# Un échantillonneur Plug and Play distribué pour la résolution de problèmes inverses de très grande dimension

Maxime BOUTON[1] Pierre-Antoine THOUVENIN[1] Audrey REPETTI[2,3] Pierre CHAINAIS[1]

[1]Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

[2]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh EH14 4AS, UK

[3]Maxwell Institute for Mathematical Sciences, Edinburgh EH8 9BT, UK

**Résumé –** Les algorithmes de Monte-Carlo par chaînes de Markov (MCMC) permettent de quantifier l'incertitude dans les problèmes inverses, en particulier en l'absence de vérité terrain. Plusieurs échantillonneurs inspirés de l'optimisation, tels que PnP-ULA, ont été développés pour intégrer des *a priori* encodés par des réseaux de neurones, améliorant ainsi considérablement la qualité de l'estimation. Mettre en oeuvre de tels algorithmes reste problématique en grande dimension en raison des coûts de calcul et de stockage des échantillons, par exemple en imagerie haute résolution. Ce travail propose une implémentation distribuée sur plusieurs GPUs de PnP-ULA, permettant de traiter rapidement des problèmes inverses de très grande taille sans compromettre la qualité d'estimation. L'approche s'appuie sur une réseau débruiteur encodé par un faible nombre de paramètres, dont la structure est compatible avec une implantation Single Program Multiple Data (SPMD). Le passage à l'échelle est évalué sur données synthétiques pour la résolution de problèmes d'*inpainting* d'images de grande taille.

**Abstract –** Markov Chain Monte Carlo (MCMC) algorithms allow uncertainty quantification in inverse problems, especially when no ground-truth data is available. Optimization-inspired MCMC algorithms, such as PnP-ULA, have been developed to integrate priors encoded by neural networks, significantly improving estimation quality. Applying such algorithms to very high dimensional problems remains challenging due to the computational and storage costs induced by the size of the samples, for example with high-resolution images. This work introduces a distributed implementation of PnP-ULA over several GPUs to quickly solve very high dimensional imaging inverse problems without compromising estimation quality. The approach utilizes a lightweight deep denoiser within a Single Program Multiple Data (SPMD) framework. The interest of this method is demonstrated on high-resolution image inpainting tasks.

## 1 Introduction

Ce travail s'intéresse à la résolution de problèmes inverses en imagerie haute résolution, visant à estimer une image $\overline{\boldsymbol{x}} \in \mathbb{R}^N$ à partir d'observations dégradées $\boldsymbol{y} \in \mathbb{R}^M$, avec $N$ et $M$ de l'ordre de $10^6$ ou plus. Ces problèmes sont basés sur un modèle d'observation de la forme

$$\boldsymbol{y} = \mathcal{A}(\boldsymbol{H}\overline{\boldsymbol{x}}), \tag{1}$$

où $\boldsymbol{H} \in \mathbb{R}^{M\times N}$ représente un opérateur de mesure donné, et $\mathcal{A} : \mathbb{R}^M \to \mathbb{R}^M$ modélise des perturbations aléatoires. La densité de la distribution *a posteriori*, combinant une distribution *a priori* de densité $p$ avec la vraisemblance du modèle, s'écrit

$$\pi(\boldsymbol{x} \mid \boldsymbol{y}) \propto \exp\left(-\phi_{\boldsymbol{y}}(\boldsymbol{H}\boldsymbol{x})\right) p(\boldsymbol{x}), \tag{2}$$

où $\phi_{\boldsymbol{y}} : \mathbb{R}^M \to ]-\infty, +\infty]$ représente l'attache aux données, supposée ici telle que $\phi_{\boldsymbol{y}} \circ \boldsymbol{H}$ est $L_1$-Lipschitz différentiable.

Adaptées d'algorithmes d'inférence standards, les approches *Plug-and-Play* (PnP) [1] remplacent un opérateur proximal associé à un *a priori* explicite par un réseau de neurone débruiteur, dont l'entraînement a permis d'apprendre un modèle génératif de l'espace des solutions. Ces représentations apportent en général une information plus riche que les distributions *a priori* explicites traditionnelles, exploitant par exemple la parcimonie dans un domaine transformé. Parmi les débruiteurs étudiés dans la littérature, les réseaux déroulés comme le Deep Dual Forward-Backward (DDFB) [2] s'inspirent de la structure d'algorithmes d'optimisation itératifs. Ils offrent de bonnes performances de reconstruction avec un nombre de paramètres environ mille fois inférieur à celui de modèles de référence comme DRUNet [3], qui en comporte environ $33\times10^6$. Les méthodes PnP ont été largement étudiées en optimisation [4], et plus récemment dans les algorithmes de Monte Carlo par chaînes de Markov (MCMC) [5].

