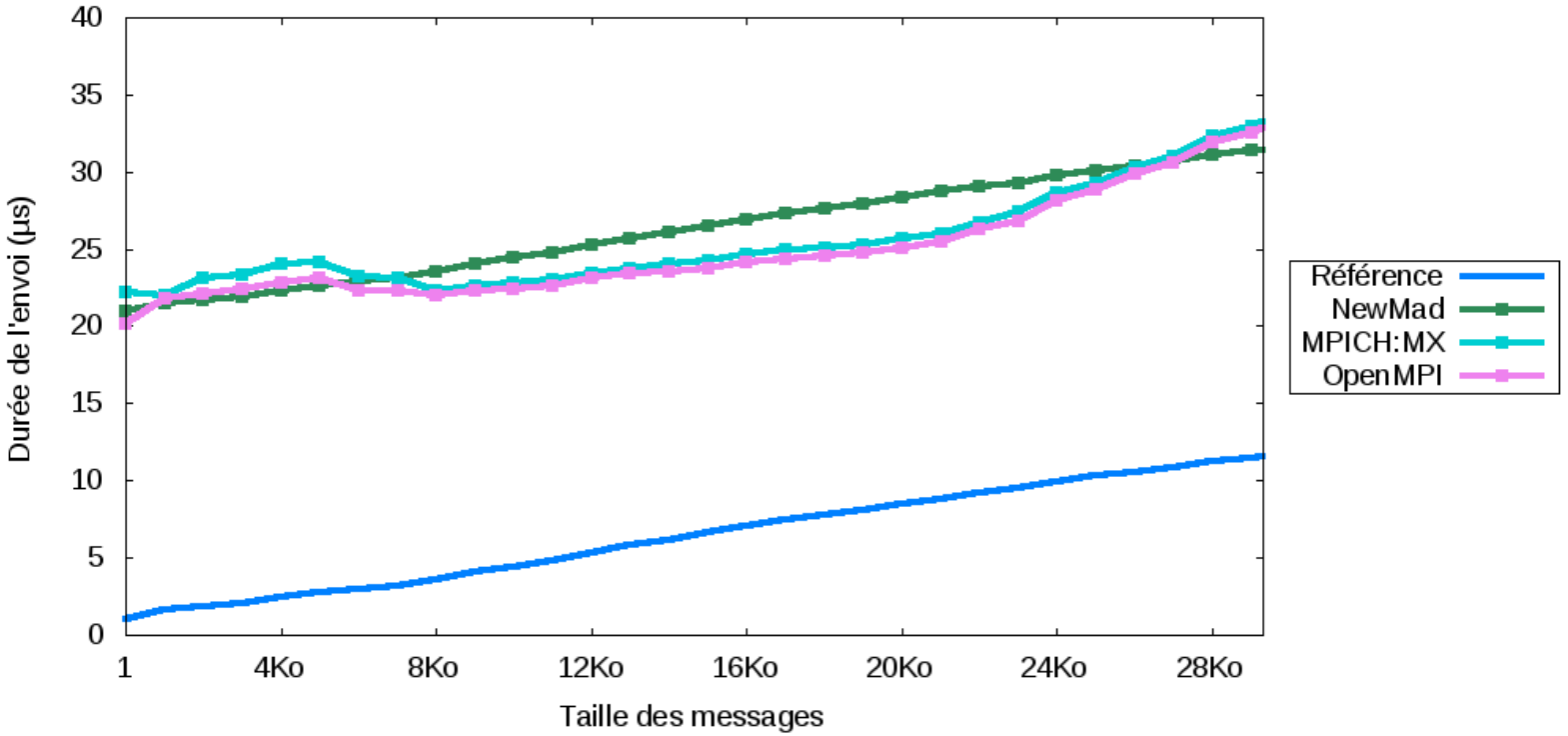


Myrinet

Micro-Benchmarks

Parallélisation de l'envoi de message

```
MPI_Isend()
Calcul(20 µs)
MPI_Wait()
```

