Expériences d'entrées-sorties sur machines massivement parallèles

Philippe.Wautelet@idris.fr

CNRS - IDRIS

Workshop « Masse de données : I/O, format de fichier, visualisation et archivage »
ENS Lyon / 13 janvier 2011

Sommaire

Problématique


TOP500
SUPERCOMPUTER SITES
Performance Development
100 PFlops
10 PFlops
1 PFlops
100 TFlops
10 TFlops
1 TFlops
100 GFlops
10 GFlops
1 GFlops
100 MFlops
Performance
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
43654.20 TF
2566.00 TF
31112 GF
1.12 TF
59.7 GF
0.42 GF
#1
#500
Sum
15/11/2010
http://www.top500.org/

Evolution technique

- La puissance de calcul des supercalculateurs double tous les ans (plus rapide que la loi de Moore)
- Le nombre de cœurs augmente rapidement (architectures massivement parallèles et *many-cores*)
- La mémoire par cœur stagne et commence à décroître.
- La capacité de stockage augmente beaucoup plus vite que la vitesse d'accès (les disques SSDs changeront peut être cela)
- Le débit vers les disques augmente moins vite que la puissance de calcul

Conséquences

- La quantité de données générée augmente avec la puissance de calcul (moins vite heureusement)
- A l'IDRIS : la quantité de données stockée double tous les 2 ans et le nombre de fichiers est multiplié par 1,6 tous les 2 ans
- L'approche classique un fichier/processus génère de plus en plus de fichiers entraînant une saturation des serveurs de méta-données (trop d'accès simultanés et risques de crash)
- Moins de mémoire par cœur et plus de cœurs risque de réduire la taille moyenne des fichiers
- Trop de fichiers = saturation des systèmes de fichiers (nombre maximum de fichiers supporté et espace gaspillé à cause des tailles de bloc fixe)
- Le temps passé dans les I/O croît (surtout pour les applications massivement parallèles)
- Les étapes de pré- et post-traitement deviennent de plus en plus lourdes (nécessité de parallélisme...)

Autres problèmes

- Difficile de gérer des millions de fichiers
- Les performances sur les I/O d'autres jobs, des effets de cache... peuvent avoir un impact très important sur les performances des entrées-sorties
- Les performances sont difficiles à mesurer (grande variabilité à cause de : impact autres jobs, effets caches/buffers système, mode dédié ou pas...)

Solutions partielles (hors I/O parallèles)

- N'écrire que ce qui est nécessaire
- Ecrire le moins souvent possible
- Réduire la précision (écrire en simple précision par exemple)
- Recalculer quand c'est plus rapide

Approches

Approches

- Fichiers séparés (1 ou plusieurs par processus)
- Processus maître collectant/distribuant les données
- Processus spécialisés
- MPI-I/O
- Parallel-NetCDF
- HDF5
- Autres bibliothèques (NetCDF-4, ADIOS, SIONlib...)

Fichiers séparés

Principe

La méthode la plus simple consiste à ce que tous les processus lisent et écrivent leurs données indépendamment les uns des autres dans des fichiers séparés.

Avantages

- Simplicité de l'implémentation
- Peu de changements par rapport au code séquentiel
- Parallélisation triviale
- Souvent la plus performante
- Pas de bibliothèque spécifique à installer

Fichiers séparés

Inconvénients

- Grand nombre de fichiers
- Dépendance vis-à-vis du nombre de processus
- Pré- et post-traitements
- Données non-portables (*endianness*, taille des types...)
- Format de fichier non normalisé (pérennité ?)

Usage conseillé

Cette approche n'est conseillée que pour des applications avec un niveau de parallélisme limité et pour des codes utilisés dans des projets de taille réduite.
A éviter dans les applications massivement parallèles

Processus maître

Principe

Un processus (généralement celui de rang 0) lit et écrit toutes les données. Les autres processus reçoivent toutes leurs données de celui-ci et lui envoient toutes les informations devant être écrites. Le processus peut participer aux phases de calcul.

Avantages

- Indépendant du nombre de processus (selon le format de fichier)
- Peu de fichiers
- Pré- et post-traitements simplifiés

Processus maître

Inconvénients

- Mauvaise extensibilité (le processus maître risque de faire attendre tous les autres)
- Problèmes de consommation mémoire au niveau du processus maître
- Très gros fichiers si peu de fichiers
- Peu performant
- Non parallèle (I/O séquentielles)
- Données non portables (*endianness*, taille des types...)
- Format de fichier non normalisé (pérennité ?)

Usage conseillé

Cette approche n'est conseillée que pour des applications avec un niveau de parallélisme limité car le processus maître devient rapidement un goulet d'étranglement pour toute l'application.
A proscrire totalement dans les applications massivement parallèles

Processus spécialisés

Principe

Un ou plusieurs processus se chargent des entrées-sorties. Les autres processus réalisent les calculs et échangent toutes les données à lire et à écrire avec ce(s) processus spécialisé(s). Selon l'approche suivie, chaque processus spécialisé s'occupe des I/O d'un groupe prédéfini de processus de calcul ou peut s'occuper de n'importe quel processus de calcul.

Avantages

- Peu de fichiers (si implémenté pour)
- Indépendant du nombre de processus (délicat mais possible)
- Pré- et post-traitement simplifié si peu de fichiers, voire partiellement réalisé par ces processus
- Performant si nombre de processus spécialisés bien choisi
- Sur certains systèmes, chaque processus peut écrire sur des disques séparés et éventuellement transférer en arrière-plan les données sur un espace partagé plus lent

Inconvénients

- Ressources de calcul inutilisées (si un processus par cœur)
- Implémentation non triviale (partage des groupe de processus, communications entre les processus d'I/O et les processus de calcul)
- Problèmes de consommation mémoire au niveau des processus d'I/O
- Dépendance du nombre de processus (sauf si implémenté pour ne pas l'être)
- Pré- et post-traitements si nombreux fichiers
- Très gros fichiers si peu de fichiers
- Données non portables (*endianness*, taille des types...)
- Format de fichier non normalisé (pérennité ?)

Usage conseillé

Cette approche peut être très utile dans certains cas particuliers (par exemple présence de nombreux disques locaux et d'un système de fichiers partagés lent). C'est également une piste qui pourrait se montrer intéressante pour le massivement parallèle.

Principe

La bibliothèque MPI fournit un ensemble de sous-programmes pour réaliser des entrées-sorties parallèles à l'intérieur de toute application MPI. Les I/O peuvent se faire avec des opérations collectives ou indépendantes.

Avantages

- Peu de fichiers
- Indépendant du nombre de processus (si implémenté pour)
- Performance (théorique) élevée
- Optimisations tenant compte de l'architecture directement intégrées dans l'implémentation
- Pré- et post-traitements simplifiés
- Approche similaire au passage de messages
- Types dérivés MPI directement utilisables
- Appels non-bloquants disponibles

Inconvénients

- Relecture des données dans une application non MPI (problème généralement surestimé)
- Données non portables (*endianness*, taille des types...). En théorie, portable en mode *external32*, mais rarement implémenté.
- Format de fichier non normalisé (pérennité ?)
- Très gros fichiers

Usage conseillé

MPI-I/O est conseillé pour toutes les applications parallèles. Cependant, pour des applications utilisées par une large communauté et devant être pérennes, il est conseillé de passer aux bibliothèques Parallel-NetCDF ou HDF5.

Parallel-NetCDF

Principe

Parallel-NetCDF est une bibliothèque d'entrées-sorties parallèles utilisant MPI-I/O pour lire et écrire des fichiers. Elle travaille avec des fichiers au format NetCDF (légèrement modifié).

Avantages

- Peu de fichiers
- Indépendant du nombre de processus (si implémenté pour)
- Utilise MPI-I/O $\Rightarrow$ optimisations MPI-I/O
- Pré- et post-traitements simplifiés
- Données portables (grâce au format de fichier NetCDF)
- Format de fichier normalisé et auto-documenté
- Format de fichier assez simple

Inconvénients

- Utilisation contraignante (définition des dimensions, des variables...)
- Manque de souplesse
- Difficile d'obtenir de bonnes performances
- Documentation incomplète
- Types de données manquants (entiers sur 8 octets...)
- Légers surcoûts par rapport à MPI-I/O (taille et performance)
- Très gros fichiers

Usage conseillé

Parallel-NetCDF est conseillé pour tous les projets ayant besoin d'un format de fichier portable et pérenne. La simplicité de ce format rend son utilisation plus simple que HDF5, mais bien moins riche.

Principe

HDF5 est une bibliothèque d'entrées-sorties parallèles utilisant MPI-I/O pour lire et écrire des fichiers. Elle travaille avec des fichiers au format HDF5.

Avantages

- Peu de fichiers
- Indépendant du nombre de processus (si implémenté pour)
- Utilise MPI-I/O $\Rightarrow$ optimisations MPI-I/O
- Pré- et post-traitements simplifiés
- Données portables (grâce au format de fichier HDF5)
- Format de fichier normalisé et auto-documenté
- Format de fichier très riche
- Nombreuses fonctionnalités

Inconvénients

- Complexe à utiliser
- Difficile d'obtenir de bonnes performances
- Légers surcoûts par rapport à MPI-I/O (taille et performance)
- Très gros fichiers

Usage conseillé

HDF5 est conseillé pour tous les projets ayant besoin d'un format de fichier portable et pérenne. Ce format étant très riche, il est parfaitement adapté aux grosses applications complexes. Son utilisation est néanmoins réservée à des utilisateurs/communautés prêts à s'investir dans son utilisation.

Autres bibliothèques

- NetCDF-4 (travaille avec des fichiers au format HDF5)
- VTK (plutôt axé visualisation, mais support du parallélisme)
- SIONlib (rassemble de façon transparente dans un ou quelques fichiers l'ensemble des données des différents processus)
- ADIOS (structure de données dans fichier XML, mode de lecture/écriture au choix (MPI-I/O, POSIX, AIO...))