La quantification de l'incertitude est essentielle pour interpréter les estimations, en particulier en l'absence de données de calibration. Contrairement aux méthodes d'optimisation, les algorithmes MCMC fournissent des estimations ponctuelles assorties d'une quantification d'incertitude. En suivant les mêmes principes qu'en optimisation, des échantillonneurs PnP tels que PnP-ULA [5] et PnP-SGS [6] ont été proposés. Sous certaines hypothèses, il existe une distribution *a priori* propre à laquelle le débruiteur est associé [5].

Le calcul distribué est une approche usuelle pour passer un algorithme à l'échelle, en exploitant plusieurs unités de calcul (CPU ou GPU) appelées *agents*. Cependant, la nature séquentielle des algorithmes MCMC complique leur déploiement sur des architectures distribuées. Une solution consiste à décomposer chaque opération sur plusieurs agents selon le paradigme *Single Program Multiple Data* (SPMD) [7]. Au prix de communications entre les agents, chaque unité im-

Ce travail est soutenu par la Chaire IA Sherlock ANR-20-CHIA-0031-01 portée par P. Chainais, par le programme national d'investissement d'avenir ANR-16-IDEX-0004 ULNE et la Région Hauts-de France, et par le projet IEA CNRS "DAISHI" porté par P.-A. Thouvenin and A. Repetti. Il a également bénéficié d'un accès aux moyens de calcul de l'IDRIS au travers de l'allocation de ressources 2024-AD010615597 attribuée par GENCI.

pliquée exécute une partie du calcul global. De nombreuses opérations typiques en traitement d'image, comme les convolutions, se prêtent à ce type d'implémentation en ce qu'elles ne nécessitent qu'un nombre réduit de communications.

Sous l'hypothèse que $\phi_{\boldsymbol{y}}$ est additivement séparable, cet article propose une version distribuée de PnP-ULA inspirée de [8], basée sur un débruiteur DDFB composé de peu de paramètres. Pour gérer efficacement de très grands problèmes, l'implémentation proposée exploite plusieurs GPUs. À la séquence de nombres pseudo-aléatoires générée près, l'algorithme proposé est équivalent à une version séquentielle de PnP-ULA, et assorti des mêmes garanties de convergence.

L'article est organisé comme suit. La partie 2 décrit les limites d'une implémentation séquentielle de PnP-ULA et introduit la version distribuée proposée. La partie 3 illustre les performances de reconstruction et son efficacité pour l'*inpainting* d'images de grande taille en présence de bruit blanc gaussien. Les conclusions sont reportées en partie 4.

# 2 Échantillonneur PnP-ULA distribué

Cette partie décrit comment distribuer efficacement PnP-ULA [5]. Sous certaines hypothèses, la méthodologie décrite dans [8] pour distribuer des *a priori* plus simples peut être appliquée à des débruiteurs profonds tels que DDFB [2].

## 2.1 Opérateurs localisés et calcul distribué

Adapté de MYULA [9], PnP-ULA [5] consiste à échantillonner la distribution (2) en remplaçant l'opérateur proximal associé à l'*a priori* par un débruiteur gaussien $D_\epsilon$ de variance $\epsilon > 0$, supposé tel que $(D_\epsilon - \mathbf{I}_N)$ est $L_2$-Lipschitzienne. La transition de l'itération $t \in \mathbb{N}$ à $t+1$ est donnée par

$$\begin{aligned} \boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} - \delta \boldsymbol{H}^* \nabla \phi_{\boldsymbol{y}}(\boldsymbol{H}\boldsymbol{x}^{(t)}) + \sqrt{2\delta}\boldsymbol{z}^{(t+1)} \\ &+ \frac{\alpha\delta}{\epsilon}\big(D_\epsilon(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) + \frac{\delta}{\lambda}\big(\Pi_{\mathcal{C}}(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big), \end{aligned} \tag{3}$$

où $\boldsymbol{z}^{(t+1)} \in \mathbb{R}^N$ est une réalisation de la distribution gaussienne multivariée $\mathcal{N}(\mathbf{0}, \mathbf{I}_N)$, $\delta > 0$ est la taille du pas de discrétisation, et $\alpha > 0$ et $\lambda > 0$ sont des paramètres de régularisation. L'opérateur $\Pi_{\mathcal{C}}$ est le projecteur orthogonal sur un ensemble non-vide compact et convexe $\mathcal{C} \subset \mathbb{R}^N$, permettant de garantir la convergence géométrique. En imagerie, l'ensemble $\mathcal{C}$ encode typiquement le domaine de validité de l'intensité des pixels, donné par le produit cartésien $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_N$.