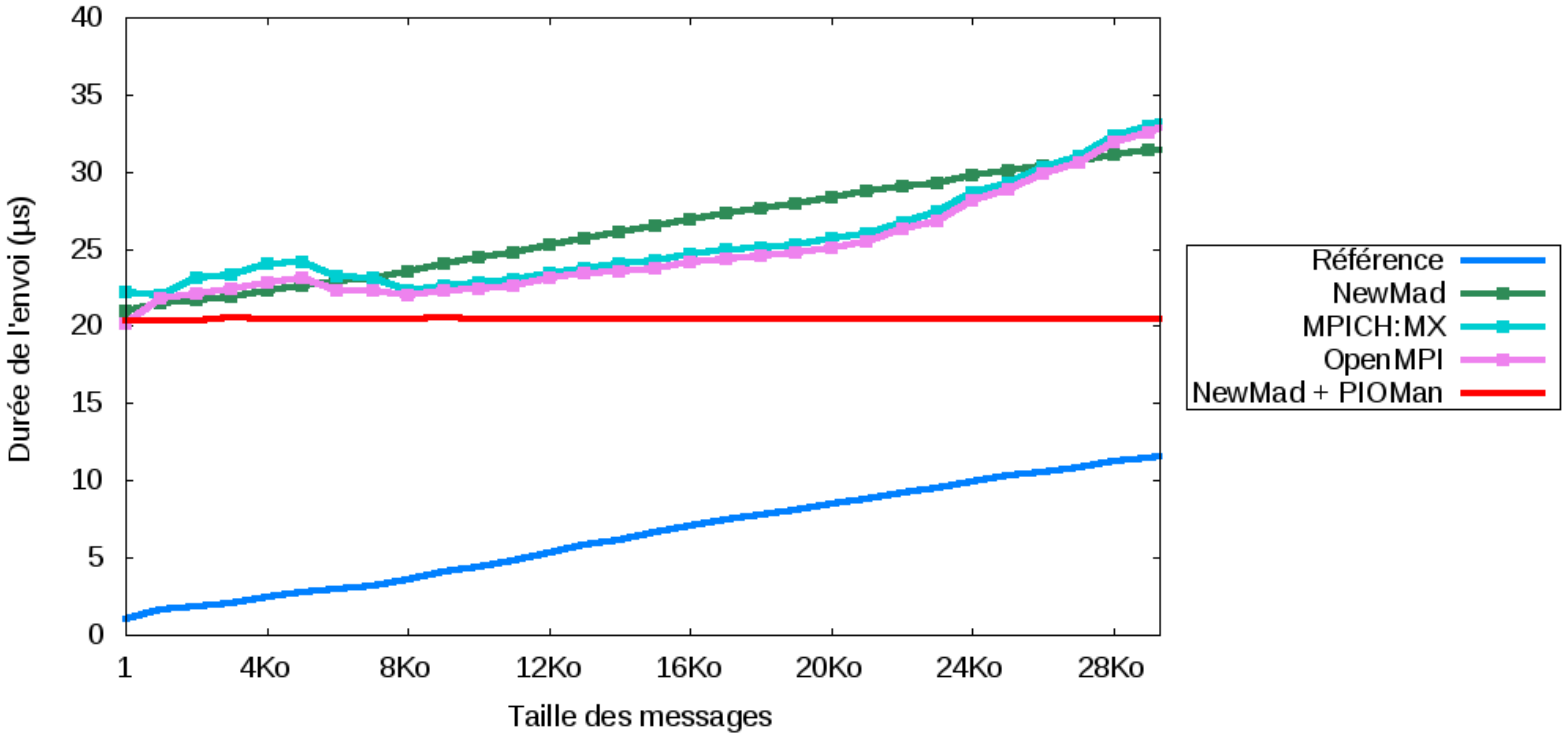


Myrinet

Micro-Benchmarks

Parallélisation de l'envoi de message

```
MPI_Isend()
Calcul(20 µs)
MPI_Wait()
```

