université PARIS-SACLAY

école normale supérieure paris-saclay

HPL High Performance Linpack Benchmark

Projet MPNA

Méthodes et Programmation Numérique Avancée

Réalisé par :

BENMALK Achraf

BEZINE Mohamed Karim

Date : 18 février 2026

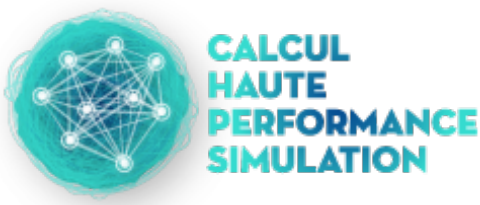

Table des matières

Table des figures

Liste des tableaux

1 Contexte

HPL (High Performance Linpack) est un benchmark fondamental en calcul haute performance qui évalue la capacité de calcul en virgule flottante d'un système. Il est largement utilisé pour mesurer l'efficacité des CPUs et des GPUs lors de la résolution de grands systèmes linéaires denses.

HPL ne mesure pas uniquement la performance brute en gigaflops ou teraflops. Il permet également d'évaluer la gestion du parallélisme, les accès mémoire ainsi que le coût des communications. Il constitue ainsi un outil essentiel pour l'optimisation et la comparaison des systèmes HPC.

2 Caractéristiques du Benchmark HPL

HPL mesure la vitesse de résolution d'un système linéaire dense de la forme :

$$Ax = b$$

où A est une matrice dense de grande taille.

Il s'agit du benchmark utilisé pour classer les supercalculateurs dans la liste TOP500.

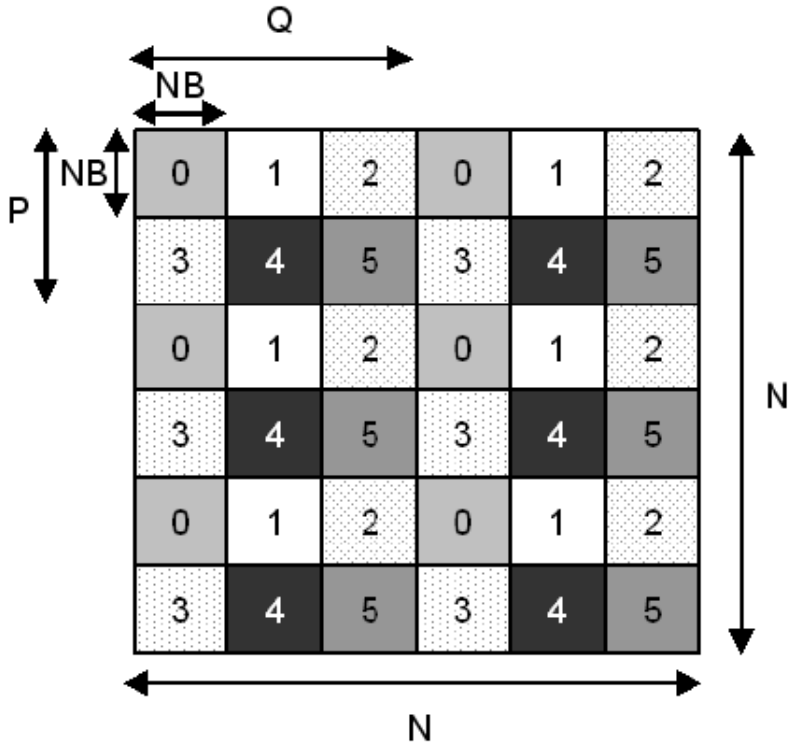

FIGURE 1 – Paramètres principaux du benchmark HPL

Les paramètres principaux sont :

- **N** : taille globale de la matrice.

- **NB** : taille des blocs.
- **P** × **Q** : grille de processus MPI.

La matrice est distribuée selon une stratégie **block-cyclic** afin d'équilibrer la charge entre les processus et de minimiser les communications.

3 Fichier d'entrée HPL.dat

Le benchmark nécessite un fichier de configuration nommé `HPL.dat`. Ce fichier définit :