Tests de performances

Configuration IDRIS

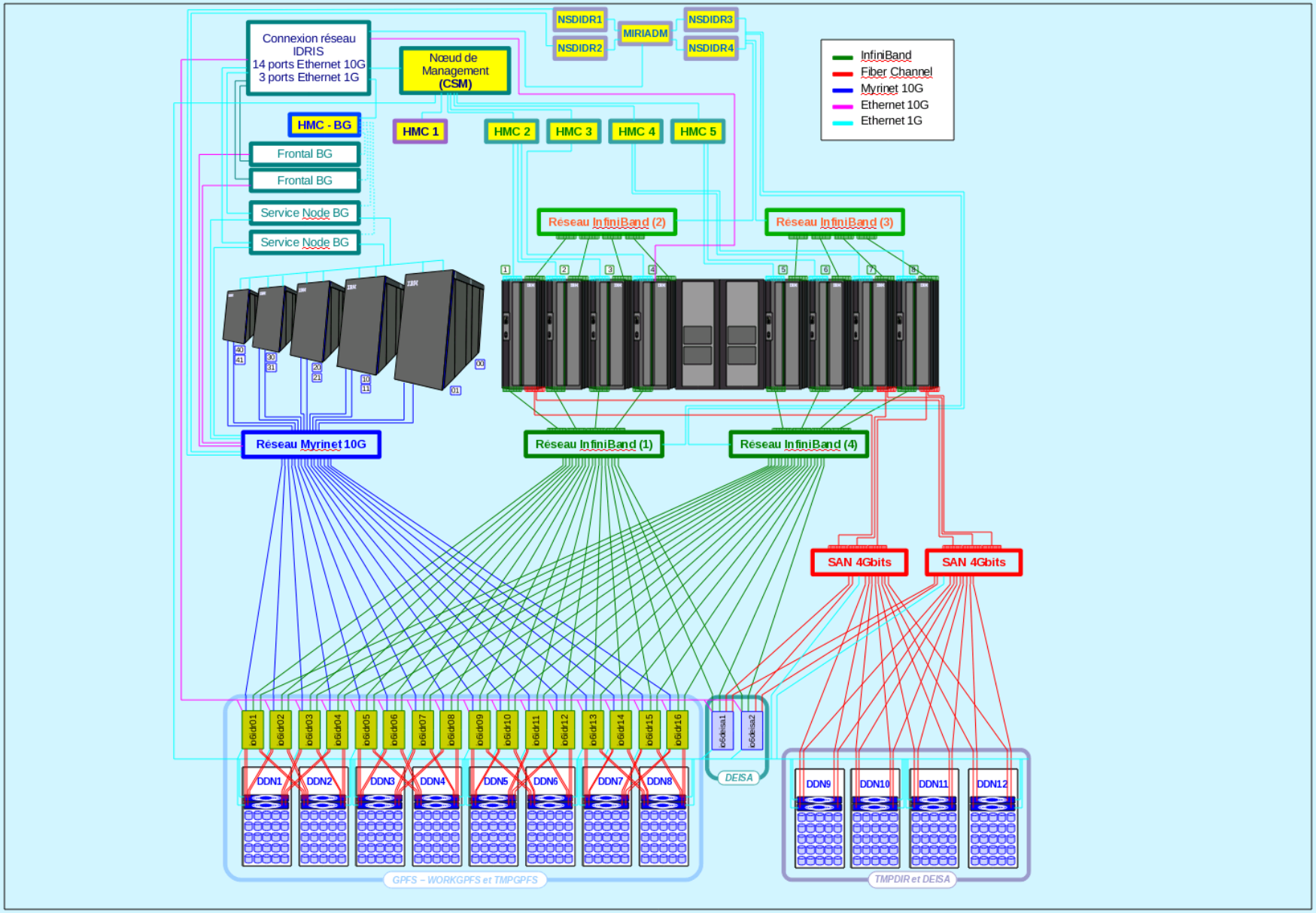
Babel : IBM Blue Gene/P

Vargas : IBM Power6

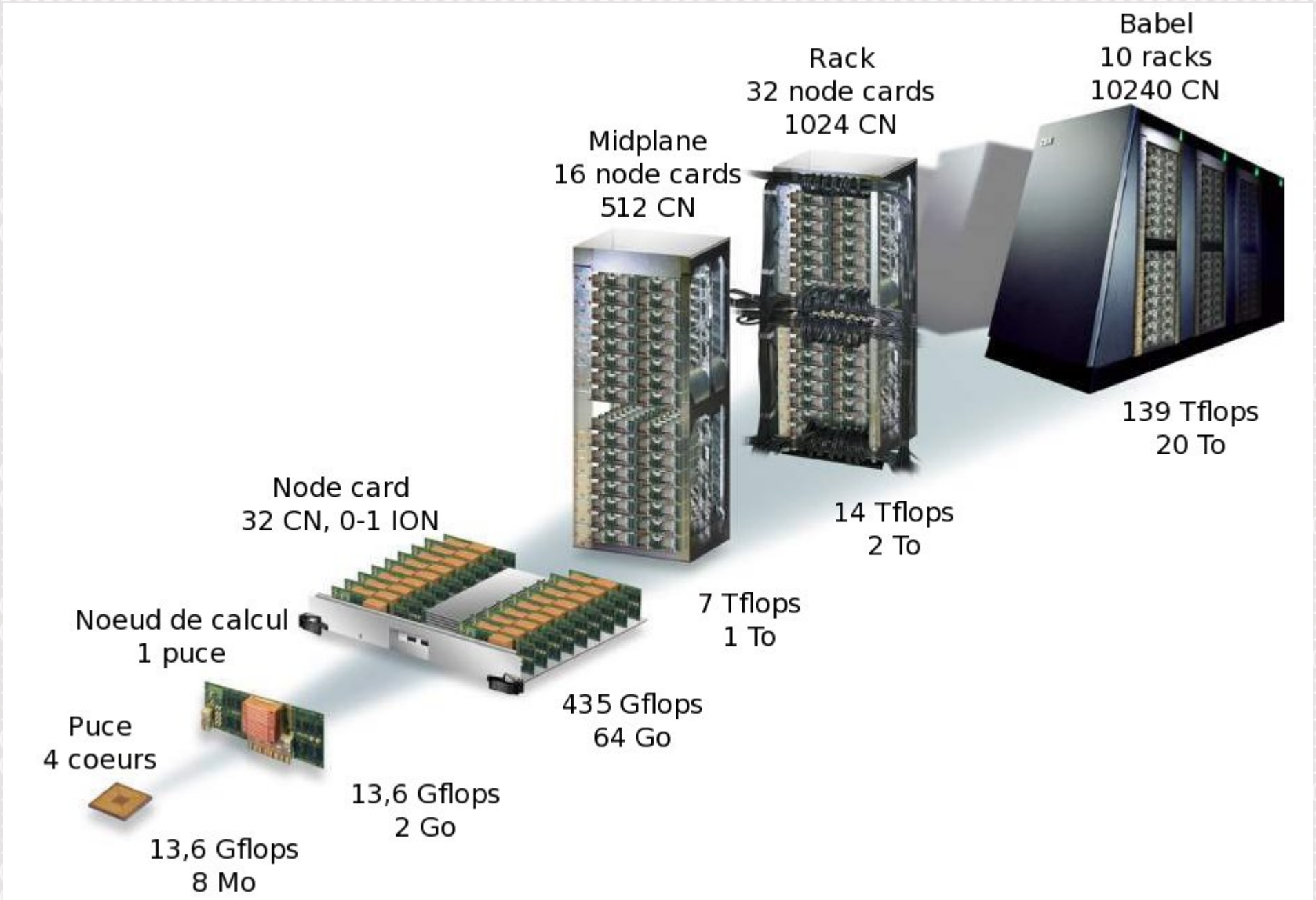
Chiffres importants

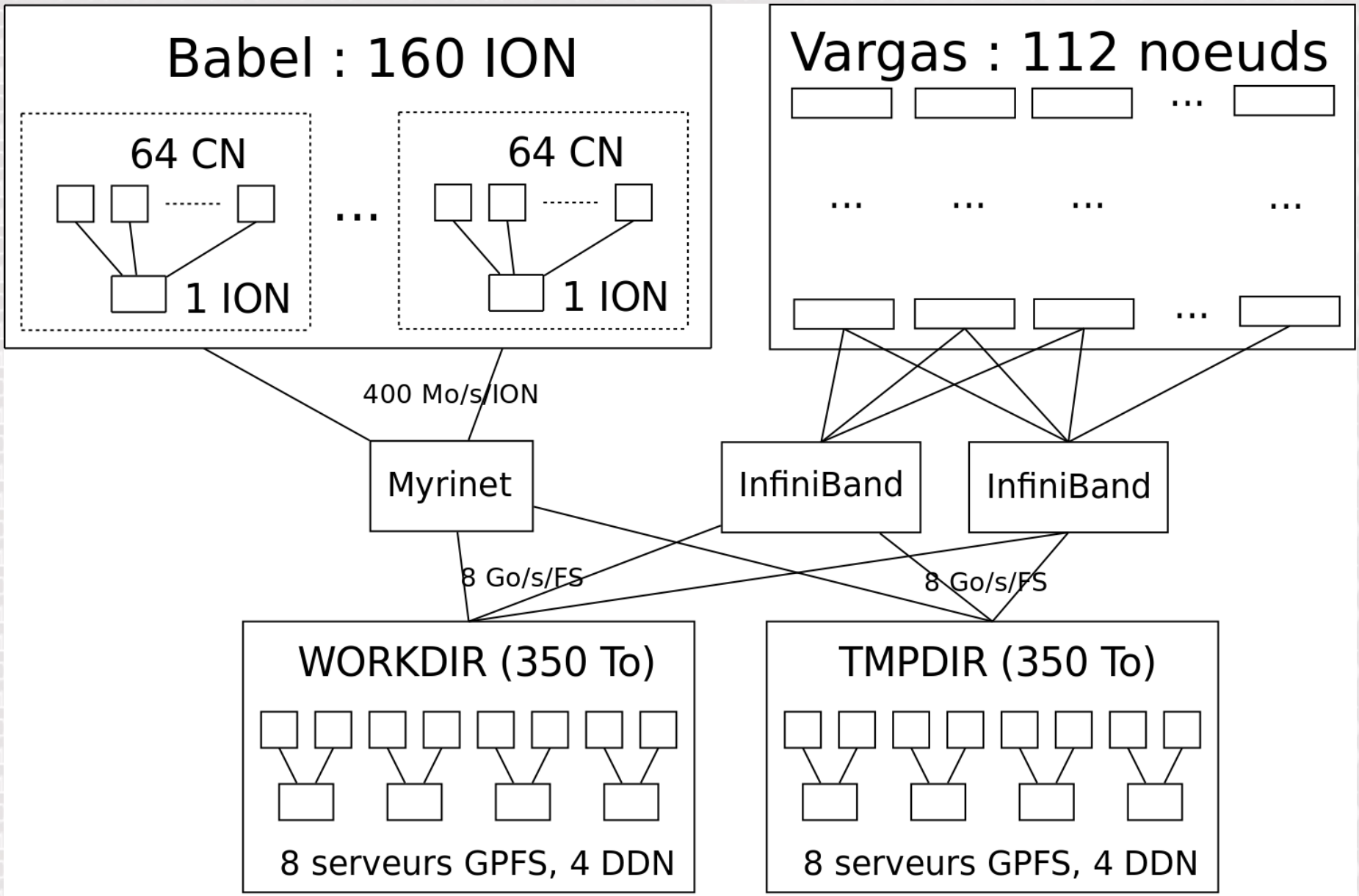
- **Babel : 10 racks Blue Gene/P** :
  - 10.240 nœuds
  - 40.960 cœurs
  - 20 To
  - 139 Tflop/s
  - 30 kW/rack (300 kW pour les 10 racks)
- **Vargas : 8 racks Power6** :
  - 112 nœuds
  - 3.584 cœurs
  - 18 To
  - 68 Tflop/s
  - 75 kW/rack (600 kW pour les 8 racks)
- 800 To sur disques partagés BG/P et P6
- 400 To pour Power6
- 950 kW pour la configuration complète (hors climatisation)

Configuration IDRIS



Architecture Blue Gene/P




Babel : 160 ION
64 CN
1 ION
64 CN
1 ION
Vargas : 112 noeuds
400 Mo/s/ION
Myrinet
InfiniBand
InfiniBand
8 Go/s/FS
8 Go/s/FS
WORKDIR (350 To)
8 serveurs GPFS, 4 DDN
TMPDIR (350 To)
8 serveurs GPFS, 4 DDN

Entrées/sorties côté clients

Côté clients BG/P (nœuds d'I/O)

- 1 I/O node tous les 64 compute nodes $\Rightarrow$ 16/rack
- les ION gèrent tous les I/O (transparent côté CN)
- ION sont points de sortie des CN
- ION fournissent sockets aux CN
- ION : jusqu'à 400 Mo/s (mesuré, 1,2 Go/s théorique)
- Tous les transferts entre les CN et les ION passent par le réseau collectif

Côté clients Vargas

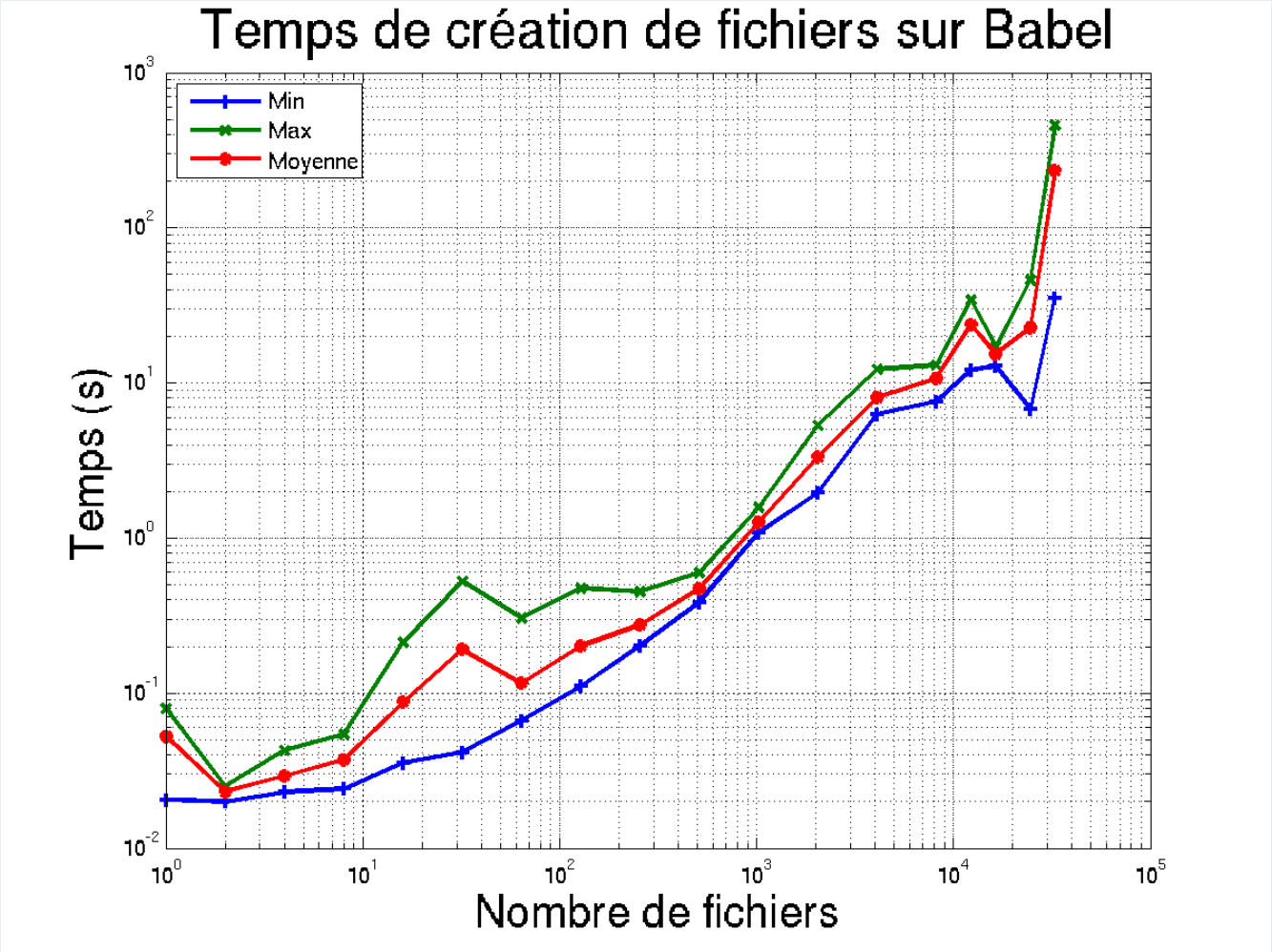
- 2 liens InfiniBand 4xDDR par nœud (2x16 Gbit/s théorique)

Côté serveurs I/O

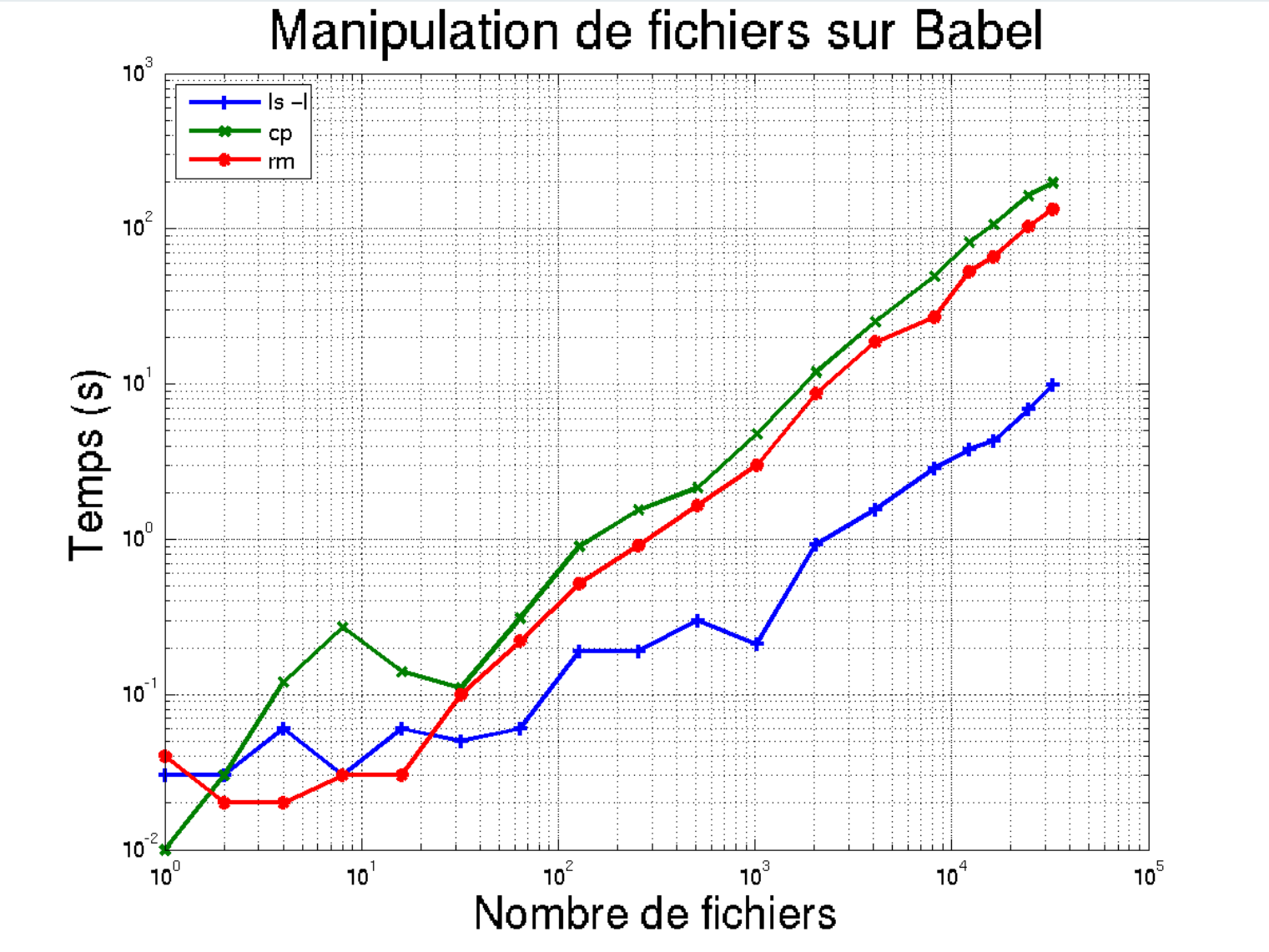
- 16 serveurs GPFS (nœuds 4 cœurs Power6 4,2 GHz)
- 8 baies DDN (2 serveurs GPFS/baie) à 2 Go/s/baie
- 800 To en GPFS partagés entre BG/P et P6
- 346 To pour le WORKDIR et 346 To pour le TMPDIR (8 Go/s chacun)
- Taille de bloc (WORKDIR et TMPDIR) : 2 Mo
- Plus petite écriture : taille_bloc/32 (2 Mo / 32 = 64 ko)

Temps de création d'un fichier par processus

Chaque processus crée un nouveau fichier et le ferme. Le temps mesuré correspond au processus le plus lent.