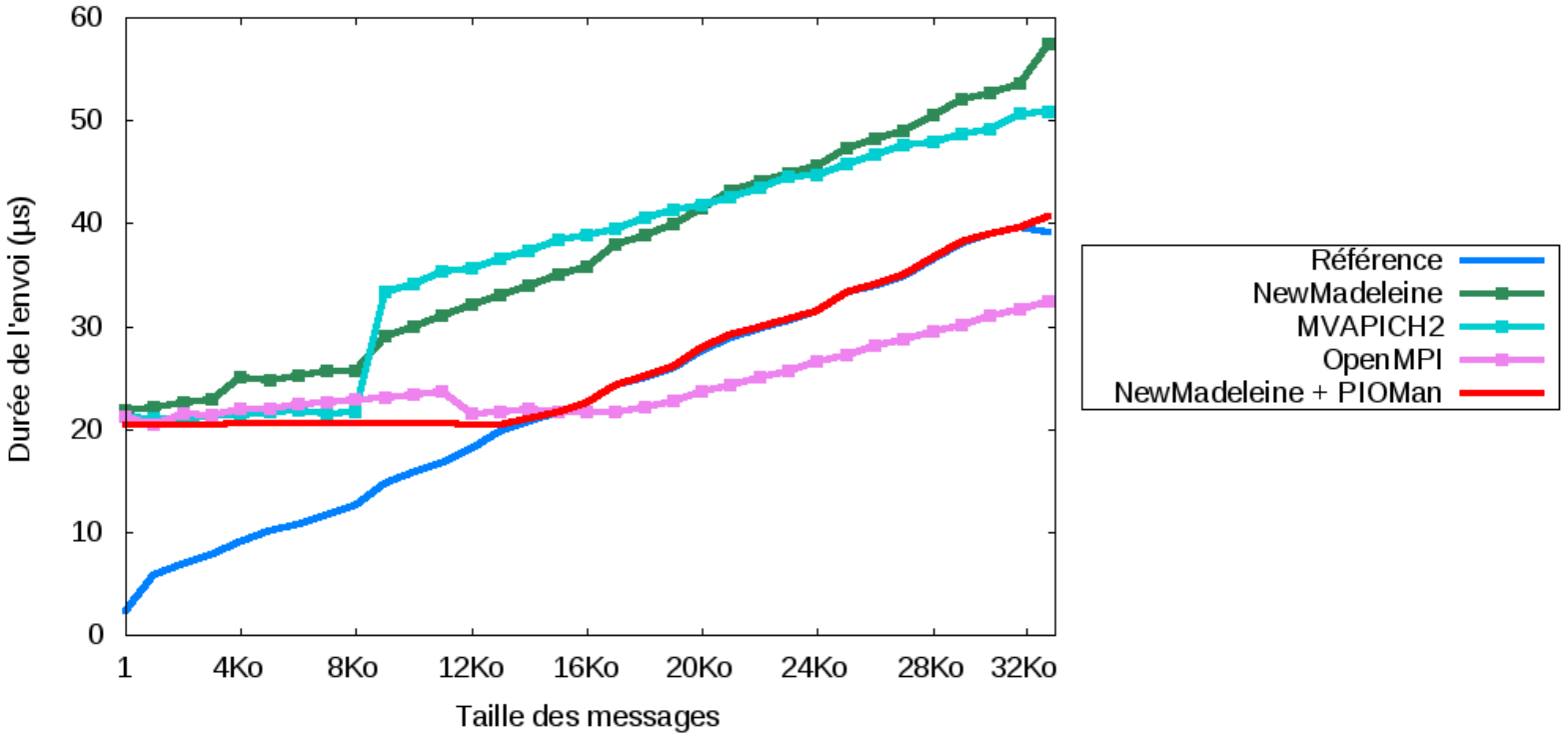


Myrinet

Micro-Benchmarks

Parallélisation de l'envoi de message

```
MPI_Isend()
Calcul(20 µs)
MPI_Wait()
```