- Les tailles de matrices testées
- Les tailles de blocs
- La grille de processus MPI
- Les paramètres algorithmiques
- La tolérance de vérification du résidu

```
1  HPLinpack benchmark input file
2  Innovative Computing Laboratory, University of Tennessee and Frankfurt Institute for Advanced Studies
3  HPL.out      output file name (if any)
4  6            device out (6=stdout,7=stderr,file)
5  5        # of problems sizes (N)
6  20000 40000 60000 80000 100000       Ns
7  1            # of NBs
8  576            NBs
9  1            PMAP process mapping (0=Row-,1=Column-major)
10 1            # of process grids (P x Q)
11 2            Ps
12 1            Qs
13 0.1           threshold
14 1            # of panel fact
15 1            PFACTs (0=left, 1=Crout, 2=Right)
16 1            # of recursive stopping criterium
17 64           NBMINs (>= 1)
18 1            # of panels in recursion
19 2            NDIVs
20 1            # of recursive panel fact.
21 0            RFACTs (0=left, 1=Crout, 2=Right)
22 1            # of broadcast
23 6            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM,6=MPI)
24 1            # of lookahead options
25 3            LOOKAHEADs (enable = 1)
26 8            memory alignment in double (> 0)
27 100            Seed for the matrix generation
```

FIGURE 2 – Exemple de fichier HPL.dat

HPL explore différentes combinaisons de paramètres afin d'identifier la configuration optimale.

4 Plateforme d'exécution

Les expériences ont été réalisées sur un **cluster HPC** équipé de nœuds GPU. Les partitions utilisées sont :

- **GPU A100** : partition `gpu`, NVIDIA A100 80 Go HBM2e (architecture Ampere)
- **GPU H100** : partition `gpu_h100`, NVIDIA H100 PCIe 80 Go HBM3 (architecture Hopper)

Le benchmark est exécuté via le conteneur **NVIDIA HPC-Benchmarks 23.10**, déployé avec Singularity. Ce conteneur fournit une version optimisée de HPL qui exploite les **Tensor Cores FP64** des GPUs pour les opérations DGEMM, ainsi que les bibliothèques cuBLAS et NCCL pour les communications inter-GPU.

La configuration HPL utilisée pour les tests GPU est : NB = 576, P × Q = 2 × 1 (2 processus MPI, 1 par GPU), BCAST = 6 (broadcast MPI). La taille de bloc NB = 576 est nettement supérieure aux valeurs CPU typiques (128–256) car les GPUs nécessitent de grands blocs pour alimenter leurs milliers de cœurs en données et masquer la latence mémoire.

L'ordonnanceur de tâches **Slurm** est utilisé pour l'allocation des ressources GPU.

5 Étapes d'exécution

Les étapes principales sont :

1. Création du fichier `HPL.dat`
2. Lancement du conteneur Singularity
3. Allocation des ressources GPU via Slurm
4. Exécution du benchmark avec MPI

```
$ cd /path/to/working/directory
$ mkdir data
$ cd data/
$ touch HPL.dat
$ vim HPL.dat
```

FIGURE 3 – Création du fichier HPL.dat

```
$ singularity exec --bind $(pwd):/mnt ./hpc-benchmarks_23.10.sif cat /mnt/HPL.dat
```

FIGURE 4 – Binding du dossier dans le conteneur

```
$ echo export SINGULARITY_BINDPATH=/home/karim.bezine-ext/lustre/sw_stack-
373lcd9r8io/users/karim.bezine-ext/src/data:/mnt >> ~/.bashrc
$ source ~/.bashrc
```

FIGURE 5 – Configuration persistante du bind

```
$ singularity run --nv hpc-benchmarks_23.10.sif
$ cd /workspace/hpl-linux-x86_64/
$ mpirun -np 1 hpl.sh --dat /mnt/HPL.dat --gpu-affinity 2 --no-multinode --cuda-compat
```

FIGURE 6 – Exécution du benchmark

Commande principale (pour 1 GPU, 1 processus MPI) :

```
mpirun -np 1 ./hpl.sh --dat /mnt/HPL.dat
```

Pour les configurations multi-GPU, le nombre de processus MPI est ajusté (`-np 2` pour 2 GPUs).

6 Partition A100

6.1 1 GPU, 1 Processus MPI

```
salloc -t 4:00:00 -n 1 --ntasks=1 -p gpu --gres=gpu:1
```

N	Temps (s)	Gflops
20000	0.51	9731
40000	1.63	15890
60000	8.36	17150
80000	19.01	17600
100000	36.94	17860

TABLE 1 – Résultats sur 1 GPU A100

Le pic théorique FP64 de l'A100 est calculé à partir du nombre de cœurs CUDA, du facteur FMA (*Fused Multiply-Add*, chaque opération FMA compte pour 2 opérations flottantes) et de la fréquence boost :

$$\text{Peak}_{A100} = \underbrace{6\,912}_{\substack{\text{cœurs}\\\text{CUDA}}} \times \underbrace{2}_{\text{FMA}} \times \underbrace{1{,}41\ \text{GHz}}_{\text{freq. boost}} \approx 19{,}5\ \text{TFLOPS}$$

Efficacité maximale atteinte (à $N = 100\,000$) :

$$\eta_{A100} = \frac{17\,860}{19\,487} \times 100 \approx 91{,}7\%$$

Cette efficacité élevée confirme que HPL est un benchmark *compute-bound* : la majorité du temps est consacrée aux opérations DGEMM qui exploitent pleinement les capacités de calcul du GPU. Tous les tests ont passé la vérification du résidu ($\|Ax-b\|_\infty/(\varepsilon\cdot(\|A\|_\infty\cdot\|x\|_\infty+\|b\|_\infty)\cdot N) < 16{,}0$), confirmant la validité numérique des résultats.