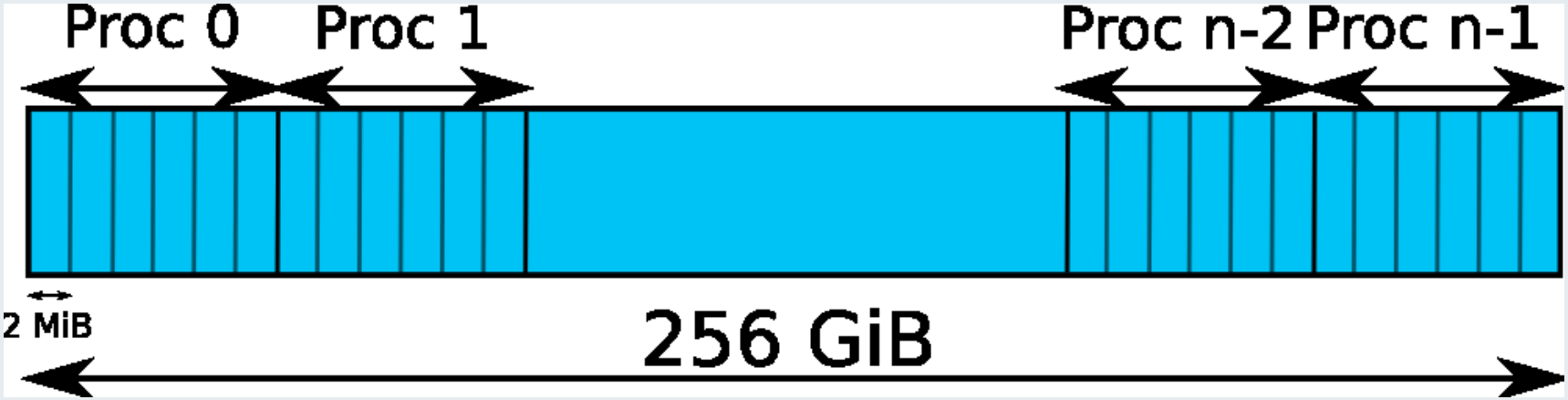


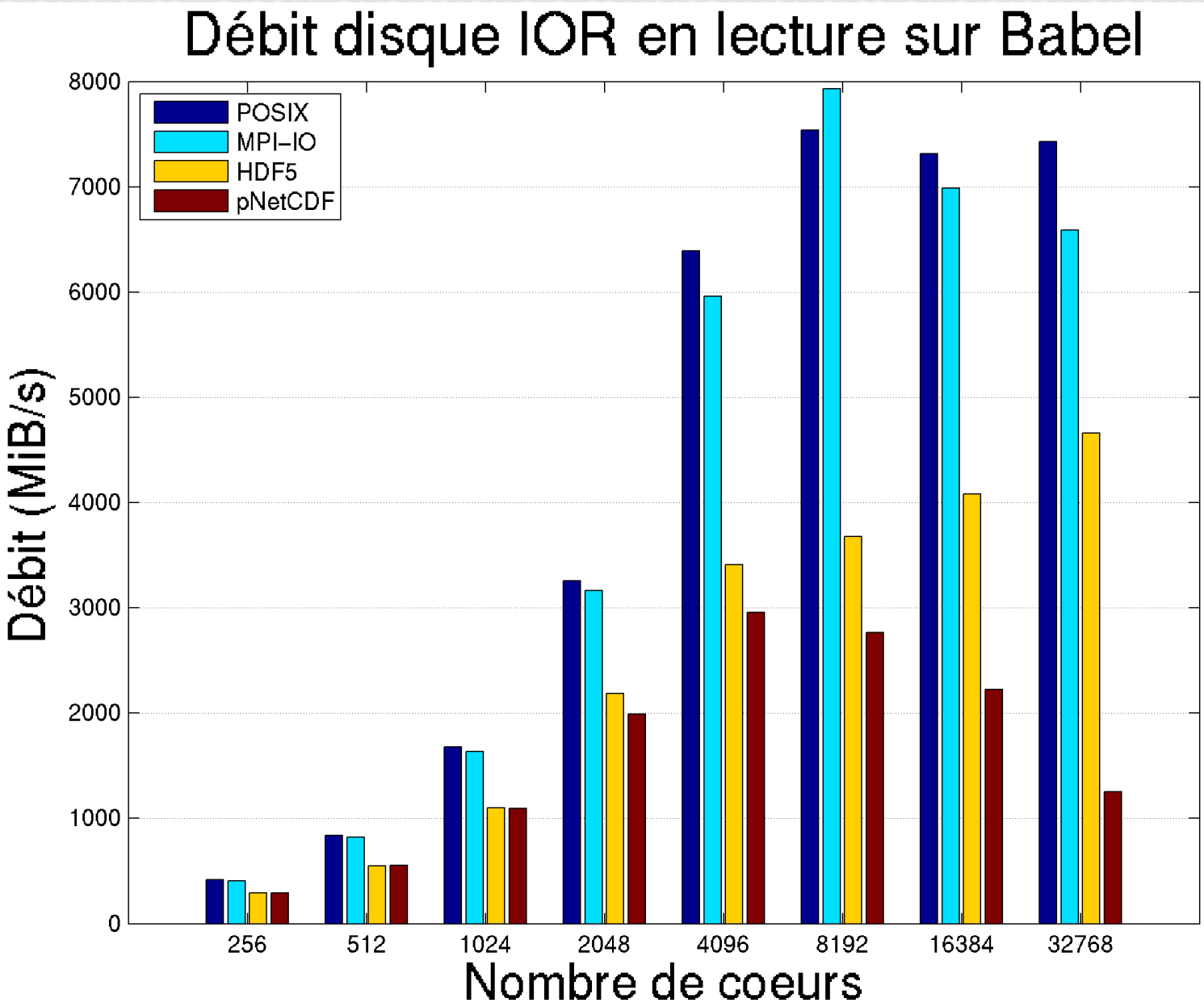
Temps de manipulation d'une série de fichiers

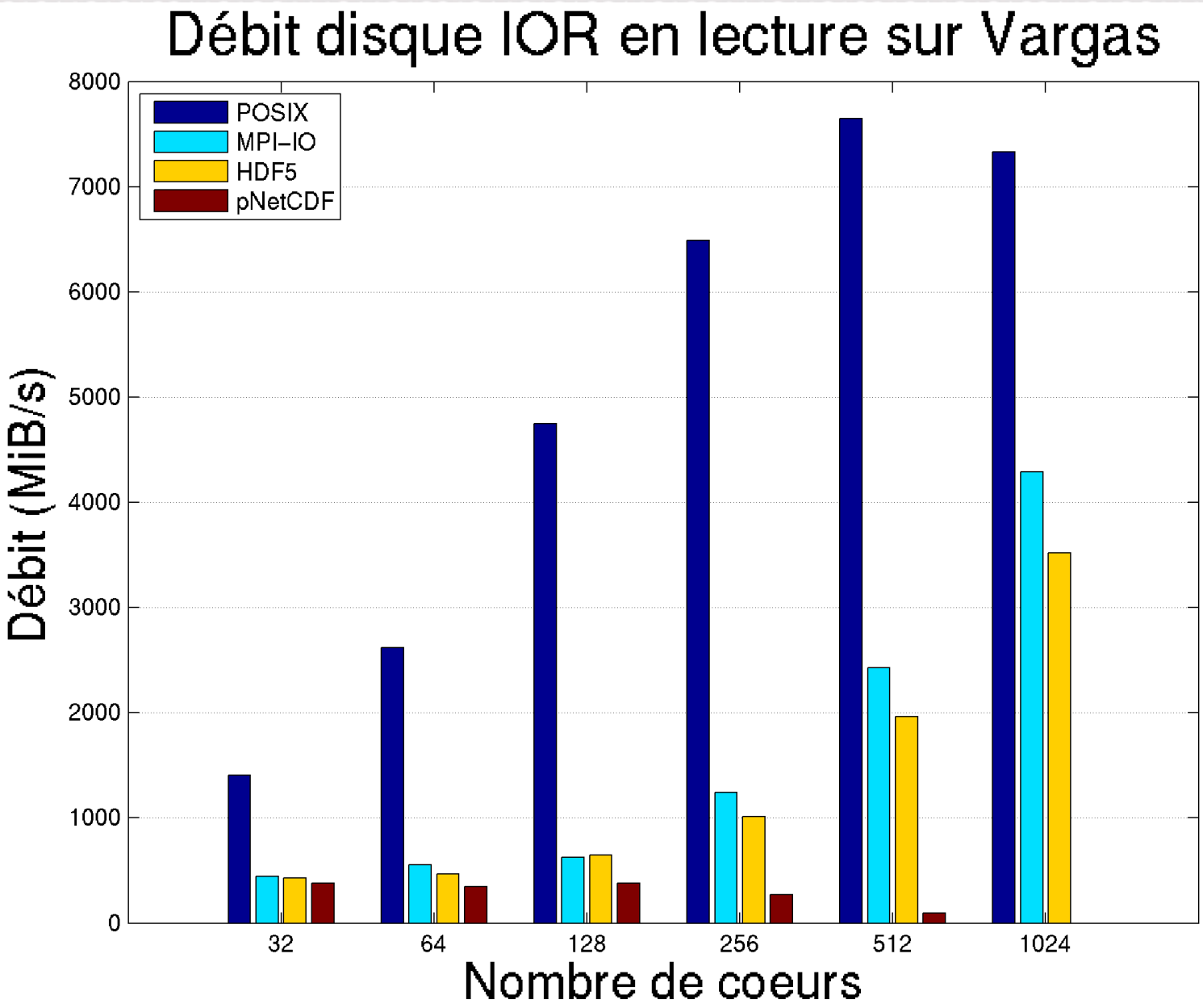


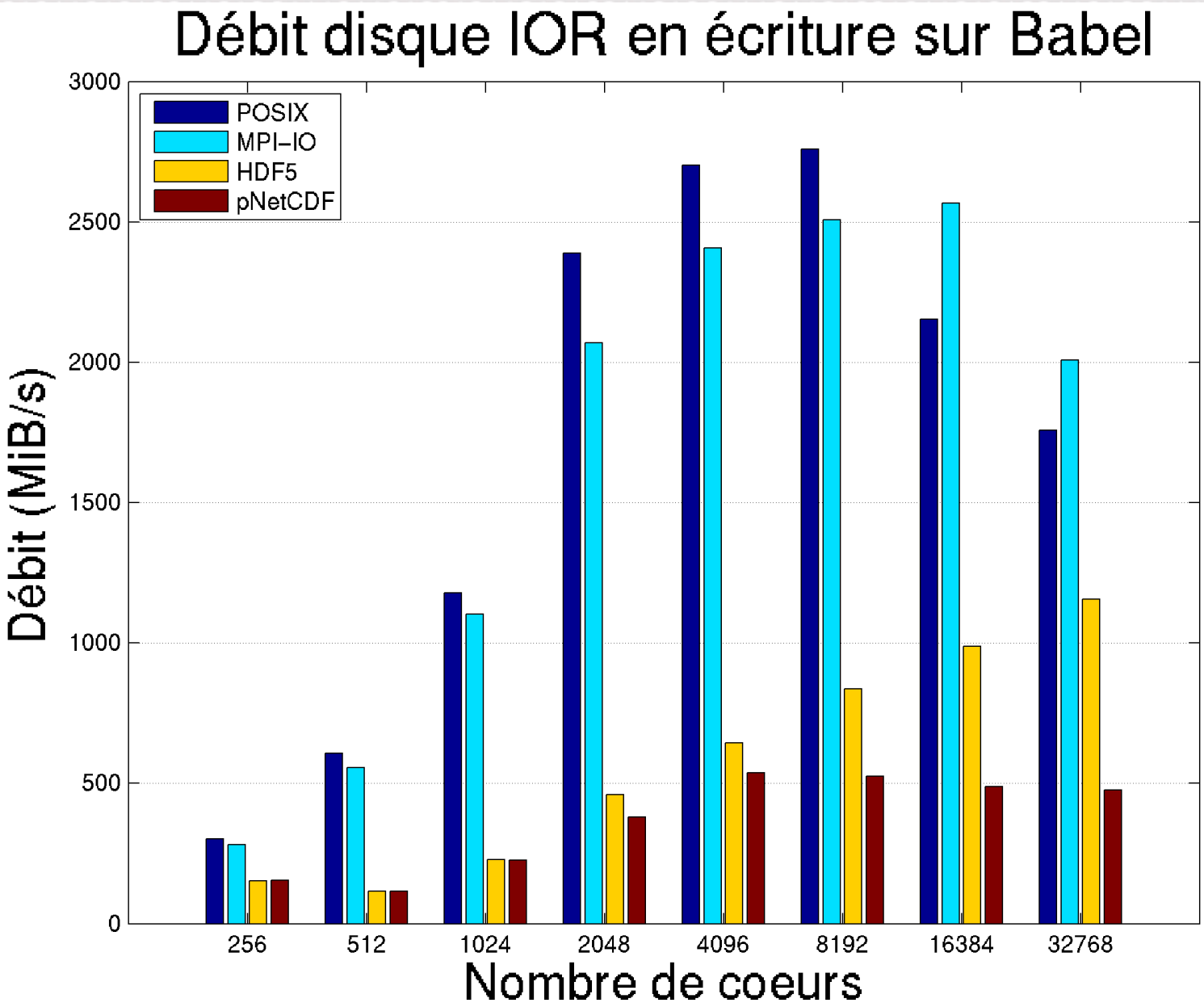
Présentation du benchmark IOR

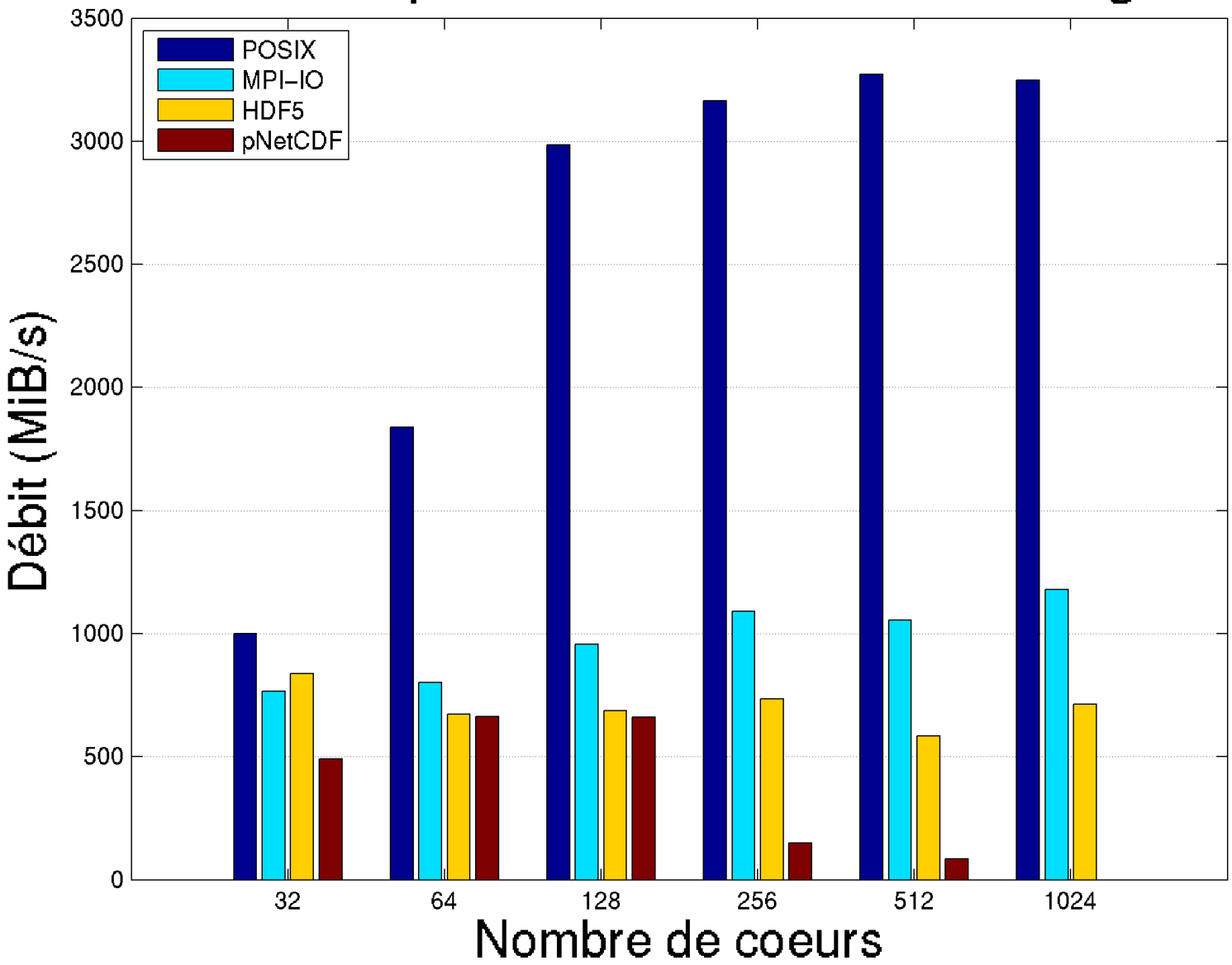
- IOR est un benchmark très utilisé pour les mesures de performances de systèmes de fichiers parallèles
- Supporte les fichiers POSIX séparés, MPI-I/O, HDF5 et Parallel-NetCDF
- Mode collectif ou non
- Chaque processus écrit son morceau de fichier de taille *blocksize* en l'écrivant par bloc de taille *xfersize*
- Tailles choisies ici : taille totale : 256 GiB, $xfersize = 2MiB$ (correspondant à la taille d'un bloc GPFS)
- Tous les tests ont été effectués en mode individuel