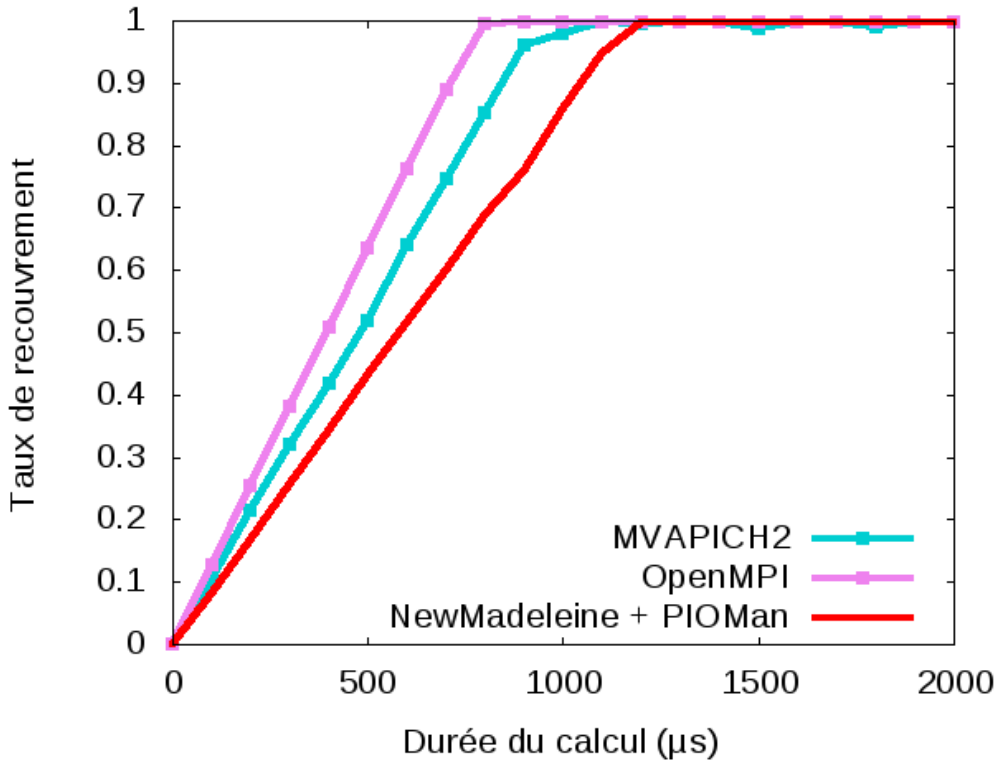


InfiniBand

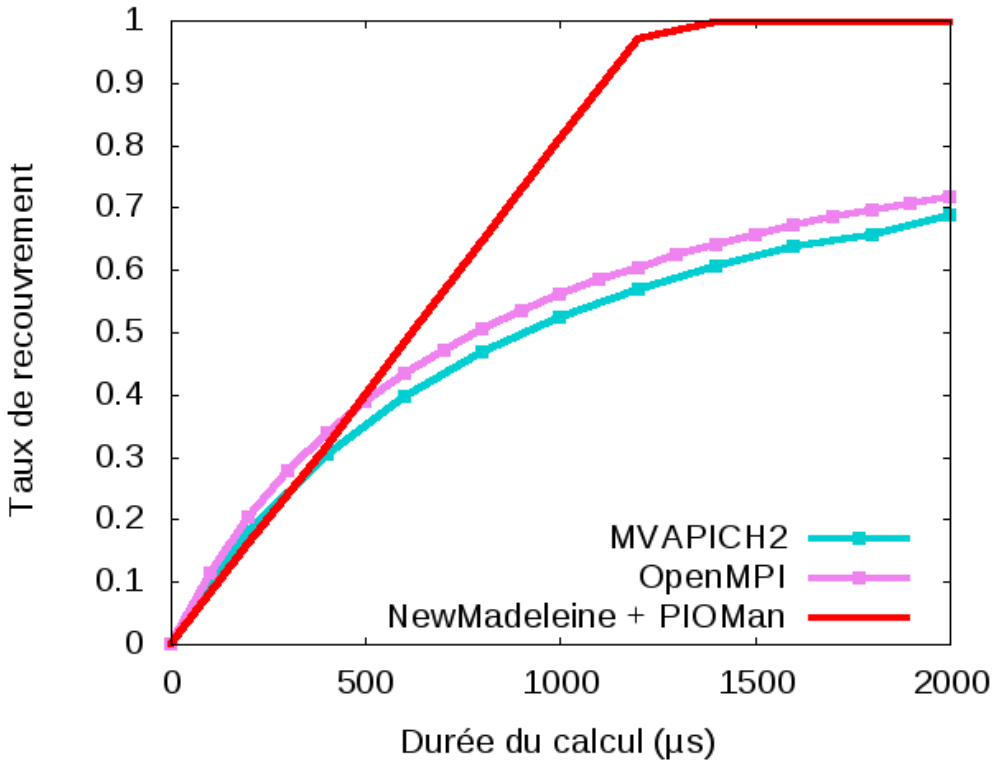
Micro-Benchmarks

Recouvrement des communication par du calcul

- Progression des rendez-vous
  - Sur InfiniBand
  - Messages de grandes tailles (1 Mo)