Lorsque $N$ et $M$ sont de l'ordre de $10^6$ et plus, l'implémentation de (3) sur un seul agent devient difficile en raison des limitations matérielles. La nécessité de stocker des milliers d'échantillons pour former des estimateurs et en quantifier l'incertitude, accentue ce problème. Partitionner les variables $\boldsymbol{x}$ et $\boldsymbol{y}$ en $B$ blocs $(\boldsymbol{x}_b)_{1\leq b\leq B}$ et $(\boldsymbol{y}_b)_{1\leq b\leq B}$ peut permettre d'échantillonner chaque bloc $\boldsymbol{x}_b$ en parallèle sur $B$ différents agents, contournant ainsi les problèmes de dimension.

Bien que l'hypothèse usuelle de séparabilité additive de $\phi_{\boldsymbol{y}}$ permette de former des blocs conditionnellement indépendants dans $\boldsymbol{y}$, cela n'est en général par le cas pour $\boldsymbol{x}$, car $\nabla\phi_{\boldsymbol{y}}$ et $D_\epsilon$ ne sont pas nécessairement des opérations terme à terme. Le problème serait sinon trivialement séparable, c'est-à-dire décomposable en $B$ sous-problèmes indépendants. En pratique, échantillonner $\boldsymbol{x}_b^{(t+1)}$ nécessite non seulement l'échantillon $\boldsymbol{x}_b^{(t)}$ mais aussi des éléments d'autres blocs $\boldsymbol{x}_{b'}^{(t)}$, avec $b' \neq b$. Comme illustré en figure 1, chaque agent $b$ doit recevoir des éléments des autres agents à chaque itération pour former $\boldsymbol{x}_b^{(t+1)}$, comme s'il avait accès à $\boldsymbol{x}^{(t)}$ en entier.

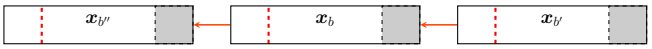



FIGURE 1 : Schéma de communication pour une convolution 1D avec une architecture SPMD. Une partie de $\boldsymbol{x}_{b'}^{(t)}$ est envoyée par l'agent $b'$ à $b$, pour que $\boldsymbol{x}_b^{(t+1)}$ puisse être calculé par $b$.

Le nombre et la taille des communications sont le principal goulot d'étranglement de cette approche. Si les opérateurs impliqués dans l'échantillonnage sont *localisés*, c'est-à-dire que chaque composante du vecteur en sortie peut être calculée à partir de quelques composantes voisines du vecteur en entrée $\boldsymbol{x}$, alors le *couplage* entre les blocs est faible. Les communications résultantes restent de taille limitée. En revanche, si ces couplages sont trop importants, le temps d'exécution de l'échantillonneur distribué sera dominé par des communications coûteuses. En pratique, de nombreux opérateurs en imagerie sont localisés, tels que les convolutions. Lorsque $\mathcal{C} = \bigtimes_{n=1}^N \mathcal{C}_n$ dans (3), la projection $\Pi_{\mathcal{C}}$ s'effectue terme-à-terme. Ainsi, chaque agent $b$ calcule sans communication préalable la projection sur un ensemble $\mathcal{C}_b$, défini comme le produit cartésien des $\mathcal{C}_n$ pour les pixels $n$ contenus dans $\boldsymbol{x}_b$. Des communications interviennent généralement pour distribuer l'évaluation de $D_\epsilon$ et le gradient de la log-vraisemblance, impliquant $\boldsymbol{H}$ et son adjoint $\boldsymbol{H}^*$. La notion de localité mentionnée ci-dessus doit donc être vérifiée non seulement par les opérateurs $\nabla\phi_{\boldsymbol{y}}$, $\boldsymbol{H}$ et $\boldsymbol{H}^*$, mais aussi par le débruiteur $D_\epsilon$.