Débit disque IOR en lecture sur Babel
POSIX
MPI-IO
HDF5
pNetCDF
Débit (MiB/s)
8000
7000
6000
5000
4000
3000
2000
1000
0
256
512
1024
2048
4096
8192
16384
32768
Nombre de coeurs


Débit disque IOR en lecture sur Vargas
POSIX
MPI-IO
HDF5
pNetCDF
Débit (MiB/s)
8000
7000
6000
5000
4000
3000
2000
1000
0
32
64
128
256
512
1024
Nombre de coeurs


Débit disque IOR en écriture sur Babel
POSIX
MPI-IO
HDF5
pNetCDF
Débit (MiB/s)
3000
2500
2000
1500
1000
500
0
256
512
1024
2048
4096
8192
16384
32768
Nombre de coeurs


Débit disque IOR en écriture sur Vargas
POSIX
MPI-IO
HDF5
pNetCDF
Débit (MiB/s)
3500
3000
2500
2000
1500
1000
500
0
32
64
128
256
512
1024
Nombre de coeurs

Commentaires tests bas niveau IOR

- *POSIX* : approche la plus performante dans (presque) tous les cas. S'approche de la crête en lecture. Les débits maximum sont atteints dès 2 racks Blue Gene/P ou 4 nœuds Power 6.
- *MPI-I/O* : assez performant. Aussi bon que POSIX sur Blue Gene/P, nettement moins sur Power 6. Performance croissante avec le nombre de processus, sauf en écriture sur Power 6 où elle est assez stable.
- *HDF5* : moins bon que MPI-I/O. Performance croissante avec le nombre de processus, sauf en écriture sur Power 6 où elle est assez stable.
- *Parallel-NetCDF* : assez mauvais. Performance diminuant au-delà d'un rack Blue Gene/P et de 4 nœuds Power 6 surtout en écriture.
- Pour rappel : tous les tests ont été effectués en mode individuel.

Présentation

- RAMSES est une application de calcul d'astrophysique développée au départ par Romain Teyssier (CEA) sous licence CeCILL.
- Résout les équations d'Euler en présence d'auto-gravité et de refroidissement traités comme termes sources dans les équations du moment et de l'énergie.
- Raffinement adaptatif de maillage (méthode AMR) avec équilibrage de charge et défragmentation de la mémoire dynamiques
- Méthode multi-grilles et gradient conjugué pour l'équation de Poisson
- Solveurs de Riemann (Lax-Friedrich, HLLC, exact) pour la dynamique des gaz adiabatiques
- Dynamique de particules (sans collisions) pour la matière noire et les étoiles
- Formation des étoiles, supernovae...
- Code utilisant MPI optimisé par l'IDRIS (entre autres) et tournant régulièrement sur plusieurs dizaines de milliers de processus

RAMSES est disponible sur le site http://irfu.cea.fr/Projets/Site_ramses/RAMSES.html

Entrées dans RAMSES (version originale)

RAMSES peut soit commencer une nouvelle simulation, soit repartir d'une solution précédente (*restart*)

- Dans tous les cas, lecture d'un fichier texte de type *namelist* Fortran contenant les paramètres de la simulation par tous les processus
- Pour une nouvelle simulation, lecture de quelques fichiers d'entrées par le processus 0 qui *broadcaste* vers les autres et lecture par tous les processus de quelques données dans un fichier d'entrées.
- Pour un *restart*, chaque processus lit son ou ses fichiers (entre 1 et 4 selon les options)

Sorties dans RAMSES (version originale)

- Les sorties sont effectuées au choix tous les *n* pas temps et/ou à certains instants dans le temps physique de la simulation
- Chaque processus écrit son/ses fichiers
- Selon les options, entre 1 et 4 fichiers sont écrits par processus
- 2 petits fichiers sont égalements écrits par le premier processus

Processus spécialisés : entrées

- Dans tous les cas, lecture d'un fichier texte de type *namelist* Fortran contenant les paramètres de la simulation par tous les processus
- Pour une nouvelle simulation, pas de changements : lecture de quelques fichiers d'entrées par le processus 0 qui *broadcaste* vers les autres et lecture par tous les processus de quelques données dans un fichier d'entrées.
- Pour un *restart*, les processus sont groupés et seul un processus par groupe lit ses fichiers à un moment donné (passage de jeton jusqu'à ce que tous les processus aient lu leurs données)

Processus spécialisés : sorties

- 2 types de processus : processus de calcul et processus d'I/O dans 2 communicateurs MPI distincts
- Ecritures remplacées par des envois aux processus d'I/O
- Les processus d'I/O peuvent écrire dans un espace disque qui leur est local
- Une fois toutes les données envoyées, les processus de calcul reprennent leurs opérations
- Les processus d'I/O peuvent pendant ce temps transférer les données locales vers un espace partagé

Bibliothèques parallèles

L'utilisation des bibliothèques d'entrées-sorties parallèles suivantes a été intégrée dans RAMSES :

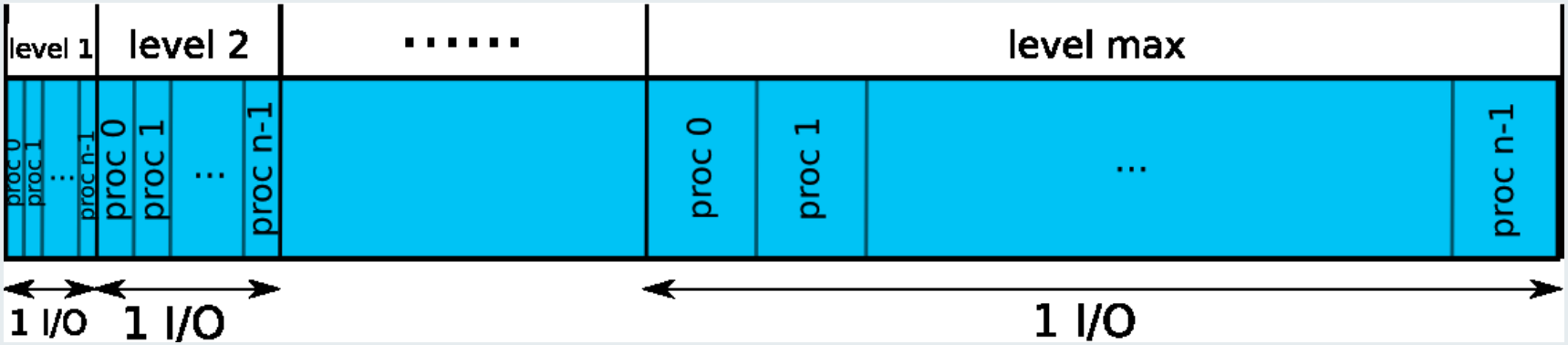
- MPI-I/O
- HDF5
- Parallel-NetCDF

Caractéristiques

- Selon les options, de 1 à 4 gros fichiers partagés entre tous les processus par sortie (chaque processus écrit sa partie)
- Toutes les I/O se font via les interfaces collectives (car les plus performantes dans les tests RAMSES au contraire des tests IOR)
- Actuellement, fichiers dépendant du nombre de processus (pas possible de faire un *restart* avec un nombre différent de processus)
- Certaines données sont encore redondantes entre les différents fichiers

Fichier AMR

- Un ensemble de paramètres identiques sur tous les processus (14 entiers, 22 doubles, un petit tableau d'entiers (10*nlevelmax) et une chaîne de caractères)
- Un ensemble de variables différentes sur chaque processus (8 entiers par processus et 3 structures de données (ncpu*nlevelmax entiers par processus)). Les valeurs sur les différents processus d'une même variable sont disposées les unes derrières les autres dans le fichier.
- Un ensemble de grosses structures de données (8 structures contenant entre 1 et 8 valeurs entières ou doubles par maille) pour un total de 156 octets * nombre de mailles. Elles sont structurées de la façon suivante :



Remarques : les grilles (ou niveaux) sont de tailles croissantes (facteur 8 pour les grilles pleines) et certains processus peuvent ne pas contenir de valeurs pour certains niveaux (dépendants du raffinement).

Structure des fichiers (utilisés dans les tests)

Fichier hydro

- Un ensemble de paramètres identiques sur tous les processus (5 entiers, 1 double)
- Un ensemble de variables différentes sur chaque processus (2 structures de données (ncpu*nlevelmax et 100*nlevelmax entiers par processus)). Les valeurs sur les différents processus d'une même variable sont disposées les unes derrières les autres dans le fichier.
- Un ensemble de grosses structures de données (5 structures contenant 8 valeurs double précision par maille) pour un total de 320 octets * nombre de mailles. Elles sont structurées de la même façon que dans AMR sauf que les valeurs sont groupées par 8 (donc 8 fois plus de valeurs par niveau).

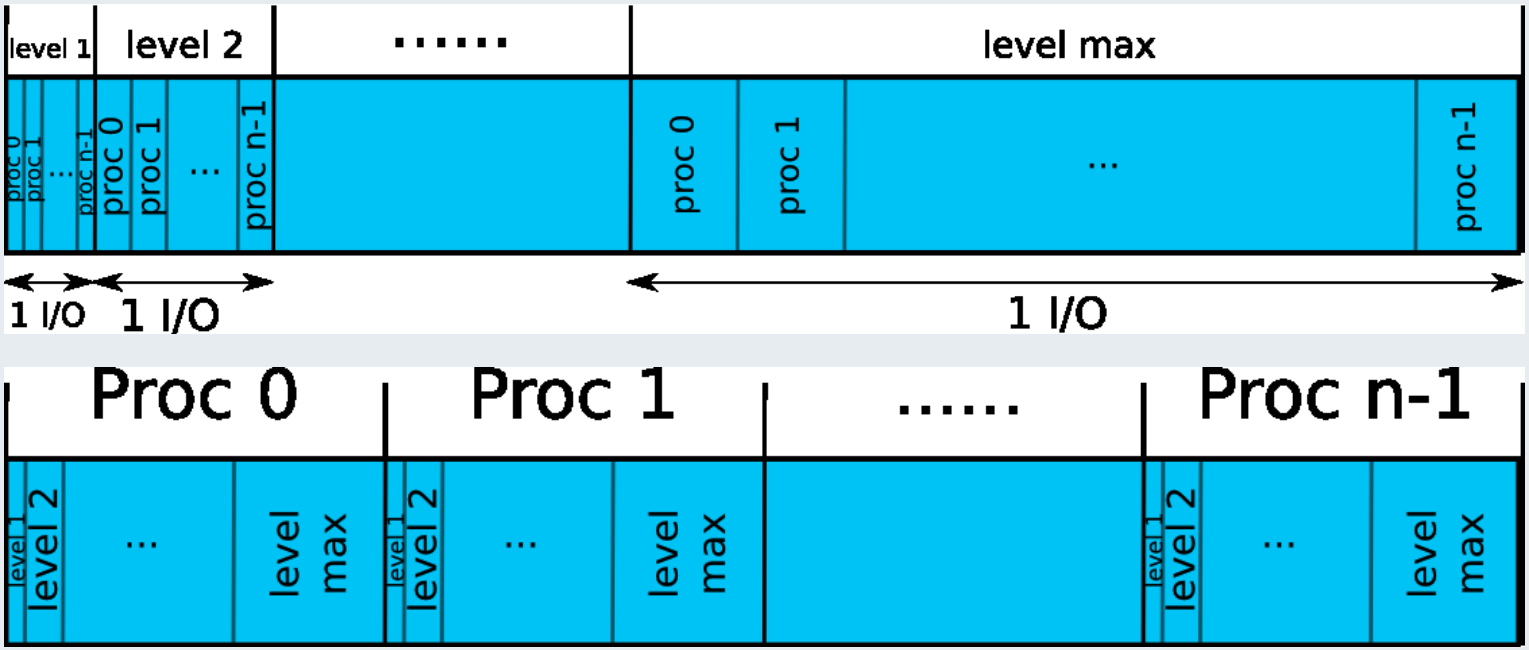
Fichier AMR vs hydro

- Le fichier AMR est plus complexe et contient plus de variables différentes
- Les 2 premières sections sont beaucoup plus importantes dans le fichier AMR (mais restent relativement petites en taille absolue)
- Le fichier hydro est environ 2 fois plus gros (320 octets par maille au lieu de 156 pour la $3^{ème}$ section)
- La granularité des écritures est nettement plus grosse dans le fichier hydro

Comparaison des approches

Tests MPI-I/O effectués sur le fichier hydro sur Blue Gene/P avec 2048 cœurs et cas test sedov3d 1024 (taille du fichier : 54,9 GiB)

	Groupes par niveau			Groupes par processus	
	write_at_all	write_at	write_ordered	write_at_all	write_at
Débit (MiB/s)	810	158	197	106	342



Définition des hints MPI-I/O

- Les hints MPI-I/O sont des indications que l'ont peut fournir à la couche MPI-I/O pour optimiser les performances (ils influencent aussi les bibliothèques reposant sur MPI-I/O comme HDF5 et Parallel-NetCDF)
- Chaque implémentation fournit sa liste de hints (parfois pas ou peu documentés)
- L'impact sur les performances peut être majeur
- Les hints peuvent être choisis lors de l'ouverture des fichiers, pour certains en cours d'utilisation des fichiers ou parfois par des variables d'environnement
- Chaque hint a une valeur par défaut

Hints sur Blue Gene/P

- Testés : romio_cb_read/romio_cb_write, romio_ds_read/romio_ds_write, cb_nodes et cb_buffer_size
- Non testés : romio_cb_fr_types, romio_cb_fr_alignment, romio_cb_alltoall, romio_cb_pfr, romio_cb_ds_threshold, romio_no_indep_rw, ind_rd_buffer_size, ind_wr_buffer_size

Hints sur Power 6

- Testés : IBM_io_buffer_size, IBM_largeblock_io et IBM_sparse_access
- Non testé/non lié aux performances : file_perm

Hints sur Blue Gene/P : exemple

Tests d'écriture HDF5 sur le fichier AMR (sedov3d 1024^3 2048 cœurs)

hint	*Temps (s)*
romio_cb_write = enable romio_ds_write = automatic	47.4
romio_cb_write = disable romio_ds_write = automatic	6784.7
romio_cb_write = automatic romio_ds_write = automatic	6273.5
romio_cb_write = enable romio_ds_write = enable	53.5
romio_cb_write = enable romio_ds_write = disable	52.8

La première ligne correspond aux valeurs par défaut.