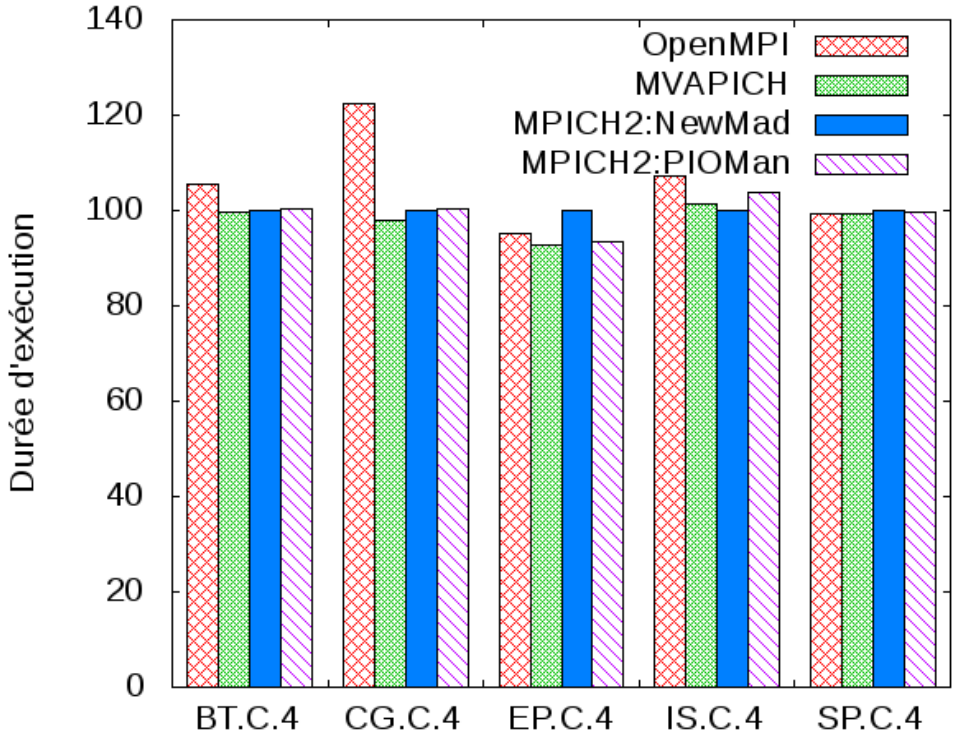
Émission

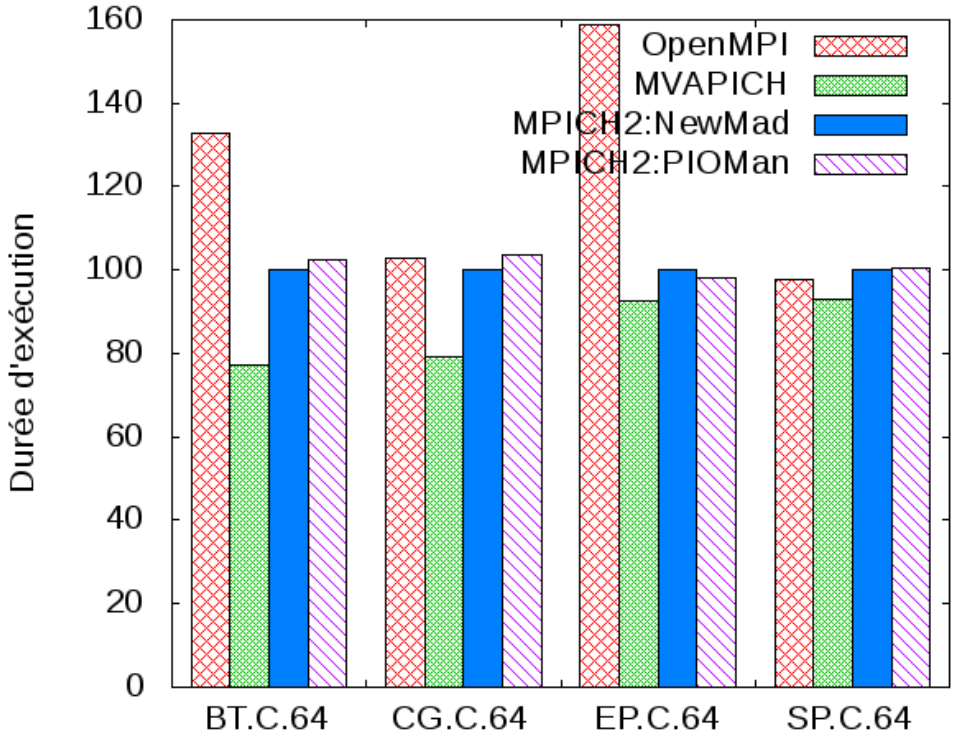


Réception

NAS Parallel Benchmarks

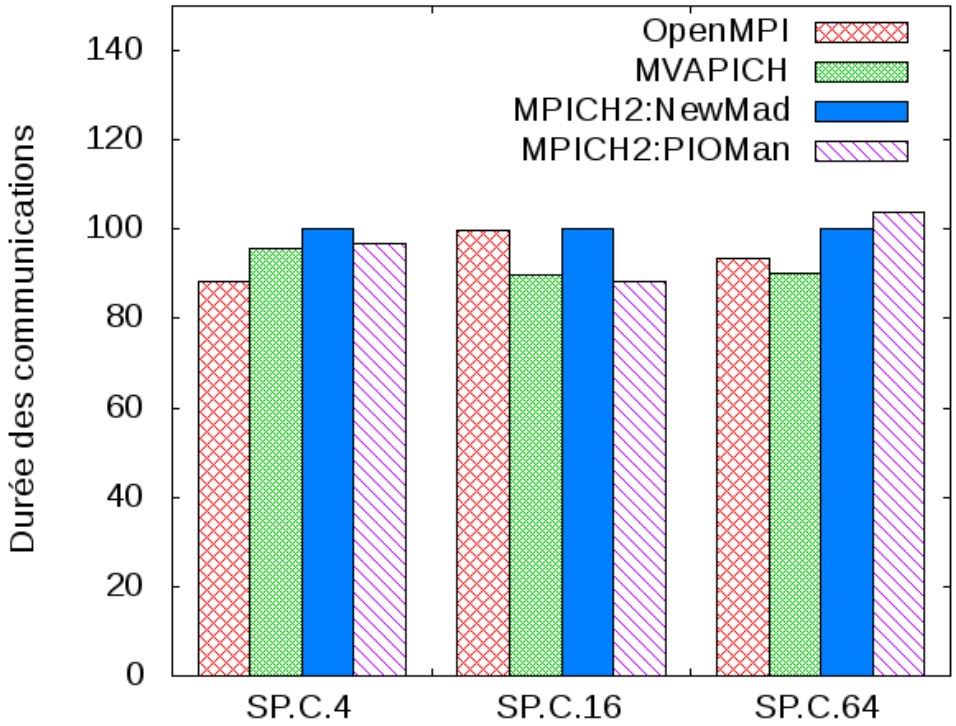
- Ensemble de benchmarks représentatif des applications réelles
  - Pas de thread
  - Uniquement des communications bloquantes (sauf SP)
- Plate-forme : borderline (8 nœuds avec 8 cœurs chacun)





NAS Parallel Benchmarks
SP

- Durée des communications faible par rapport au temps de calcul
  - Recouvrement des communications par du calcul
  - SP.C.4 et SP.C.16 : cœurs disponibles, progression des communications
  - SP.C.64 : aucun cœur disponible. Pas de progression des communications



Conclusion

- Évolution des architectures et des modèles de programmation
  - Implémentations MPI inadaptées
- PIOMan : collaboration entre les communications et l'ordonnancement
  - Progression des communications
    - Détection adaptative des événements réseau
  - Accès concurrents
    - Verrouillage dans NewMadeleine
    - Attente mixte
  - Traitement parallèle des communications
    - Exploitation des unités de calcul libres

Perspectives

- Parallélisation avancée de NewMadeleine
  - Pré-calcul des optimisations
  - Enregistrement mémoire en arrière-plan
  - Compression à la volée
- Auto-adaptation
  - Sélection automatique des mécanismes (verrouillage, attente mixte, …)
    - Analyse de l'historique
    - Informations données par le développeur / compilateur
  - Placement des tâches de communication [S. Moreaud]