## 2.2 DDFB comme débruiteur distribué

Les débruiteurs de référence comme les modèles de diffusion [10] offrent d'excellentes performances, mais n'ont en général pas une structure compatible avec une approche SPMD, car ils sont basés sur des opérations non locales, comme les couches d'attention. Les réseaux convolutifs, basés sur une succession de convolutions et d'opérations terme-à-terme, sont une meilleure alternative de ce point de vue.

Les réseaux convolutifs état-de-l'art comme DRUNet se prêtent à une implémentation distribuée, mais leur taille et leur complexité induiraient un coût de communication démesuré. Le réseau déroulé DDFB [2] offre une alternative intéressante car il comporte beaucoup moins de paramètres. Il se compose de $K$ couches, impliquant uniquement les convolutions $\mathbf{G}_k$ et $\mathbf{G}_k^*$, et des opérations de seuillage doux, qui peuvent être distribuées efficacement selon le paradigme SPMD.

## 2.3 Implémentation distribuée proposée

La méthodologie décrite ci-dessus est illustrée sur un problème d'*inpainting* sous bruit blanc gaussien. Le terme d'attache aux données s'écrit

$$(\boldsymbol{x} \in \mathbb{R}^N) \quad \phi_{\boldsymbol{y}}(\boldsymbol{H}\boldsymbol{x}) = \frac{1}{2\sigma^2}\|\boldsymbol{H}\boldsymbol{x} - \boldsymbol{y}\|_2^2, \tag{4}$$

avec $\boldsymbol{H}$ est un opérateur de masquage uniforme et $\sigma^2 > 0$ la variance du bruit. Ce problème satisfait l'ensemble des hypothèses de la partie 2.1. La fonction $\phi_{\boldsymbol{y}}$ est additivement séparable, et $\phi_{\boldsymbol{y}} \circ \boldsymbol{H}$ est $L_1$-Lipschitz différentiable avec $L_1 = \frac{1}{\sigma^2}$.

**Algorithme 1 :** PnP-ULA distribué proposé

**Entrées :** $\boldsymbol{y} \in \mathbb{R}^M, \lambda \in \left(0, (2\alpha L_2/\epsilon + 4L_1)^{-1}\right)$, $\delta \in \left(0, \frac{1}{3}(\alpha L_2/\epsilon + L_1 + 1/\lambda)^{-1}\right), \epsilon > 0, \alpha > 0$

**pour** *chaque agent* $b \in \{1, \dots, B\}$ **faire** en parallèle
- Charger et stocker $\boldsymbol{y}_b \in \mathbb{R}^{M_b}$;
- Initialiser $\boldsymbol{x}_b^{(0)} \in \mathbb{R}^{N_b}$;
- **pour** $t = 0$ **à** $T-1$ **faire**
  - Communiquer à chaque couche du DDFB pour calculer $\boldsymbol{d}_b{}^{(t)} = \left[D_\epsilon(\boldsymbol{x}^{(t)})\right]_b$;
  - `// Mise à jour locale`
  - $\boldsymbol{g}_b{}^{(t)} = \frac{1}{\sigma^2}\boldsymbol{H}_b^*\left(\boldsymbol{H}_b\boldsymbol{x}_b{}^{(t)} - \boldsymbol{y}_b\right)$;
  - Tirer $\boldsymbol{z}_b{}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{N_b})$;
  - $\boldsymbol{x}_b{}^{(t+1)} = \boldsymbol{x}_b{}^{(t)} - \delta\boldsymbol{g}_b{}^{(t)} + \frac{\alpha\delta}{\epsilon}\left(\boldsymbol{d}_b{}^{(t)} - \boldsymbol{x}_b{}^{(t)}\right)$
  - $\quad + \frac{\delta}{\lambda}\left(\Pi_{\mathcal{C}_b}(\boldsymbol{x}_b{}^{(t)}) - \boldsymbol{x}_b{}^{(t)}\right) + \sqrt{2\delta}\boldsymbol{z}_b{}^{(t+1)}$;
- **fin**

**fin**

**Sorties :** $(\boldsymbol{x}^{(t)})_{1\le t\le T}$

L'opérateur $\boldsymbol{H}$ est une opération terme-à-terme, décomposable en masques locaux $(\boldsymbol{H}_b)_{1\le b\le B}$. Aucune communication n'est ainsi nécessaire pour distribuer le calcul de $\nabla(\phi_{\boldsymbol{y}} \circ \boldsymbol{H})(\boldsymbol{x})$.