Hints sur Blue Gene/P : exemple

Tests de lecture HDF5 sur le fichier AMR (sedov3d 1024^3 2048 cœurs)

hint	*Temps (s)*
romio_cb_read = enable romio_ds_read = automatic	69.9
romio_cb_read = disable romio_ds_read = automatic	315.2
romio_cb_read = automatic romio_ds_read = automatic	324.3
romio_cb_read = enable romio_ds_read = disable	61.3
romio_cb_read = enable romio_ds_read = enable	67.5

La première ligne correspond aux valeurs par défaut.

Choix des hints pour les tests

- Blue Gene/P
  - En écriture : valeurs par défaut
  - En lecture : romio_ds_read = disable. Pour fichier hydro uniquement : romio_cb_read = disable
- Power 6
  - En écriture : IBM_largeblock_io = true sauf pour HDF5 qui garde la valeur par défaut (sinon plantages)
  - En lecture : IBM_sparse_access = true sauf pour HDF5 qui garde la valeur par défaut (sinon plantages)

Taille des fichiers de sortie

Taille (en GiB) des fichiers de sortie HDF5 sur 1024 cœurs en fonction de la taille du problème étudié (tous les tests sont basés sur un sedov3d).

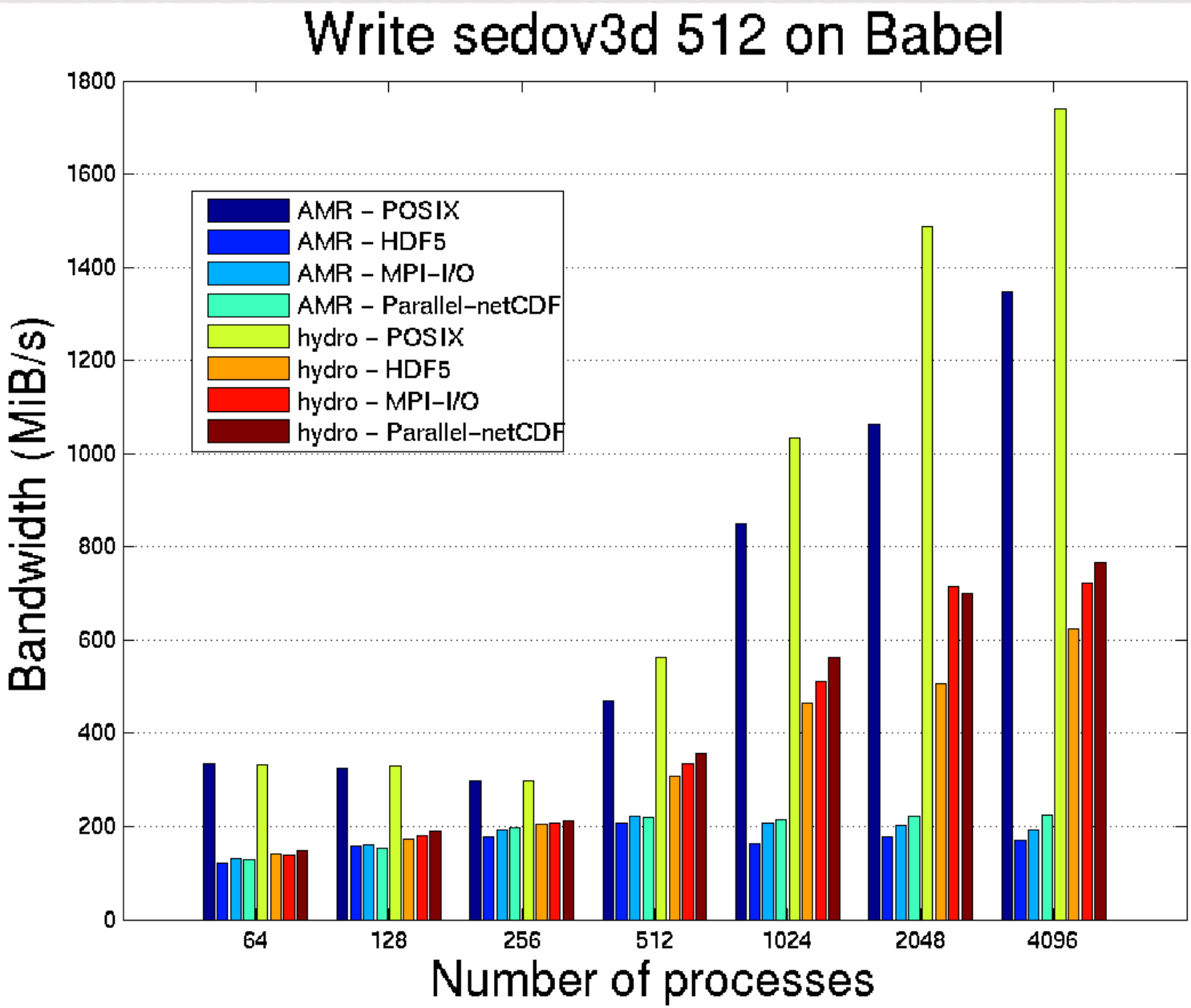
	128^3	256^3	512^3	1024^3	2048^3
AMR	0,213	0,707	3,814	25,860	191,8
hydro	0,299	1,292	7,646	49,150	393,2

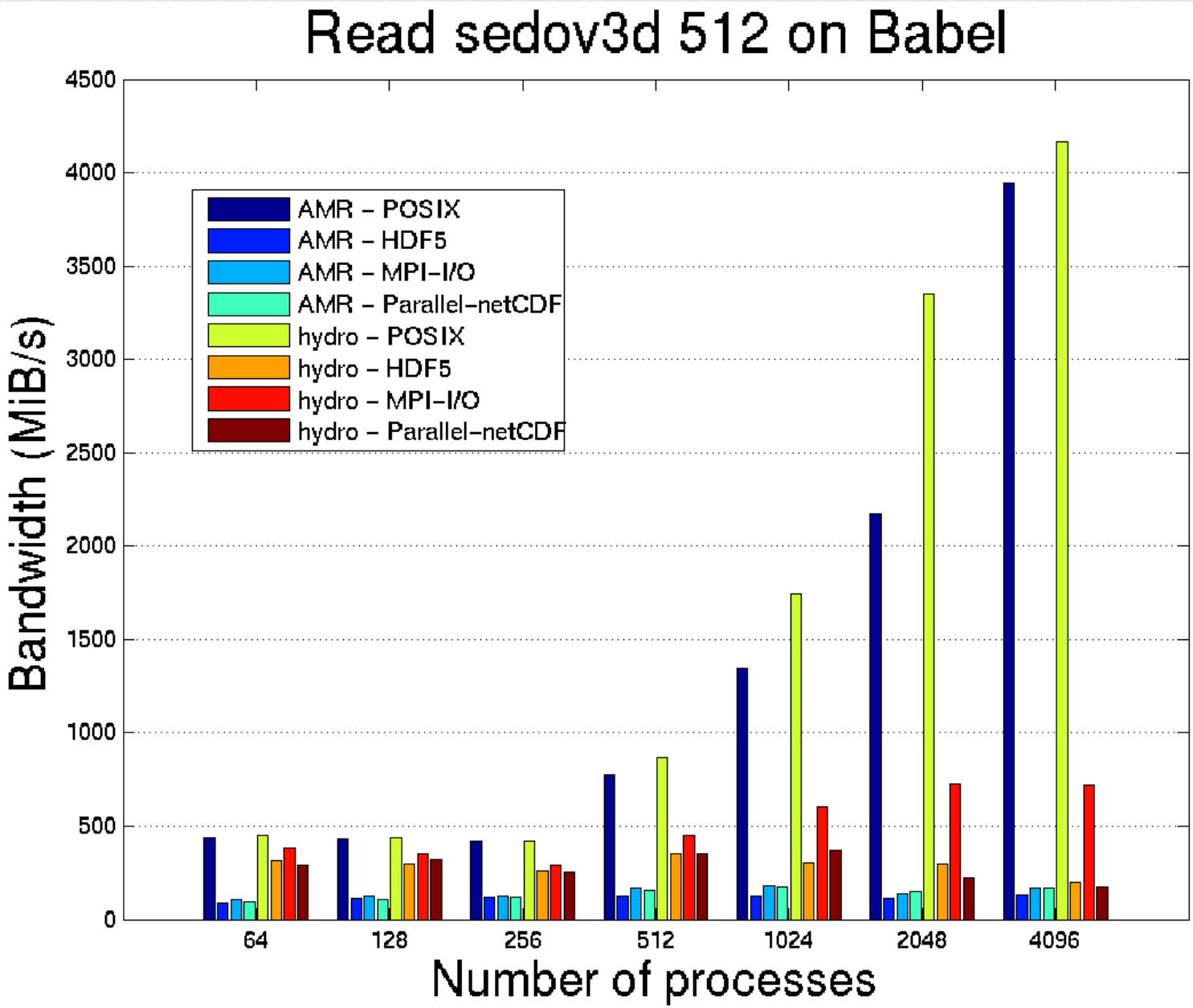
Remarque : la taille des fichiers dépend également du nombre de processus à cause des variables et structures différentes sur chacun d'entre-eux.

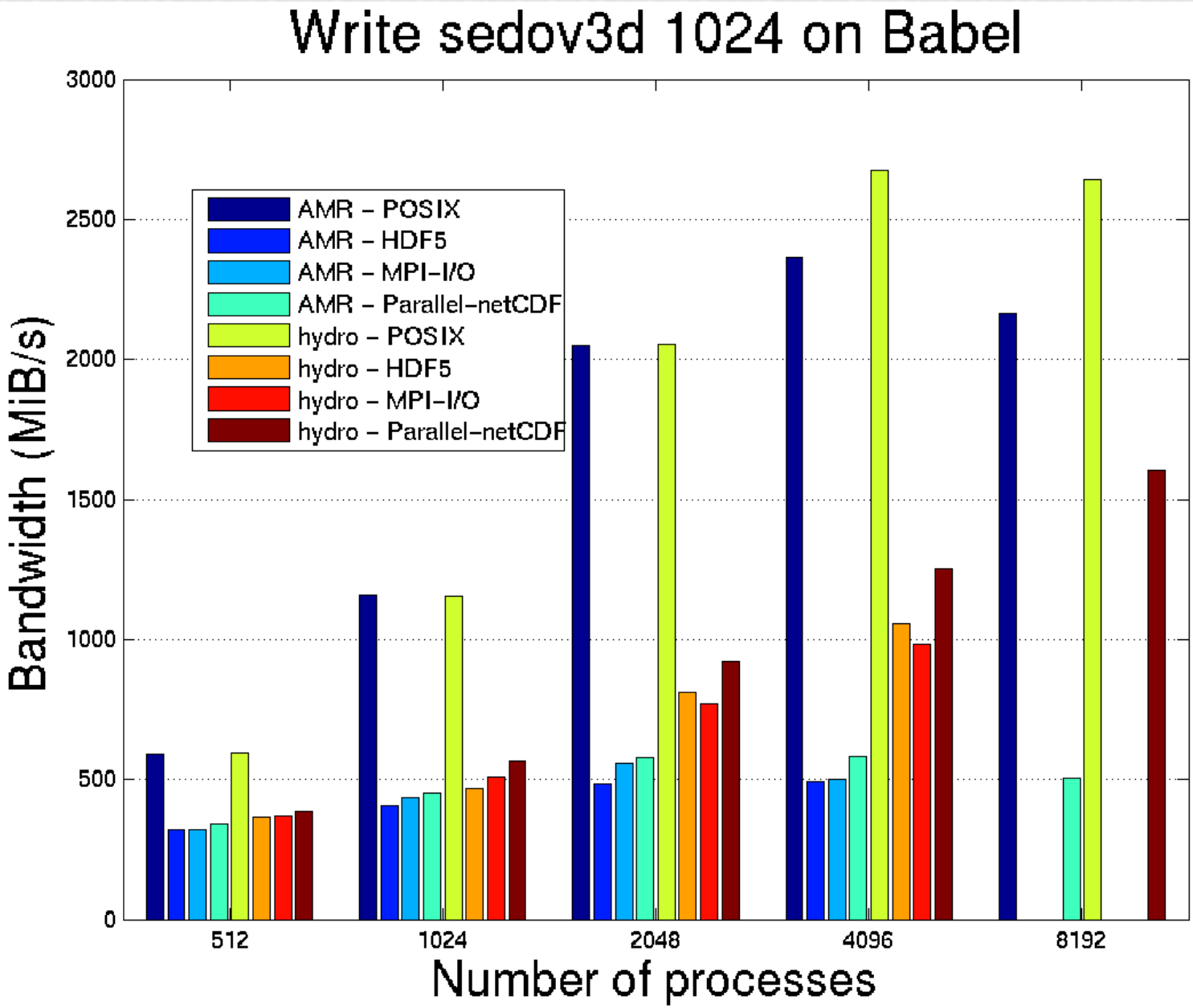
Procédure de test

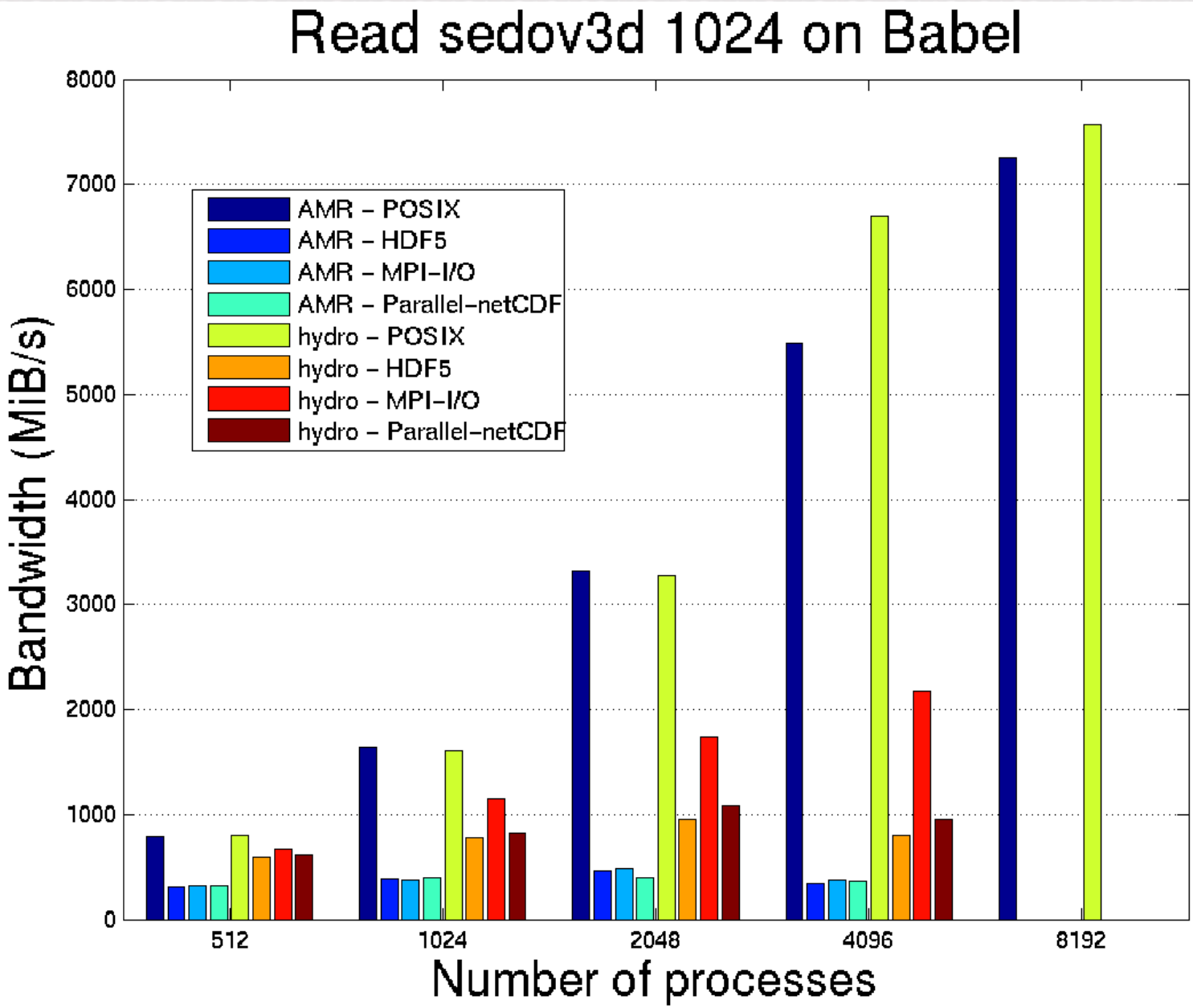
- Cas test choisi : sedov3d (explosion dans une boîte)
- Taille du domaine : 128^3, 256^3, 512^3, 1024^3 et 2048^3
- Nombre de processus : du plus petit au plus grand possible (pour les plus petits domaines, le nombre de processus a été volontairement limité)
- Pas de raffinement de maillage adaptatif
- Toutes les mesures effectuées 5 fois
- Runs non dédiés sur les machines (machines en production normale)
- Versions utilisées : HDF5 1.8.5 et Parallel-NetCDF 1.2.0