6.2 2 GPU, 2 Processus MPI, 1 Processus MPI/GPU

```
salloc -t 4:00:00 -n 1 --gres=gpu:2
```

N	Temps (s)	Gflops
20000	0.55	9106
40000	1.66	25230
60000	4.56	31430
80000	9.98	33560
100000	19.00	34720

TABLE 2 – Résultats sur 2 GPUs A100

Speedup à $N = 100\,000$:

$$S_{A100} = \frac{34\,720}{17\,860} \approx 1{,}95$$

Efficacité parallèle :

$$E_{A100} = \frac{S_{A100}}{2} \times 100 = \frac{1{,}95}{2} \times 100 \approx 97\%$$

Remarque : À $N = 20\,000$, les 2 GPUs (9 106 GFLOPS) sont *plus lents* que le GPU unique (9 731 GFLOPS). Ce phénomène est analysé en section 8.

7 Partition H100

7.1 1 GPU, 1 Processus MPI

```
salloc -t 4:00:00 -n 1 -p gpu_h100 --gres=gpu:1
```

N	Temps (s)	Gflops
20000	0.31	16130
40000	1.17	35760
60000	3.43	41760
80000	7.57	44130
100000	11.43	45110

TABLE 3 – Résultats sur 1 GPU H100

Le pic théorique FP64 du H100 est calculé de manière analogue à l'A100 :

$$\text{Peak}_{H100} = \underbrace{16\,896}_{\substack{\text{cœurs}\\ \text{CUDA}}} \times \underbrace{2}_{\text{FMA}} \times \underbrace{1{,}6\ \text{GHz}}_{\text{freq. boost}} \approx 54\ \text{TFLOPS}$$

Efficacité maximale atteinte (à $N = 100\,000$) :

$$\eta_{H100} = \frac{45\,110}{54\,067} \times 100 \approx 83{,}4\%$$

L'efficacité du H100 est inférieure à celle de l'A100 (83,4% vs 91,7%). Le H100 possède 2,4× plus de cœurs CUDA que l'A100 (16 896 vs 6 912), ce qui rend plus difficile la saturation complète de toutes les unités de calcul pour une même taille de problème. Pour atteindre une efficacité comparable, des valeurs de N encore plus grandes seraient nécessaires.

7.2 2 GPU, 2 Processus MPI, 1 Processus MPI/GPU

```
salloc -t 4:00:00 -n 1 --ntasks=2 -p gpu_h100 --gres=gpu:2
```

N	Temps (s)	Gflops
20000	0.40	11510
40000	1.01	41460
60000	1.11	64700
80000	4.36	76730
100000	7.95	81970

TABLE 4 – Résultats sur 2 GPUs H100

Speedup à $N = 100\,000$:

$$S_{H100} = \frac{81\,970}{45\,110} \approx 1{,}82$$

Efficacité parallèle :

$$E_{H100} = \frac{S_{H100}}{2} \times 100 = \frac{1{,}82}{2} \times 100 \approx 91\%$$

Remarque : Le passage multi-GPU est moins efficace sur H100 (91%) que sur A100 (97%). À $N = 20\,000$, les 2 GPUs H100 (11 510 GFLOPS) sont même 28,6% *plus lents* que le GPU unique (16 130 GFLOPS). Cette dégradation est analysée en section 8. Tous les tests ont passé la vérification du résidu.

8 Analyse des résultats

8.1 Évolution des performances avec la taille du problème

Sur les deux architectures, les performances (GFLOPS) augmentent avec N avant de converger vers un plateau. Ce comportement s'explique par le rapport entre calcul et communication :

- Le volume de calcul de la factorisation LU croît en $\mathcal{O}(N^3)$.
- Le volume de communication croît en $\mathcal{O}(N^2)$.

Ainsi, pour des grandes valeurs de N, le calcul domine largement et le GPU peut maintenir un taux d'utilisation élevé de ses unités de calcul (principalement via les opérations DGEMM). Pour les petites valeurs de N, le *overhead* de communication et de synchronisation représente une fraction significative du temps total, ce qui réduit les GFLOPS mesurés.