Un débruiteur DDFB pré-entraîné est utilisé dans l'algorithme 1. Le débruitage de Algorithme 1 algorithme 1 nécessite 2 phases de communications pour chacune des couches $k$ du réseau convolutif : l'une pour appliquer l'opérateur de convolution $\mathbf{G}_k$ et l'autre pour appliquer son adjoint $\mathbf{G}_k^*$. Pour calculer $\mathbf{G}_k\boldsymbol{x}^{(t)}$, chaque agent $b$ doit d'abord communiquer aux agents voisins $b'$, une partie des bordures de la portion d'image $\boldsymbol{x}_b$ dont il s'occupe. Cela permet à chacun de réaliser les même calculs que sur l'image entière. La largeur des bordures est dictée par la taille du noyau de convolution. De même manière, en notant $\boldsymbol{z}^{(t)} = \mathbf{G}_k\boldsymbol{x}^{(t)}$, le calcul de $\mathbf{G}_k^*\boldsymbol{z}^{(t)}$ impose de communiquer des bordures de $\boldsymbol{z}_b{}^{(t)}$ aux agents voisins. D'autres stratégies pourraient être envisagées pour la distribution du réseau DDFB [11].

L'algorithme 1 comporte donc deux phases distinctes : l'une associée au calcul distribué du débruitage, et l'autre à la mise-à-jour de variables locales à chaque agent.

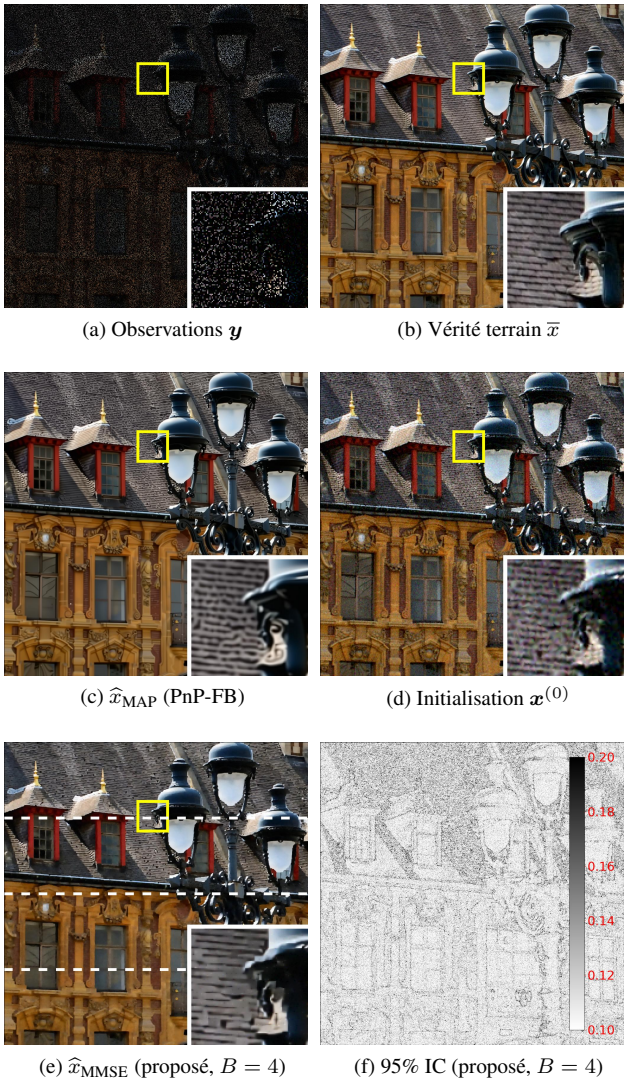
# 3 Illustrations expérimentales

## 3.1 Paramètres expérimentaux

Les performances ont été évaluées sur des problèmes d'inpainting d'images synthétiques $\overline{\boldsymbol{x}} \in [0,1]^N$, avec 80% de données manquantes et un niveau de bruit $\sigma = 0.03$.

Les expériences ont été réalisées sur le supercalculateur Jean Zay (CNRS, hébergé par GENCI à l'IDRIS) sur un seul noeud de calcul muni de 2 processeurs Intel Cascade Lake 6248 (20 cœurs chacun) et 4 GPUs Nvidia Tesla V100 SXM2 (16 Go de RAM chacun). L'implémentation Python proposée[1] repose sur les librairies `mpi4py` pour la parallélisation MPI [12], `cupy` pour le calcul sur GPU [13], et `pytorch` pour les réseaux.