Ecriture Sedov3d 512^3 sur Blue Gene/P

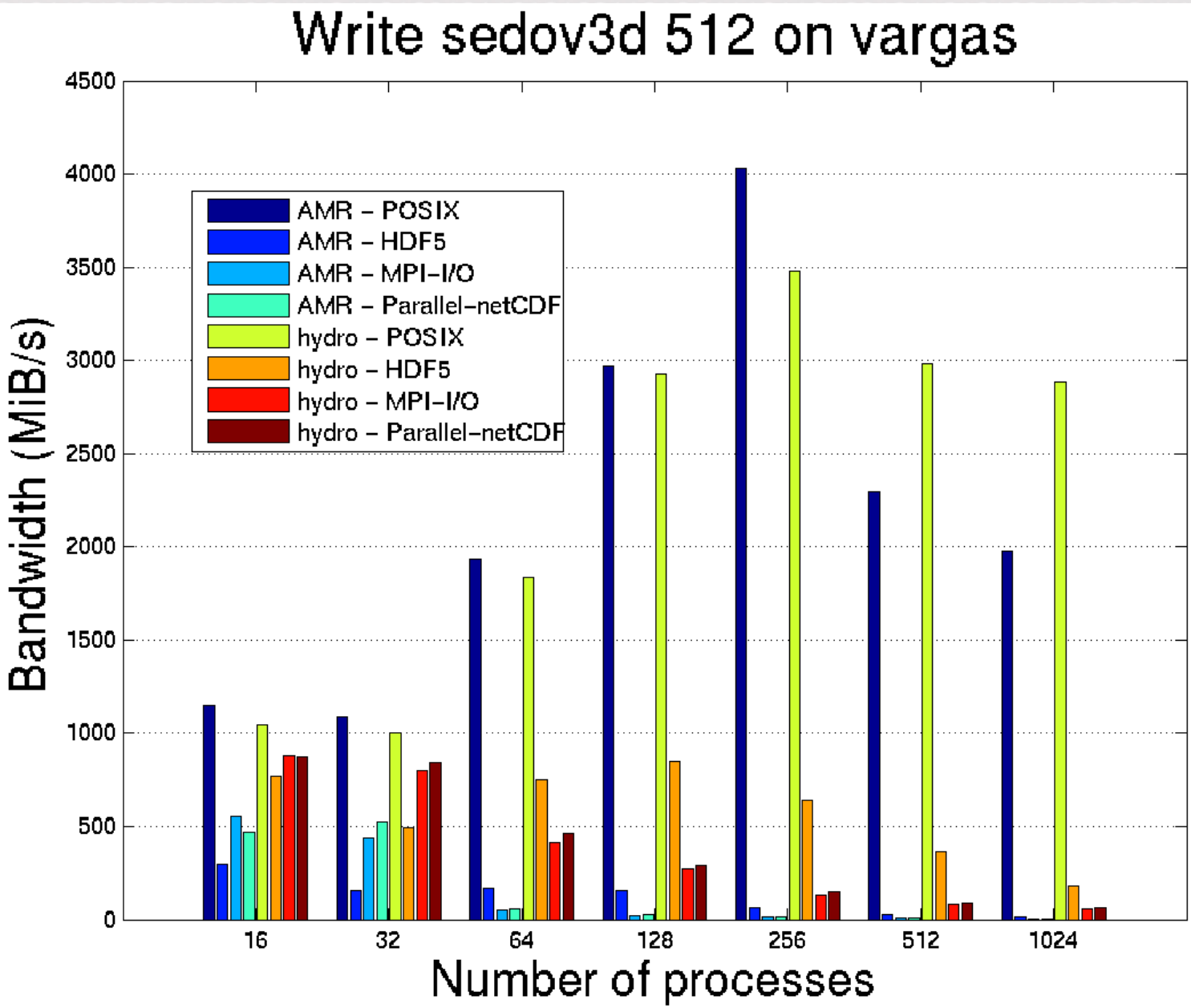


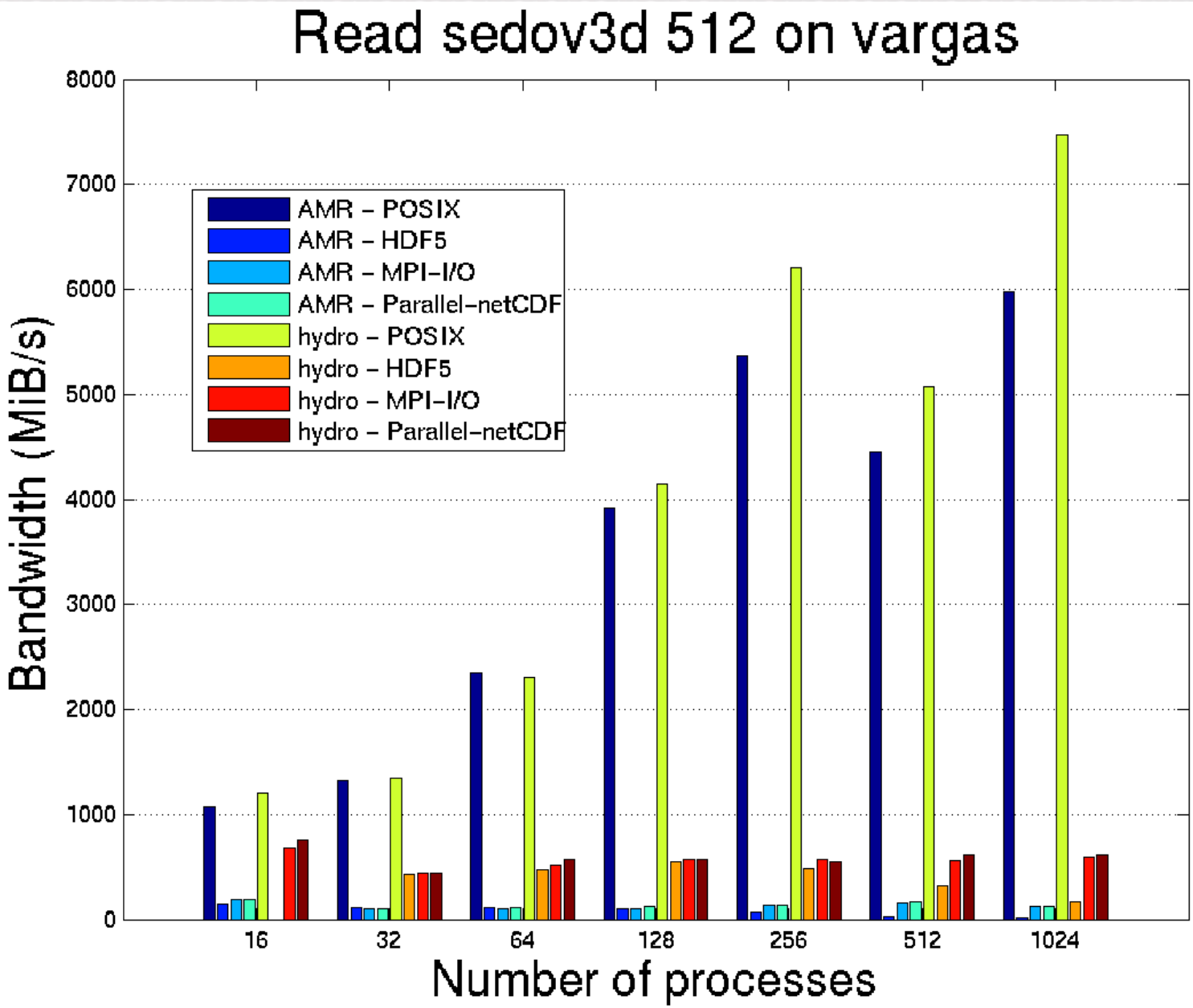

Read sedov3d 512 on Babel
Bandwidth (MiB/s)
4500
4000
3500
3000
2500
2000
1500
1000
500
0
AMR – POSIX
AMR – HDF5
AMR – MPI–I/O
AMR – Parallel–netCDF
hydro – POSIX
hydro – HDF5
hydro – MPI–I/O
hydro – Parallel–netCDF
64
128
256
512
1024
2048
4096
Number of processes


Write sedov3d 1024 on Babel
Bandwidth (MiB/s)
3000
2500
2000
1500
1000
500
0
AMR – POSIX
AMR – HDF5
AMR – MPI–I/O
AMR – Parallel–netCDF
hydro – POSIX
hydro – HDF5
hydro – MPI–I/O
hydro – Parallel–netCDF
512
1024
2048
4096
8192
Number of processes

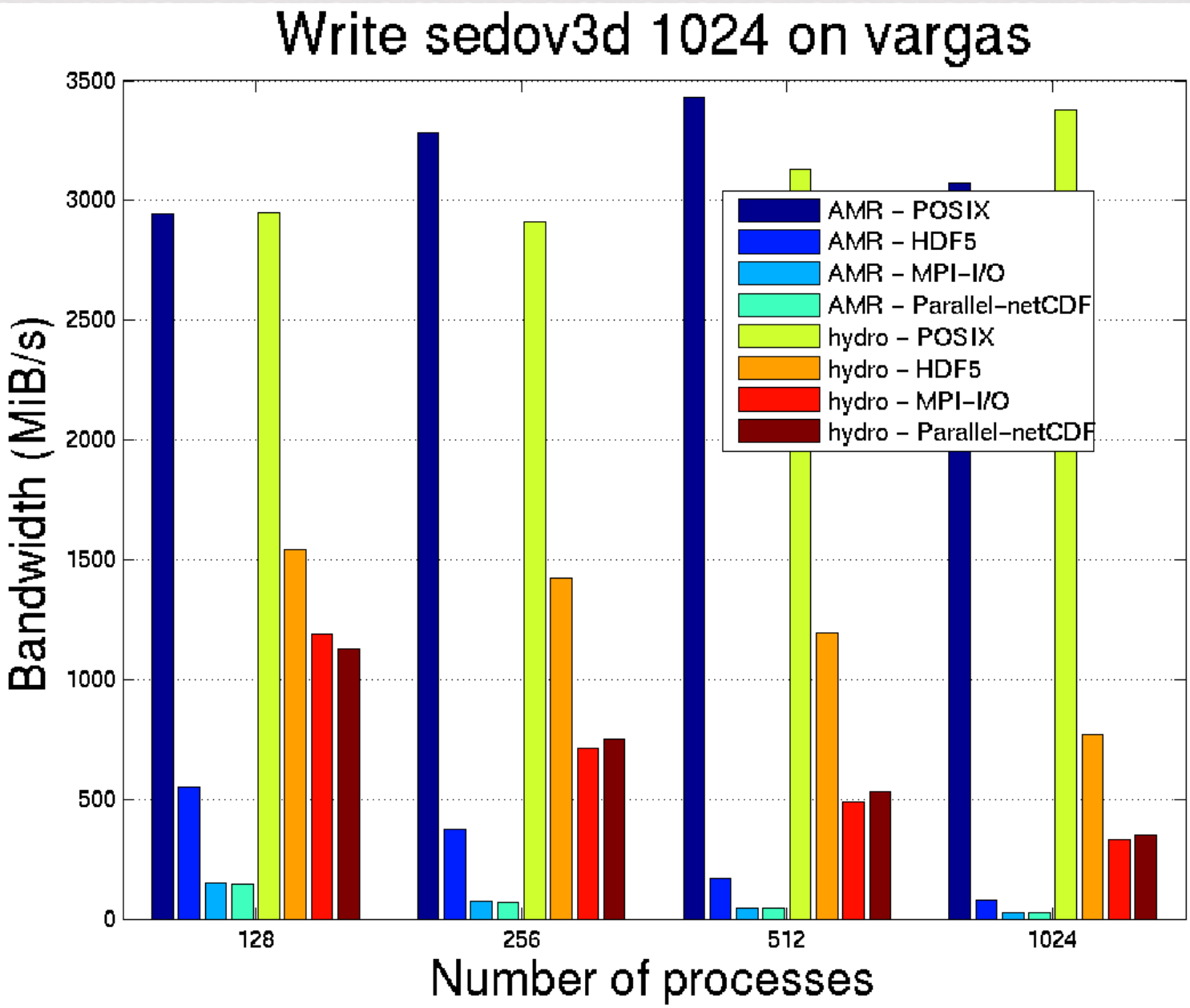

Read sedov3d 1024 on Babel
Bandwidth (MiB/s)
8000
7000
6000
5000
4000
3000
2000
1000
0
AMR – POSIX
AMR – HDF5
AMR – MPI–I/O
AMR – Parallel–netCDF
hydro – POSIX
hydro – HDF5
hydro – MPI–I/O
hydro – Parallel–netCDF
512
1024
2048
4096
8192
Number of processes