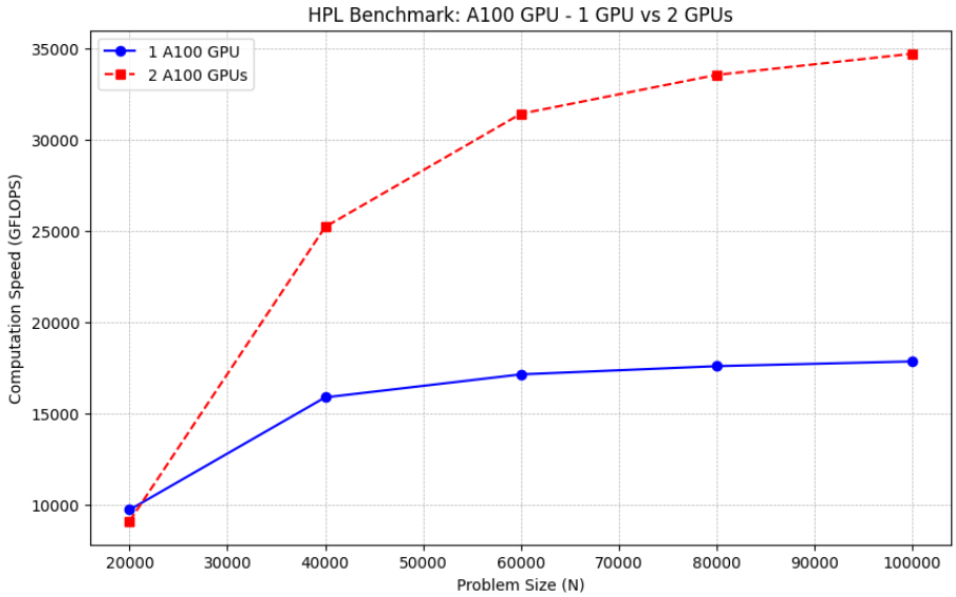

FIGURE 7 – Performances A100 : 1 GPU vs 2 GPUs

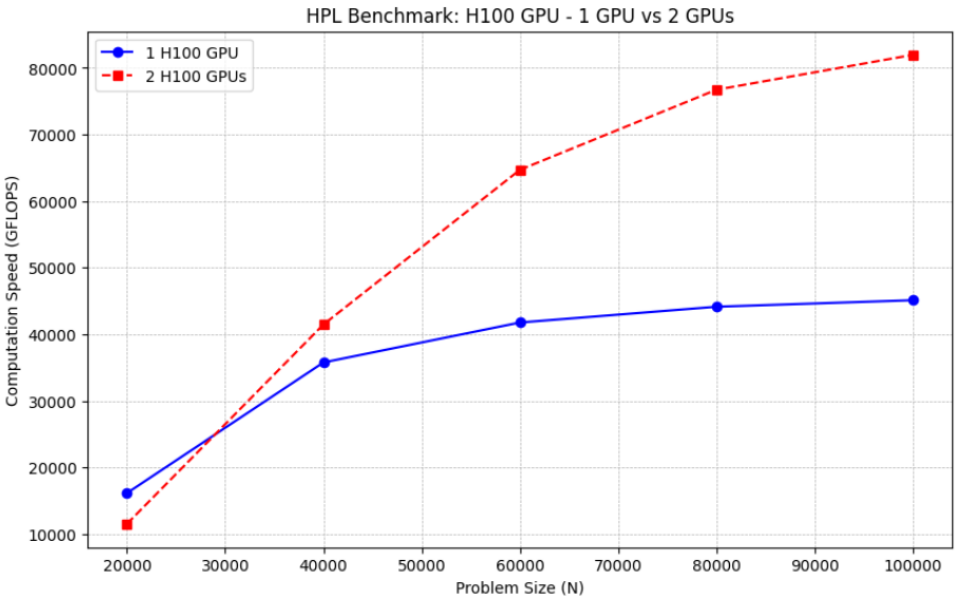

FIGURE 8 – Performances H100 : 1 GPU vs 2 GPUs

8.2 Passage à l'échelle multi-GPU

Le tableau 5 résume le passage à l'échelle pour $N = 100\,000$:

GPU	1 GPU (GFLOPS)	2 GPUs (GFLOPS)	Speedup	Efficacité parallèle
A100	17 860	34 720	1,95×	97%
H100	45 110	81 970	1,82×	91%

TABLE 5 – Passage à l'échelle multi-GPU à $N = 100\,000$

L'A100 affiche une meilleure efficacité parallèle (97%) que le H100 (91%). Cette différence s'explique par le fait que le H100, étant plus rapide en calcul pur, termine la phase de calcul plus rapidement, ce qui rend la latence de communication inter-GPU (via NVLink) proportionnellement plus coûteuse. Le rapport calcul/communication est donc moins favorable sur H100 pour une même taille de problème.