Perspectives

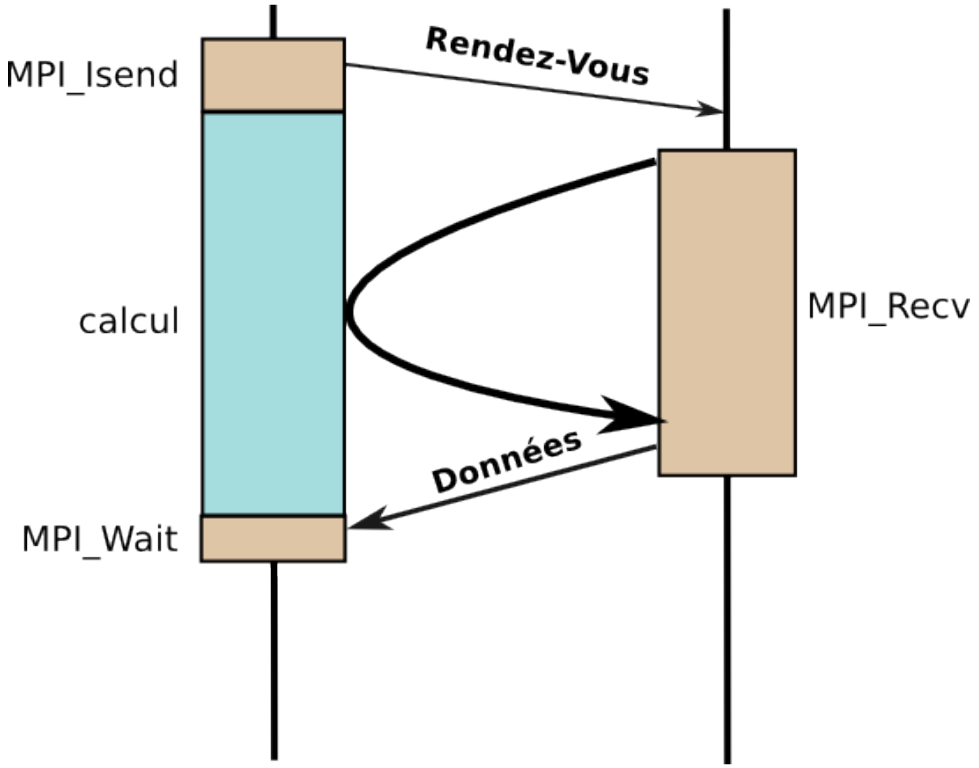
- Extension à plusieurs processus [J. Clet-Ortega]
  - Modèles hybrides MPI+threads sur machines NUMA
  - Décisions pour l'ensemble des ressources du nœud
- Gestion de tous les événements du système
  - Accès aux disques / systèmes de fichiers distants
    - Augmentation de la masse de données à gérer
  - MPI + threads + CUDA [C. Augonnet]
- Influence sur les implémentations de MPI