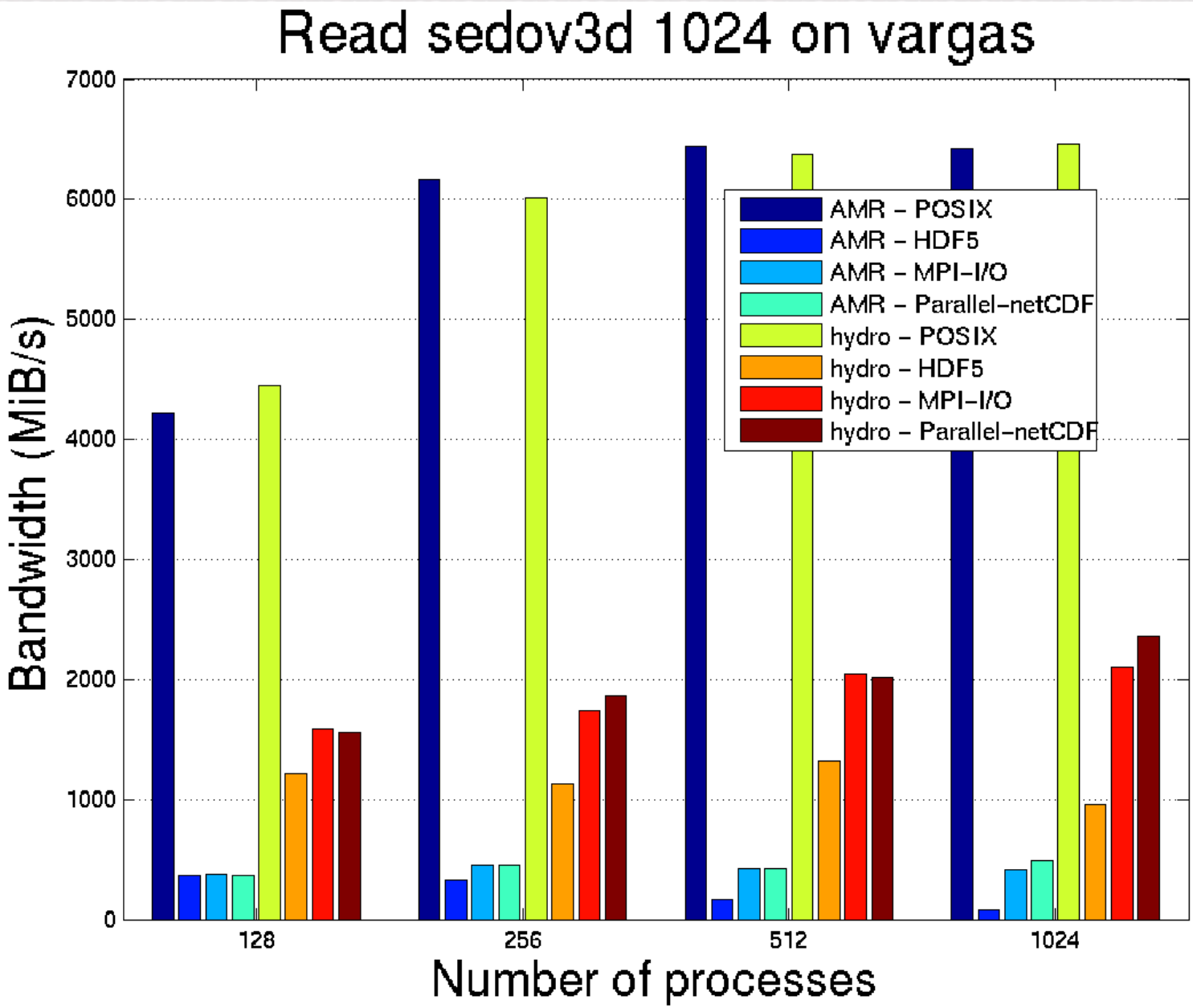
Ecriture Sedov3d 512^3 sur Power 6



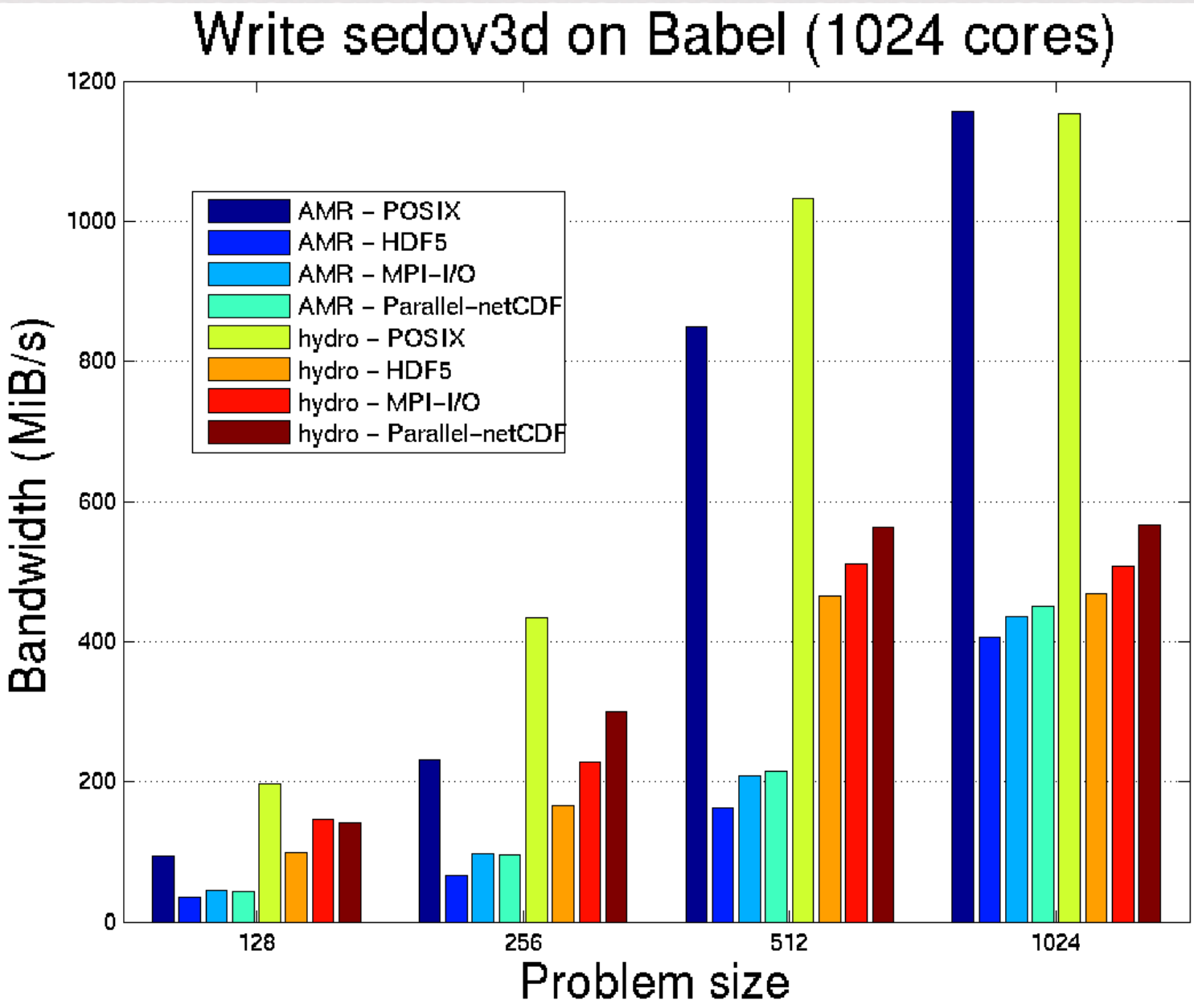

Read sedov3d 512 on vargas
Bandwidth (MiB/s)
Number of processes
AMR – POSIX
AMR – HDF5
AMR – MPI–I/O
AMR – Parallel–netCDF
hydro – POSIX
hydro – HDF5
hydro – MPI–I/O
hydro – Parallel–netCDF
0
1000
2000
3000
4000
5000
6000
7000
8000
16
32
64
128
256
512
1024

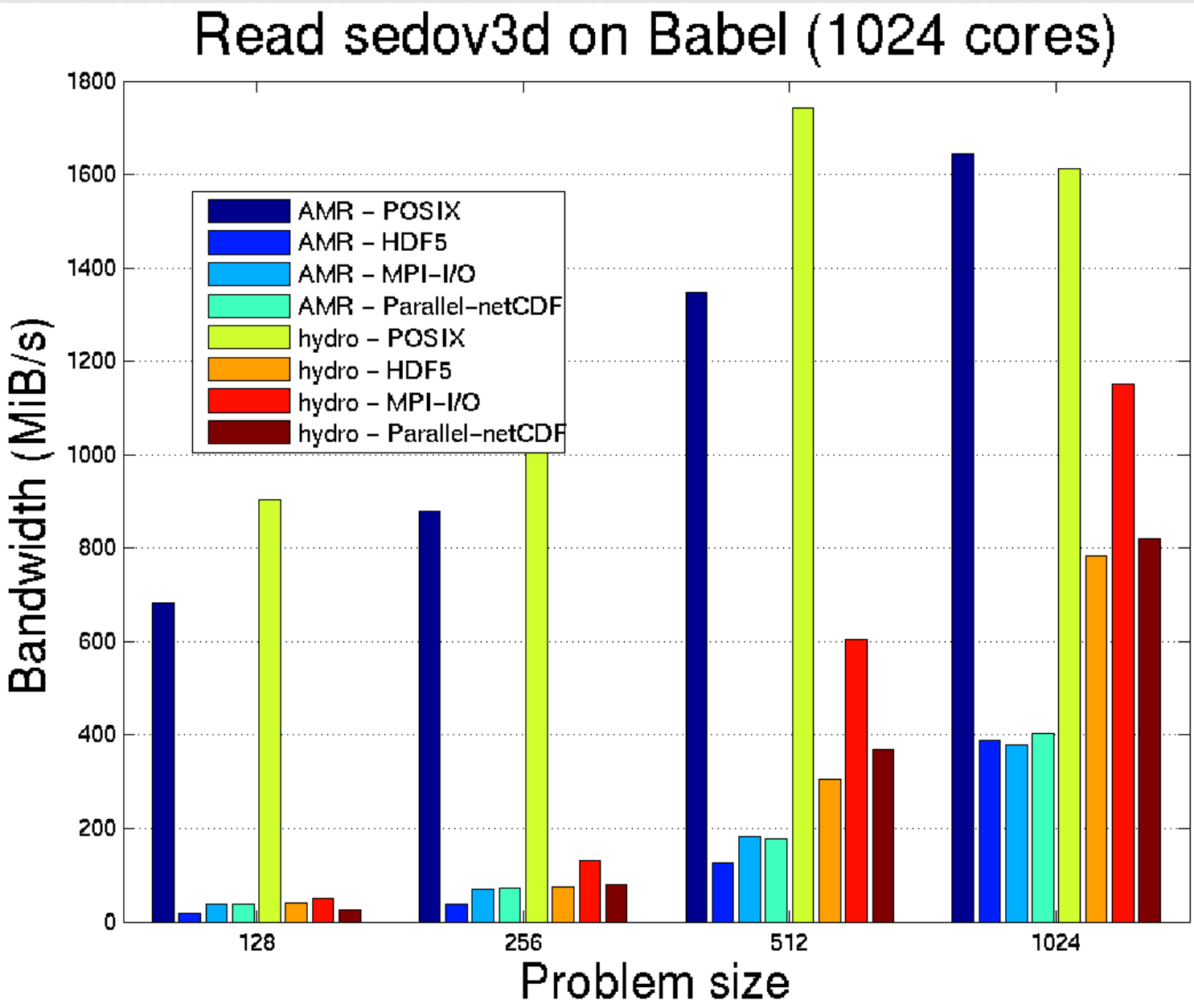
Ecriture Sedov3d 1024^3 sur Power 6



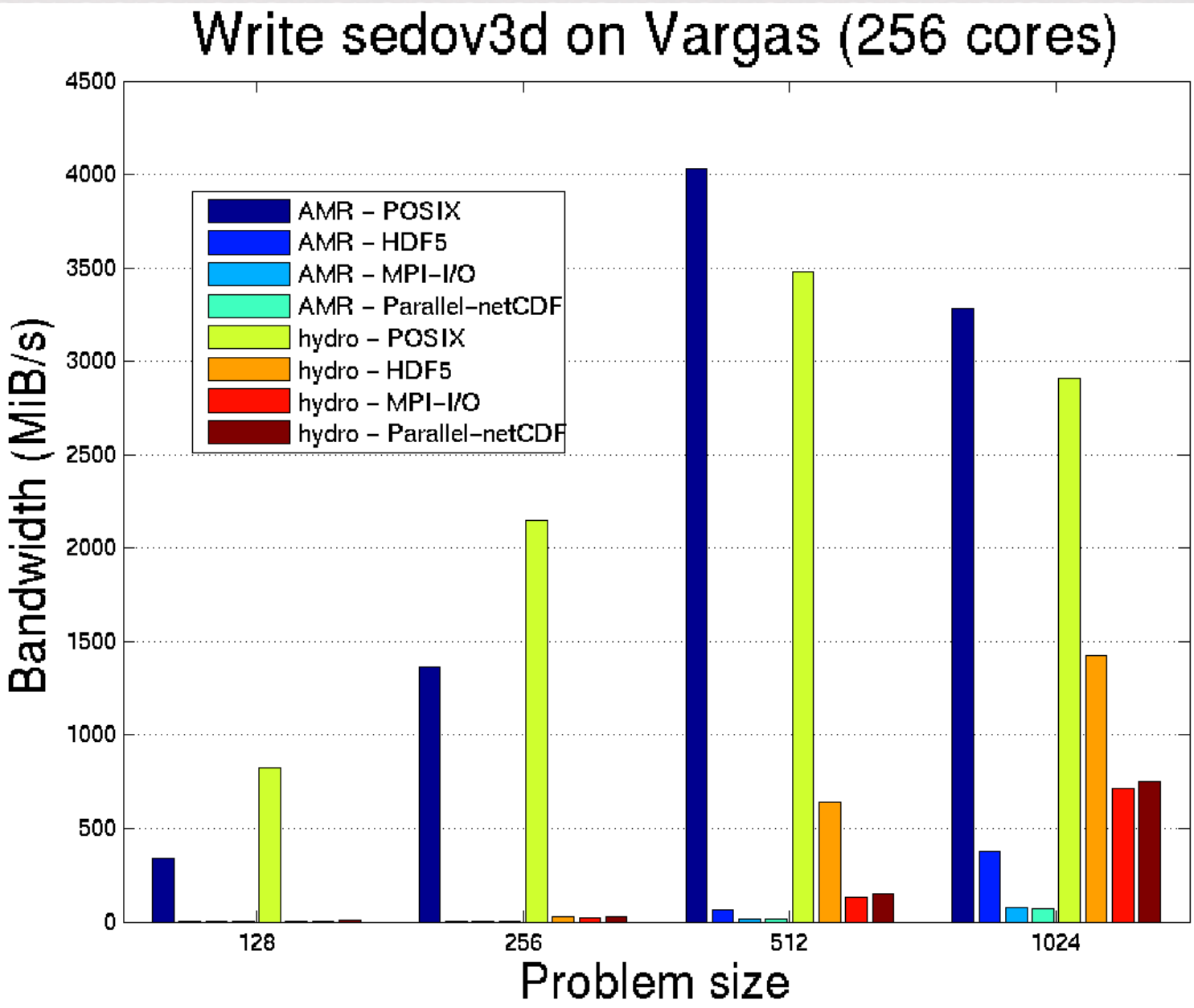

Read sedov3d 1024 on vargas
Bandwidth (MiB/s)
7000
6000
5000
4000
3000
2000
1000
0
128
256
512
1024
Number of processes
AMR – POSIX
AMR – HDF5
AMR – MPI–I/O
AMR – Parallel–netCDF
hydro – POSIX
hydro – HDF5
hydro – MPI–I/O
hydro – Parallel–netCDF