8.3 Anomalie à $N = 20\,000$: 2 GPUs plus lents qu'un seul

Un résultat remarquable concerne la dégradation de performance à $N = 20\,000$ en configuration multi-GPU :

GPU	1 GPU (GFLOPS)	2 GPUs (GFLOPS)	Dégradation
A100	9 731	9 106	−6,4%
H100	16 130	11 510	−28,6%

TABLE 6 – Dégradation de performance à $N = 20\,000$ en multi-GPU

Avec un problème de petite taille ($N = 20\,000$), chaque GPU reçoit une portion de matrice trop réduite pour saturer ses milliers de cœurs CUDA. Le coût de synchronisation et d'échange de données entre les deux GPUs dépasse alors le bénéfice du parallélisme supplémentaire. Ce phénomène est plus prononcé sur H100 (−28,6%) car ce GPU possède davantage de cœurs CUDA à alimenter en données (16 896 vs 6 912).

Il existe donc un seuil de taille de problème en dessous duquel l'ajout de GPUs est contre-productif. Nos données indiquent que ce seuil se situe entre $N = 20\,000$ (dégradation observée) et $N = 40\,000$ (gain observé) pour les deux architectures.

8.4 Comparaison architecturale A100 vs H100

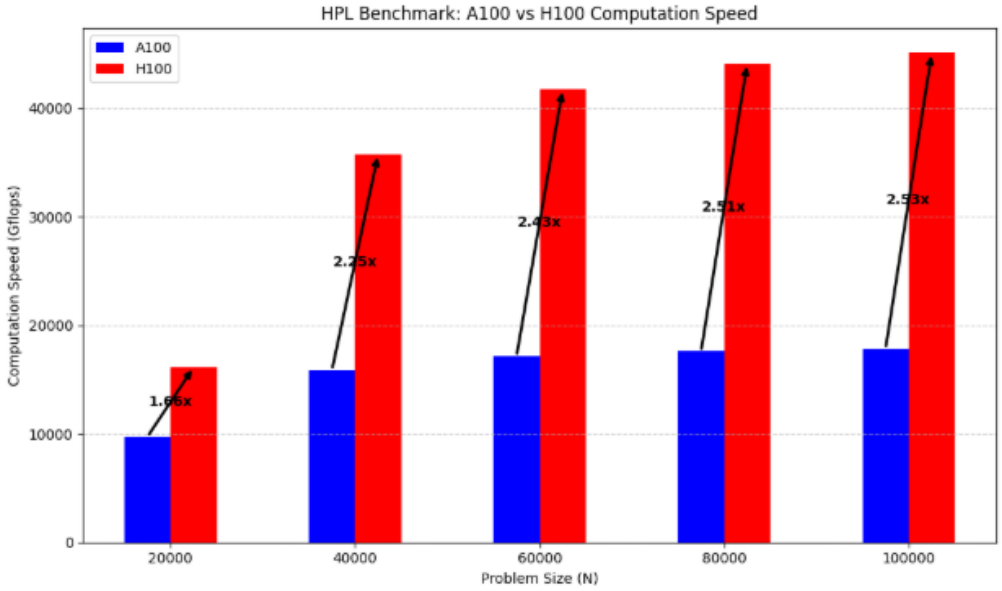

FIGURE 9 – Comparaison des performances A100 vs H100 (1 GPU)

Le tableau 7 présente le facteur d'accélération du H100 par rapport à l'A100 pour chaque taille de problème :

N	A100 (GFLOPS)	H100 (GFLOPS)	Ratio H100/A100
20 000	9 731	16 130	1,66×
40 000	15 890	35 760	2,25×
60 000	17 150	41 760	2,43×
80 000	17 600	44 130	2,51×
100 000	17 860	45 110	2,53×

TABLE 7 – Facteur d'accélération H100 par rapport à A100

Le ratio converge vers $\approx 2{,}5\times$ pour les grandes tailles de problème. Cela est cohérent avec le rapport des pics théoriques FP64 :

$$\frac{\text{Peak}_{H100}}{\text{Peak}_{A100}} = \frac{54\,067}{19\,487} \approx 2{,}77$$

Le facteur mesuré (2,53×) est proche du facteur théorique (2,77×), avec un écart de $\approx 9\%$. Cet écart s'explique par la différence d'efficacité entre les deux architectures : l'A100 atteint 91,7% de son pic tandis que le H100 n'atteint que 83,4%, car il est plus difficile de saturer un GPU disposant de davantage de cœurs.