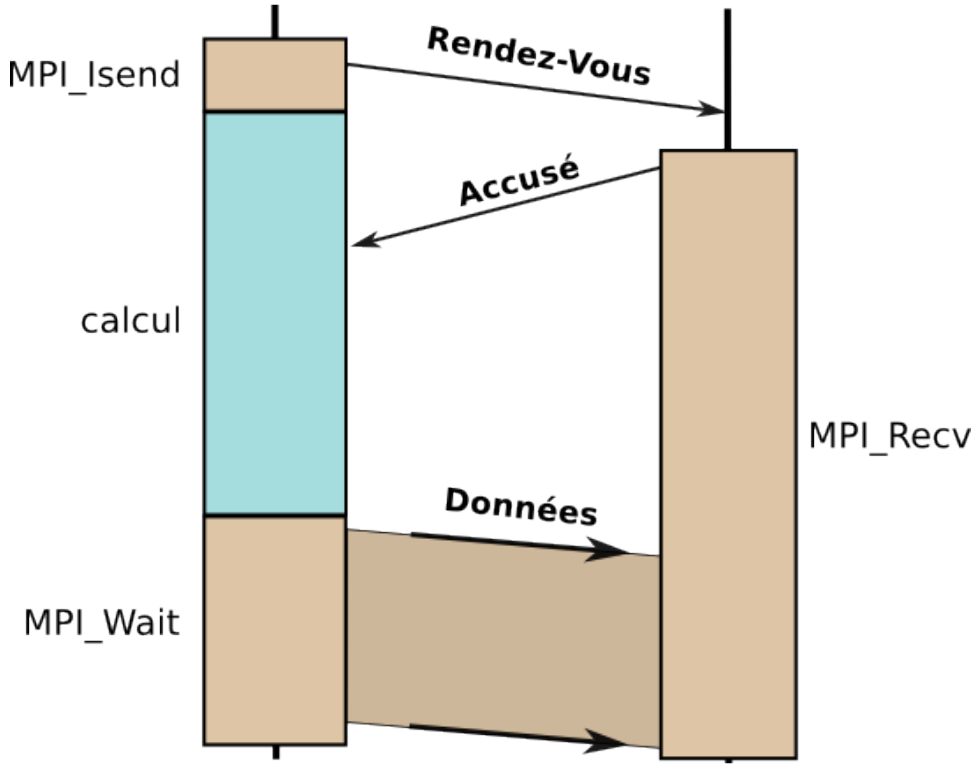
Merci pour votre attention

http://runtime.bordeaux.inria.fr/Runtime/

http://gforge.inria.fr/projects/pm2/

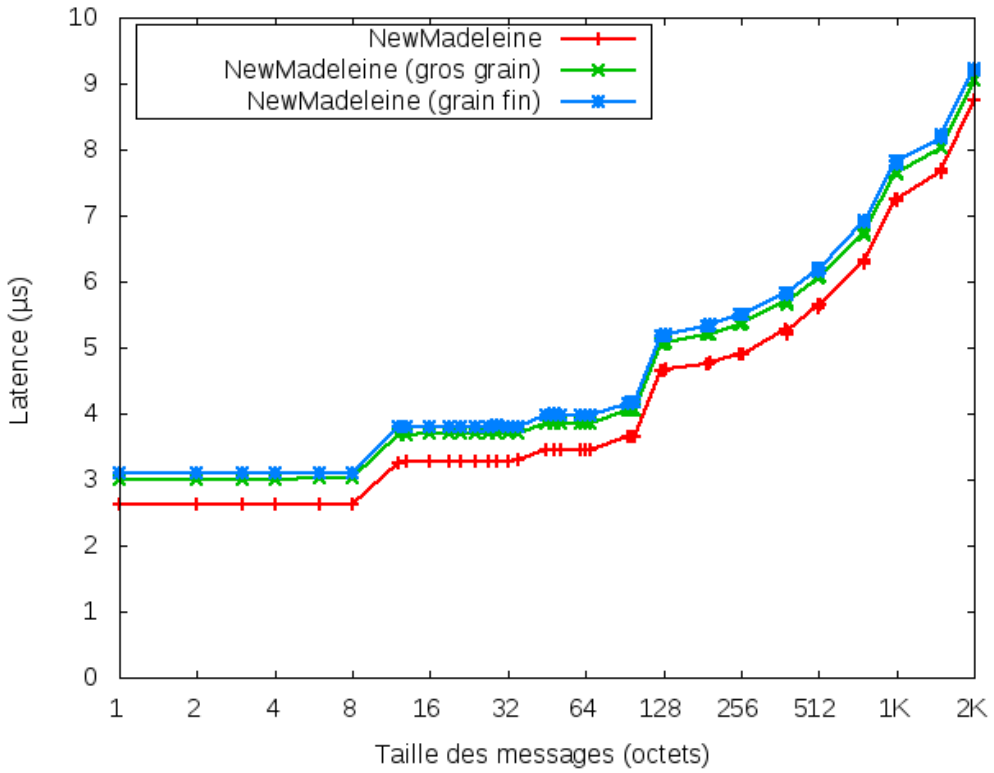
Protocoles de rendez-vous sur InfiniBand



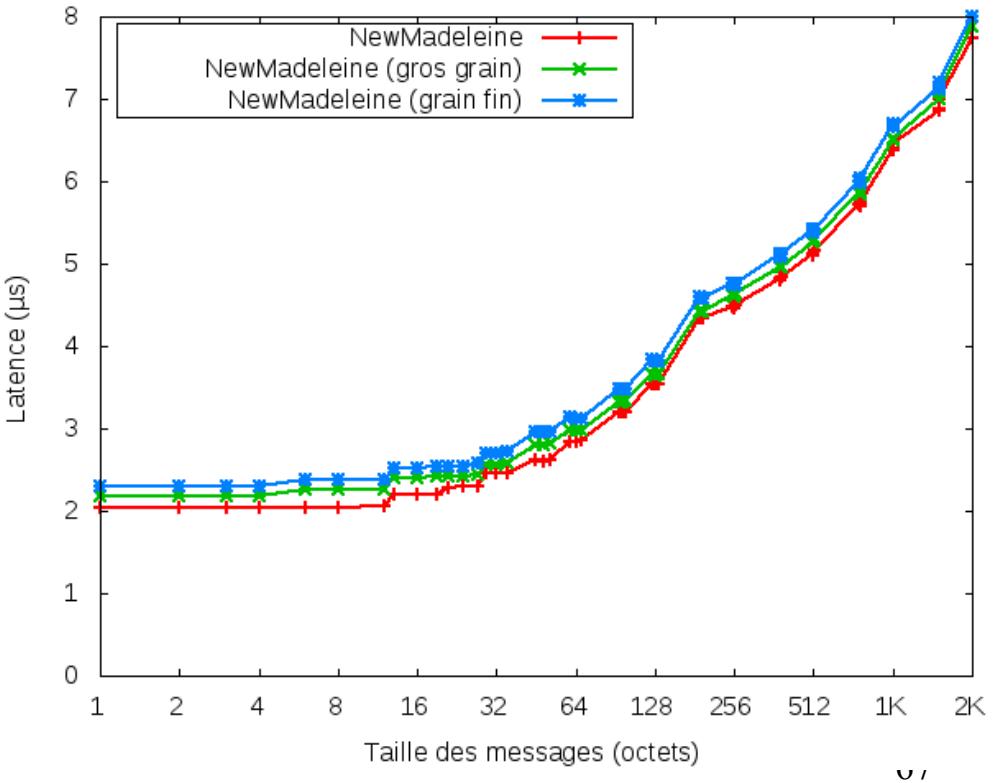


Micro-Benchmarks
Mécanismes de protection

- Myrinet
  - Sans verrou : 2,62 µs
  - Gros grain : 3 µs (+ 400 ns)
  - Grain fin : 3,10 µs (+ 500 ns)