Ecriture Sedov3d sur 1024 cœurs Blue Gene/P

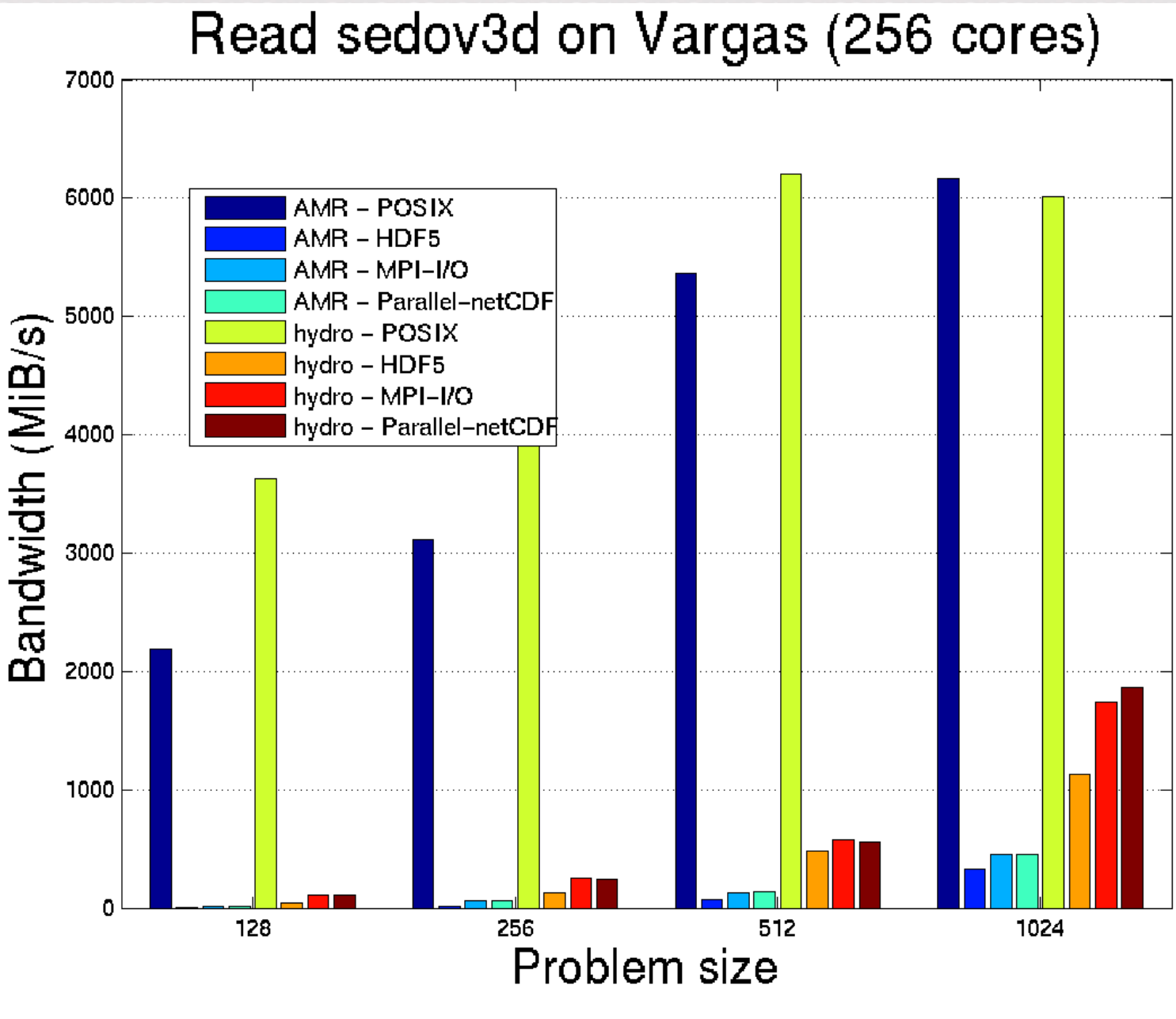



Read sedov3d on Babel (1024 cores)
Bandwidth (MiB/s)
Problem size
0
200
400
600
800
1000
1200
1400
1600
1800
128
256
512
1024
AMR – POSIX
AMR – HDF5
AMR – MPI–I/O
AMR – Parallel–netCDF
hydro – POSIX
hydro – HDF5
hydro – MPI–I/O
hydro – Parallel–netCDF

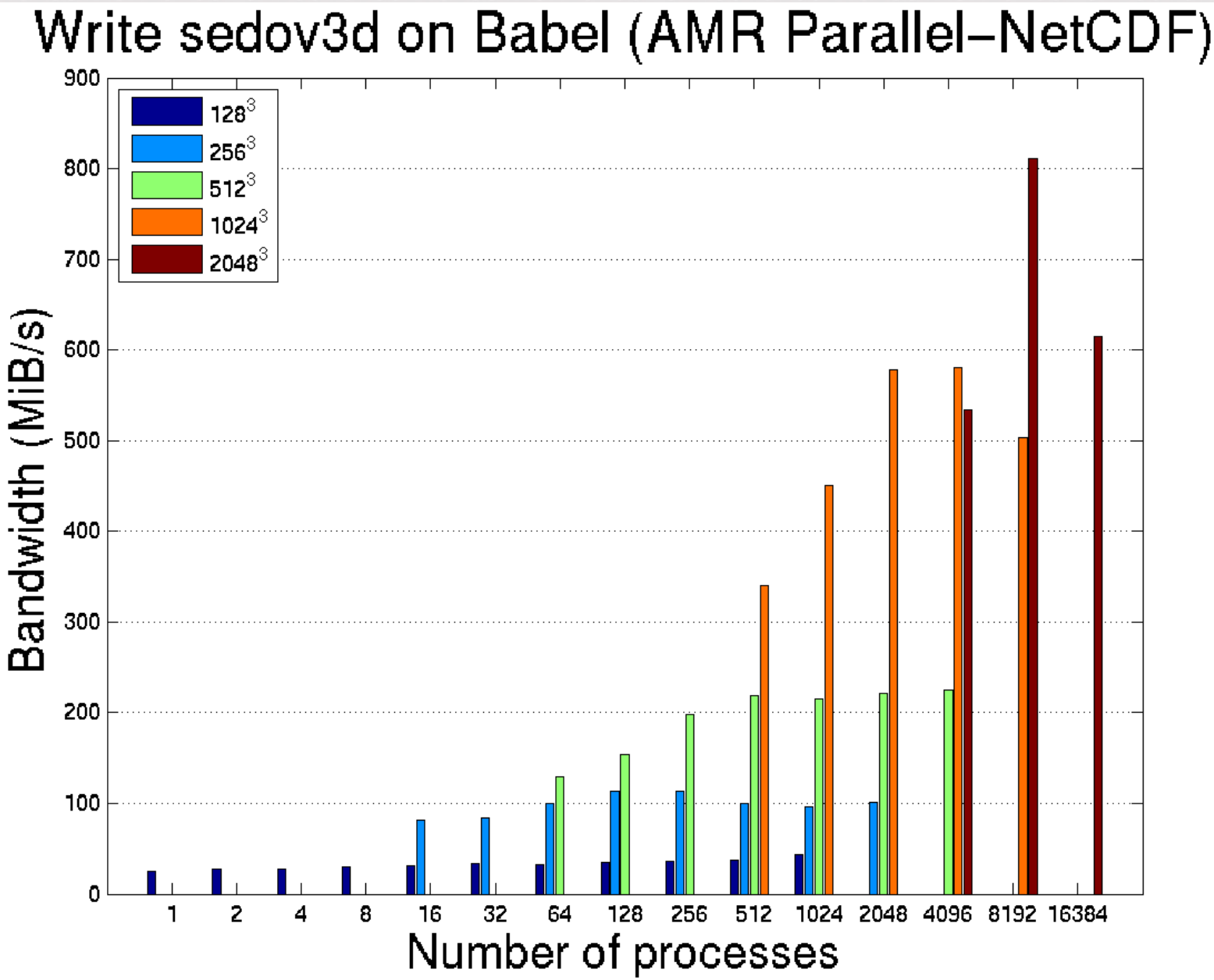
Ecriture Sedov3d sur 256 cœurs Power 6



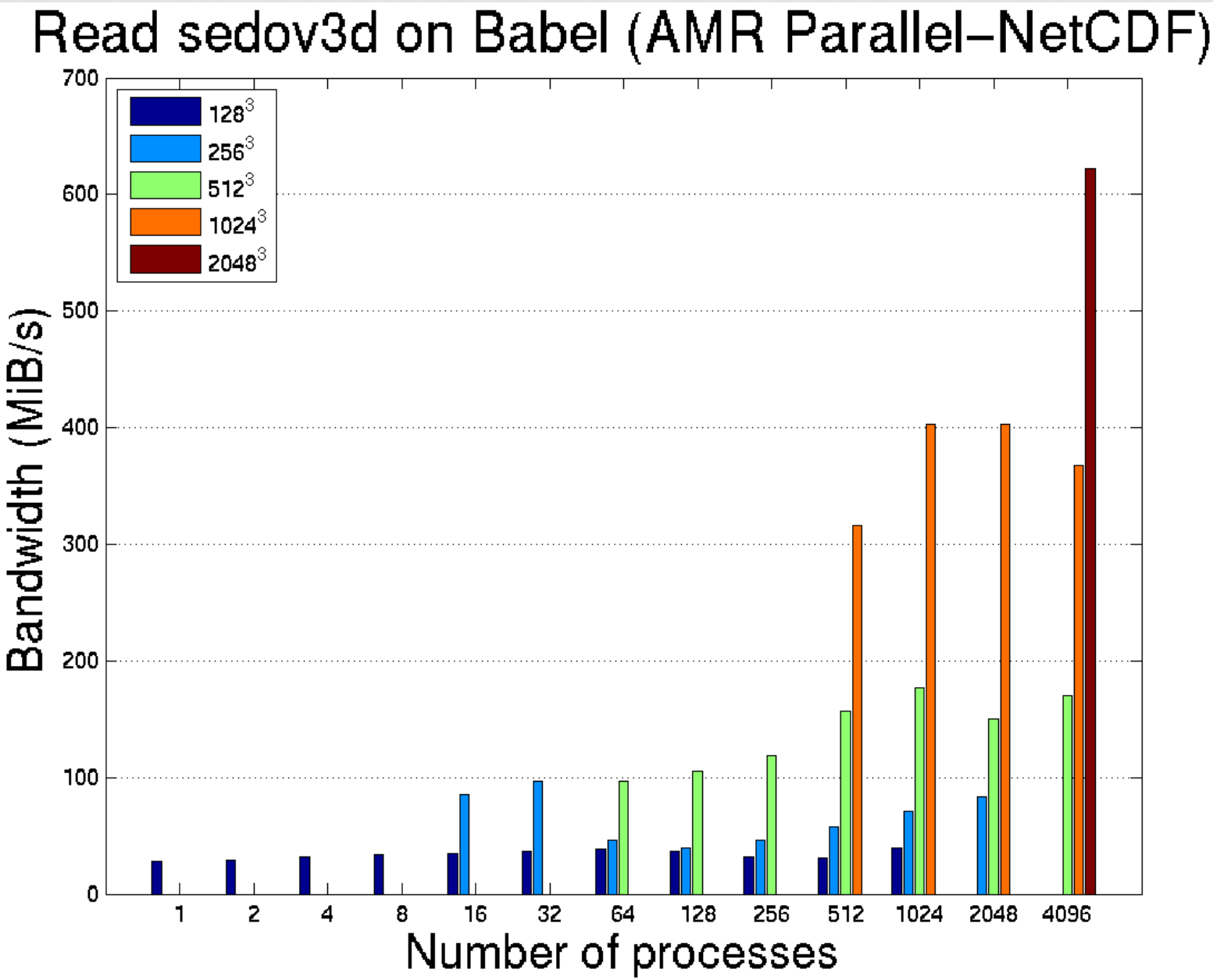
Lecture Sedov3d sur 256 cœurs Power 6



Ecriture Sedov3d sur Blue Gene/P du fichier AMR avec Parallel-NetCDF



Lecture Sedov3d sur Blue Gene/P du fichier AMR avec Parallel-NetCDF



Commentaires tests applicatifs RAMSES

- L'approche fichiers POSIX séparés est toujours nettement plus rapide. Avec un problème suffisamment gros et un nombre de processus élevé, le débit crête du système de fichiers est atteint en lecture.
- Pour les 3 autres approches, MPI-I/O est généralement la plus performante sauf en écriture sur Vargas où HDF5 l'emporte.
- Parallel-NetCDF est souvent proche de MPI-I/O (sauf en lecture sur Babel). Cela paraît logique car la surcouche sur MPI-I/O n'est pas très importante. Parallel-NetCDF se montre légèrement plus lent que MPI-I/O.
- HDF5 est l'approche la plus lente (sauf en écriture sur Vargas). Cela est probablement dû au fait que l'interface HDF5 est la plus complexe.
- A taille de problème fixe, les écarts entre les différents paradigmes tendent à s'accroître avec le nombre de processus.
- Les 3 approches MPI-I/O, HDF5 et Parallel-NetCDF sont malgré tout assez proches (surtout que les tests n'ont pas été effectués en mode dédié).
- Jusqu'à 4096 processus, tous les fichiers ont pu être écrits et relus sauf pour HDF5 sur le cas test 2048^3 (plantage à partir de 1024 processus sur Power 6 et 4096 sur Blue Gene/P).
- Au-delà de 4096 processus, seuls les fichiers séparés n'ont pas posé de problèmes. Seuls certains fichiers Parallel-NetCDF ont été écrits, mais n'ont jamais pu être relus (ces fichiers ne sont donc peut-être pas valides).

Pistes pour améliorer les performances

Pistes

- Processus spécialisés (approche suivie par d'autres groupes)
- Structurer les données différemment
- Accès par blocs de taille minimale ou fixe (pour éviter les I/O trop petits par rapport à la taille d'un bloc du système de fichiers)
- Appels MPI-I/O non-bloquants
- Evolution des implémentations MPI-IO et des bibliothèques HDF5 et Parallel-NetCDF
- Autres bibliothèques d'I/O (SIONlib...)

Conclusions

Avertissement

Les conclusions sur les différentes approches d'I/O parallèles (POSIX, MPI-I/O, HDF5 et Parallel-NetCDF) présentées ici ne sont valables que :

- pour les applications utilisées (IOR et RAMSES),
- pour les architectures cibles testées (systèmes IBM Power6 et Blue Gene/P de l'IDRIS),
- avec les versions des applications, systèmes et bibliothèques à la date de la rédaction de ce document.

Elles sont donc susceptibles de nettement différer dans d'autres conditions.

Conclusions et remarques finales

- Les besoins en entrées-sorties sont toujours croissants.
- L'approche fichiers séparés commence à montrer ses limites, même si elle reste la plus performante.
- Les paradigmes d'entrées-sorties parallèles tels que MPI-I/O, HDF5 et Parallel-NetCDF sont fonctionnels, mais souffrent encore de problèmes gênants.
- Ils permettent une gestion et une logistique simplifiée des données (fichiers en nombre plus limité, outils, portabilité garantie avec HDF5 et Parallel-NetCDF).
- Ces approches sont néanmoins complexes à mettre en œuvre (difficulté croissante de MPI-I/O à HDF5 en passant par Parallel-NetCDF).
- Les performances de celles-ci laissent à désirer. Atteindre un niveau raisonnable n'est pas trivial et nécessite un niveau d'expertise poussé.
- Des problèmes de robustesse et d'extensibilité existent sur un nombre de processus important (plusieurs milliers) avec ces paradigmes parallèles. Ce qui va à l'encontre de l'objectif de ces bibliothèques. Notons néanmoins les évolutions récentes très encourageantes.
- Un travail important de toute la communauté HPC (développeurs des bibliothèques et des applications, constructeurs et utilisateurs) est cependant encore nécessaire avant d'avoir des solutions performantes, robustes et extensibles sur les architectures massivement parallèles qui se généralisent dans tous les centres de calcul.

Documentation et liens

Documentation et liens

Pour en savoir plus

- Le site web de l'IDRIS : http://www.idris.fr
- Le site d'HDF5 : http://www.hdfgroup.org/HDF5/
- Le site de Parallel-NetCDF : http://trac.mcs.anl.gov/projects/parallel-netcdf
- Les implémentations MPI *open source* : OpenMPI http://www.open-mpi.org et MPICH2 http://www.mcs.anl.gov/research/projects/mpich2/
- Le logiciel de benchmark IOR : http://sourceforge.net/projects/ior-sio/
- RAMSES : http://irfu.cea.fr/Projets/Site_ramses/RAMSES.html. Attention, la version avec I/O parallèles n'est pas disponible sur ce site. Contactez l'auteur de cette présentation pour plus d'informations.
- Une liste de benchmarks est proposée sur ce site : http://www.mcs.anl.gov/ thakur/pio-benchmarks.html
- Le site de NetCDF http://www.unidata.ucar.edu/software/netcdf/index.html dont la version 4 utilise le format HDF5
- La bibliothèque SIONlib : http://www.fz-juelich.de/jsc/sionlib/
- La bibliothèque ADIOS : http://adiosapi.org/index.php5 ?title=Main_Page