À $N = 20\,000$, le ratio n'est que de 1,66× car les deux GPUs sont sous-utilisés : ni l'un ni l'autre ne peut exploiter pleinement ses unités de calcul avec un problème aussi petit.

8.5 Synthèse de l'efficacité

Configuration	Pic théorique TC (TFLOPS)	Meilleur résultat (GFLOPS)	Efficacité
A100 (1 GPU)	19,5	17 860	91,7%
H100 (1 GPU)	54,0	45 110	83,4%
A100 (2 GPUs)	39,0	34 720	89,0%
H100 (2 GPUs)	108,0	81 970	75,8%

TABLE 8 – Synthèse de l'efficacité HPL sur les différentes configurations

On observe que l'efficacité décroît lorsqu'on passe à des configurations plus puissantes. Plus le pic théorique est élevé, plus il est difficile de l'exploiter pleinement. L'A100 atteint une excellente efficacité de 91,7% grâce à un ratio cœurs/performance mieux maîtrisé, tandis que le H100 2 GPUs descend à 75,8%. Cela témoigne néanmoins de la nature *compute-bound* du benchmark HPL et de la qualité de l'implémentation GPU fournie par NVIDIA dans le conteneur HPC-Benchmarks.

8.6 Conclusion

Ce travail a permis d'évaluer les performances du benchmark HPL sur deux architectures GPU NVIDIA — l'A100 (Ampere) et le H100 (Hopper) — au sein d'un cluster HPC.

Les principaux enseignements sont :

1. **HPL est un benchmark compute-bound :** Les efficacités mesurées (83–92%) confirment que la performance est principalement limitée par la puissance de calcul et non par la bande passante mémoire ou les communications.
2. **La taille du problème est déterminante :** Les performances augmentent significativement avec N en raison du rapport $\mathcal{O}(N^3)/\mathcal{O}(N^2)$ entre calcul et communication. Un N trop petit conduit à une sous-utilisation des ressources.
3. **Le multi-GPU a un seuil de rentabilité :** En dessous d'un certain N (entre 20 000 et 40 000 dans nos tests), l'ajout d'un second GPU dégrade les performances au lieu de les améliorer.
4. **Le H100 offre un gain de $\approx 2{,}5\times$ sur l'A100 :** Ce ratio mesuré (2,53×) est cohérent avec le rapport des pics théoriques FP64 (2,77×), l'écart de $\approx 9\%$ s'expliquant par la différence d'efficacité entre les deux architectures.
5. **L'efficacité parallèle diminue avec la puissance :** L'A100 affiche une meilleure efficacité parallèle multi-GPU (97%) que le H100 (91%), car la latence de communication devient proportionnellement plus coûteuse sur un GPU plus rapide.

Ces résultats illustrent les compromis fondamentaux du calcul haute performance : puissance brute, passage à l'échelle, et dimensionnement du problème sont étroitement liés. L'optimisation des paramètres HPL (taille de matrice, taille de bloc, grille de processus) reste essentielle pour exploiter au mieux les architectures modernes.

9 Mini-HPL parallèle avec MPI

9.1 Rappels mathématiques

Système linéaire et élimination de Gauss

On cherche $x \in \mathbb{R}^N$ tel que $Ax = b$, avec $A \in \mathbb{R}^{N\times N}$ inversible. On travaille sur la matrice augmentée $[A \mid b]$.

À l'étape k, on annule les coefficients de la colonne k pour toutes les lignes $i > k$:

$$\ell_{ik} = \frac{a_{ik}}{a_{kk}}, \qquad \text{ligne}_i \leftarrow \text{ligne}_i - \ell_{ik} \cdot \text{ligne}_k \tag{1}$$

Complexité et GFLOPS

Le nombre d'opérations flottantes est $W \approx \frac{2}{3}N^3$. La performance est donc :

$$\text{GFLOPS}_{\text{approx}} = \frac{\frac{2}{3}N^3}{t_{\text{élim}} \times 10^9}.$$

Pivotage partiel

Sans pivotage, l'algorithme est instable si a_{kk} est petit. Le pivotage partiel permute la ligne k avec la ligne p telle que :

$$p = \arg\max_{i\geq k} |a_{ik}|.$$

9.2 Exemple à la main — Gauss avec pivotage ($N = 3$)

Exemple à la main

Soit le système $Ax = b$ avec :

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 3 & 8 & 1 \\ 0 & 4 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 12 \\ 2 \end{pmatrix}.$$