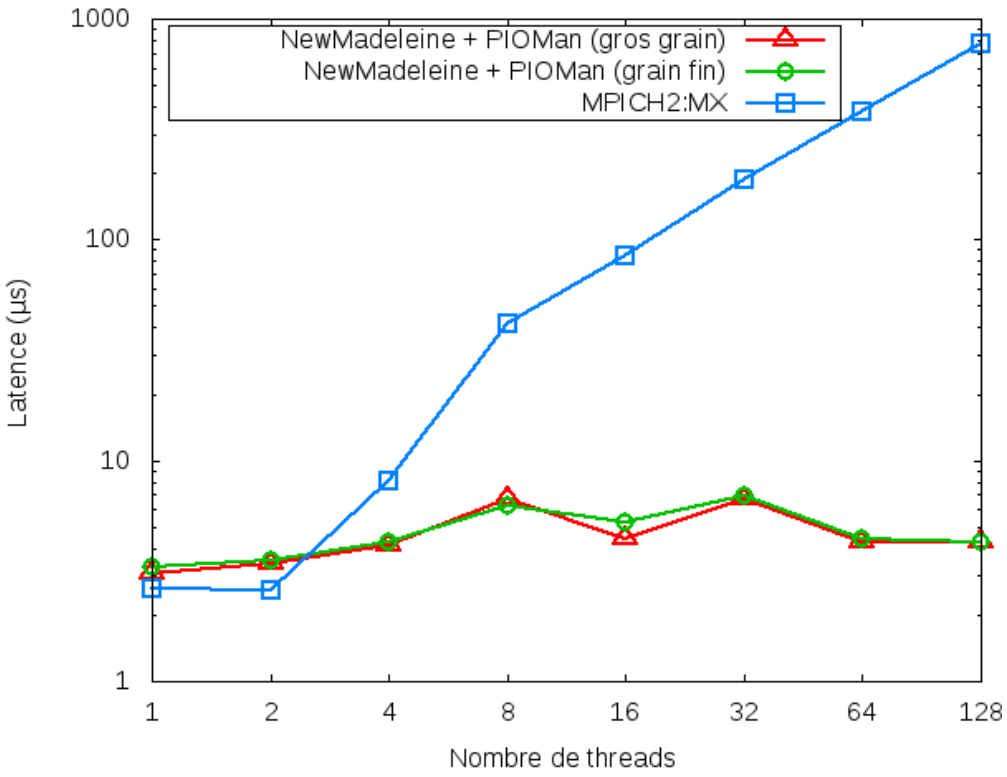


- InfiniBand
  - Sans verrou : 2,05 µs
  - Gros grain : 2,19 µs (+ 150 ns)
  - Grain fin : 2,30 µs (+ 250 ns)

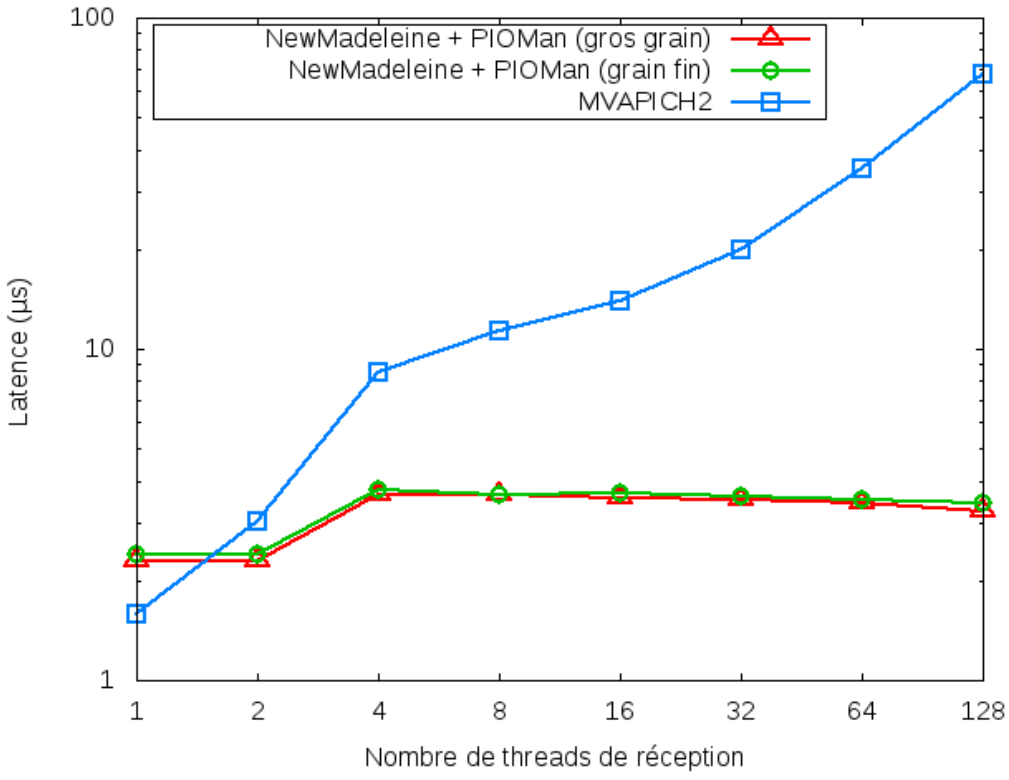


Micro-Benchmarks
Attentes concurrentes

- N threads récepteurs
- 1 thread émetteur



Myrinet



InfiniBand