Étape $k = 0$: max en colonne $0 = |a_{10}| = 3$. On échange $L_0 \leftrightarrow L_1$:

$$\left(\begin{array}{ccc|c} 3 & 8 & 1 & 12 \\ 1 & 2 & 1 & 2 \\ 0 & 4 & 1 & 2 \end{array}\right).$$

$\ell_{10} = \frac{1}{3}$, $\ell_{20} = 0$. $L_1 \leftarrow L_1 - \frac{1}{3}L_0$:

$$\left(\begin{array}{ccc|c} 3 & 8 & 1 & 12 \\ 0 & -\frac{2}{3} & \frac{2}{3} & -2 \\ 0 & 4 & 1 & 2 \end{array}\right).$$

Étape $k = 1$: max en colonne 1 (lignes ≥ 1) $= |a_{21}| = 4$. Échange $L_1 \leftrightarrow L_2$:

$$\left(\begin{array}{ccc|c} 3 & 8 & 1 & 12 \\ 0 & 4 & 1 & 2 \\ 0 & -\frac{2}{3} & \frac{2}{3} & -2 \end{array}\right).$$

$\ell_{21} = \frac{-2/3}{4} = -\frac{1}{6}$. $L_2 \leftarrow L_2 + \frac{1}{6}L_1$:

$$\left(\begin{array}{ccc|c} 3 & 8 & 1 & 12 \\ 0 & 4 & 1 & 2 \\ 0 & 0 & \frac{5}{6} & -\frac{5}{3} \end{array}\right).$$

Back-substitution : $x_2 = \frac{-5/3}{5/6} = -2$, $x_1 = \frac{2-1\cdot(-2)}{4} = 1$, $x_0 = \frac{12-8\cdot 1-1\cdot(-2)}{3} = 2$.
Solution : $x = (2, 1, -2)^\top$. ✓

9.3 Architecture MPI : distribution bloc-cyclique

Définition

Avec P processus et une taille de bloc NB, la ligne globale i appartient au processus :

$$\texttt{OWNER}(i) = \left\lfloor \frac{i}{NB} \right\rfloor \bmod P.$$

Son indice local sur ce processus est :

$$\texttt{LOCAL\_LI}(i) = \left\lfloor \frac{\lfloor i/NB \rfloor}{P} \right\rfloor \cdot NB + (i \bmod NB).$$

Exemple à la main

$N = 12$, $NB = 2$, $P = 3$.

Ligne globale	0	1	2	3	4	5	6	7	8	9	10	11
Processus	0	0	1	1	2	2	0	0	1	1	2	2

Contrainte simplificatrice

Dans notre Mini-HPL, N doit être divisible par $NB \times P$. Cela évite la gestion des blocs partiels (présente dans ScaLAPACK/HPL).

9.4 Explication du code

Macros d'indexation

```
/* Acces element (i,j) dans tableau row-major de largeur N */
#define IDX(i, j, N)  ((i) * (N) + (j))

/* Distribution bloc-cyclique (NB et size visibles dans la portee) */
#define OWNER(gi)     (((gi) / NB) % size)
#define LOCAL_LI(gi)  ((((gi) / NB) / size) * NB + (gi) % NB)
```

Listing 1 – Macros définies avant main()

Initialisation

```c
void initialize_matrix(double *A, double *b,
                       int N, int NB,
                       int local_rows, int rank, int size)
{
    for (int li = 0; li < local_rows; li++) {
        int panel_local  = li / NB;
        int within_panel = li % NB;
        int global_i = (panel_local * size + rank) * NB +
           within_panel;
        for (int j = 0; j < N; j++) {
            if (j == global_i)
                A[IDX(li,j,N)] = (double)N + (double)(global_i+j+1)
                   /N;
            else
                A[IDX(li,j,N)] = (double)(global_i+j+1)/N;
        }
        b[li] = 1.0;
    }
}
```

Listing 2 – Matrice diagonalement dominante

Étape 1 — Recherche du pivot : `MPI_DOUBLE_INT`

```c
struct { double val; int idx; } local_cand, global_cand;
local_cand.val = 0.0;
local_cand.idx = -1;

/* ... calcul local du max dans la colonne k ... */

MPI_Allreduce(&local_cand, &global_cand, 1,
              MPI_DOUBLE_INT, MPI_MAXLOC, MPI_COMM_WORLD);

int pivot_gi    = global_cand.idx;
int pivot_owner = OWNER(pivot_gi);
int pivot_li    = LOCAL_LI(pivot_gi);
```

Listing 3 – Réduction MAXLOC (type standard C)

Étape 2 — Échange de lignes (`MPI_Sendrecv`)

```c
MPI_Sendrecv(tmp_row,  N+1, MPI_DOUBLE, pivot_owner, 0,
             pivot_row, N+1, MPI_DOUBLE, pivot_owner, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Listing 4 – Swap entre processus sans deadlock

Étape 3 — Broadcast de la ligne pivot

```
MPI_Bcast(pivot_row, N+1, MPI_DOUBLE, k_owner, MPI_COMM_WORLD);
```

Listing 5 – Diffusion de la ligne pivot + b

Vérification du pivot et arrêt propre

```
double pivot_val = pivot_row[k];
if (fabs(pivot_val) < 1e-14) {
    if (rank == 0)
        fprintf(stderr,
            "ERREUR : pivot quasi-nul a k=%d (|pivot|=%.2e). Arret
               .\n",
            k, fabs(pivot_val));
    MPI_Abort(MPI_COMM_WORLD, 2);
}
```

Listing 6 – Arrêt immédiat si pivot quasi-nul

Étape 4 — Mise à jour SAXPY

```
double factor = A[IDX(li, k, N)] / pivot_val;
for (int j = k; j < N; j++)
    A[IDX(li, j, N)] -= factor * pivot_row[j];
b_vec[li]         -= factor * pivot_row[N];
A[IDX(li, k, N)]  = factor;   /* stocke L in-place */
```

Listing 7 – Mise a jour in-place + LU

9.5 Back-substitution

On résout $Ux = b$ de $k = N-1$ à 0 :

$$x_k = \frac{b_k - \sum_{j=k+1}^{N-1} u_{kj} x_j}{u_{kk}}.$$

```
MPI_Bcast(&x_k, 1, MPI_DOUBLE, k_owner, MPI_COMM_WORLD);
```

Listing 8 – Back-substitution parallèle

9.6 Résiduelle normée HPL

$$r = \frac{\|Ax - b\|_\infty}{\|A\|_\infty \cdot \|x\|_\infty \cdot N \cdot \varepsilon} \qquad \text{PASSED si } r < 16,$$

avec $\varepsilon = 2{,}22 \times 10^{-16}$.

```
MPI_Reduce(&local_res_norm, &global_res_norm, 1,
           MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
MPI_Reduce(&local_A_norm, &global_A_norm, 1,
           MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
```

Listing 9 – Réduction des normes (MPI$_MAX$)

9.7 Tableau récapitulatif des communications MPI

Fonction	Type	Étape	Rôle
`MPI_Allreduce`	`MPI_DOUBLE_INT`	Pivot	Pivot max global (MAXLOC)
`MPI_Sendrecv`	`MPI_DOUBLE`	Swap	Échange de lignes
`MPI_Bcast`	`MPI_DOUBLE`	Gauss	Diffusion ligne pivot
`MPI_Bcast`	`MPI_DOUBLE`	Back-sub	Diffusion x_k
`MPI_Reduce`	`MPI_MAX`	Check	Normes globales
`MPI_Barrier`	–	Timing	Synchronisation

9.8 Compilation et exécution

```
mpicc -O2 -Wall -o mini_hpl mini_hpl.c -lm
mpirun -np 4 ./mini_hpl 1024
# Contrainte : N divisible par NB*P
```

Listing 10 – Compilation et lancement

9.9 Conclusion

Cette implémentation du Mini-HPL parallèle illustre les principes fondamentaux du benchmark HPL : élimination de Gauss avec pivotage partiel, distribution bloc-cyclique des données et coordination via communications MPI. Les opérations collectives (`MPI_Allreduce`, `MPI_Bcast`, `MPI_Reduce`) assurent la cohérence globale du calcul, tandis que le coût dominant reste en $\mathcal{O}(N^3)$. La vérification par la résiduelle normée garantit la validité numérique des résultats. Ce Mini-HPL constitue ainsi une base pédagogique pour comprendre le fonctionnement interne d'un benchmark HPC parallèle.

Références

[1] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary, *HPL — A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*, version 2.3, 2018. `https://www.netlib.org/benchmark/hpl/`

[2] TOP500 Project, *TOP500 Supercomputer Sites*. `https://www.top500.org/`

[3] NVIDIA Corporation, *NVIDIA A100 Tensor Core GPU — Datasheet*, 2020. `https://www.nvidia.com/en-us/data-center/a100/`

[4] NVIDIA Corporation, *NVIDIA H100 Tensor Core GPU — Datasheet*, 2022. `https://www.nvidia.com/en-us/data-center/h100/`

[5] NVIDIA Corporation, *NVIDIA HPC-Benchmarks Container*, NGC Catalog. `https://catalog.ngc.nvidia.com/orgs/nvidia/containers/hpc-benchmarks`