Le réseau DDFB utilisé dans l'algorithme 1 est composé de $K = 19$ couches, et entraîné suivant la procédure décrite dans [2, Paragraphe IV. A. 4), training setting 2]. Le paramétrage de PnP-ULA suit les recommandations de [5], avec $\alpha = 1.25, \mathcal{C} = [-1,2]^N, \epsilon = \sigma^2, \lambda = 0.99(2\alpha L_2/\epsilon + 4L_1)^{-1}$ et $\delta = \frac{0.99}{3}(\alpha L_2/\epsilon + L_1 + 1/\lambda)^{-1}$. Une approximation de $L_2$ est estimée avec la méthode de puissance itérée décrite dans [2, Section 4.4], menant au choix $L_2 = 1$. Les algorithmes sont initialisés par une image $\boldsymbol{x}^{(0)}$ obtenue par interpolation des observations à l'aide de splines bicubiques.




(a) Observations $\boldsymbol{y}$ (b) Vérité terrain $\overline{x}$

(c) $\widehat{x}_{\text{MAP}}$ (PnP-FB) (d) Initialisation $\boldsymbol{x}^{(0)}$

(e) $\widehat{x}_{\text{MMSE}}$ (proposé, $B = 4$) (f) 95% IC (proposé, $B = 4$)

FIGURE 2 : Comparaison de la qualité de restauration.

**Qualité de restauration :** L'estimation du MMSE $\widehat{x}_{\text{MMSE}}$, formée par l'algorithme 1 avec $B = 1$ et 4 GPUs, est comparée à l'estimation du maximum *a posteriori* (MAP) $\widehat{x}_{\text{MAP}}$ obtenue avec l'algorithme d'optimisation PnP-FB [14] basé sur DRUNet [3][2]. La qualité d'estimation est évaluée en terme de SSIM [15] et de rapport signal sur bruit (rSNR) défini par

$$\text{rSNR}(\widehat{\boldsymbol{x}}) = 10\log_{10}\left(\frac{\|\overline{\boldsymbol{x}}\|_2^2}{\|\widehat{\boldsymbol{x}} - \overline{\boldsymbol{x}}\|_2^2}\right), \tag{5}$$

où $\widehat{\boldsymbol{x}}$ est une estimation de la vérité terrain $\overline{\boldsymbol{x}}$. La taille d'image considérée dans cette expérience est $N = 1024^2$. Les estimations $\widehat{x}_{\text{MMSE}}$ ont été calculées sur 20 000 échantillons, dont 5000 pour la période de chauffe. Les intervalles de crédibilité

[1] Accessible à https://gitlab.cristal.univ-lille.fr/phd_maxime_bouton/dpnp-ula-gretsi-2025

[2] Codes et poids utilisés : https://github.com/cszn/DPIR.

TABLE 1 : Résultats de scalabilité faible : configuration, temps moyen par itération et efficacité $\big(t(1)/t(B)\big)$

| #GPUs (taille $N$) | Temps (ms) | Efficacité |
|---|---|---|
| 1 $\left(1024^2\right)$ | $199 \pm 0.32$ | 1 |
| 2 $\left(1448^2\right)$ | $239 \pm 8.86$ | 0.83 |
| 4 $\left(2048^2\right)$ | $284 \pm 31.43$ | 0.70 |

(IC) ont été calculés sur les 500 derniers échantillons pour des contraintes de stockage des échantillons.

**Efficacité :** L'efficacité de l'algorithme 1 a été mesurée en allouant un cœur CPU à chacun des $B \geq 1$ GPUs. La *scalabilité faible* est évaluée en augmentant proportionnellement la taille du problème et le nombre de GPUs alloués dans les configurations suivantes : $(N, B) \in \{(1024^2, 1), (1448^2, 2), (2048^2, 4)\}$. Le temps moyen d'exécution par itération de l'agent le plus lent est reporté comme métrique représentative de la complexité temporelle. Pour $B > 1$, les images $\boldsymbol{x}$ et $\boldsymbol{y}$ ont été divisées équitablement le long de l'axe vertical, suivant lequel les données ne sont pas contiguës en mémoire.

### 3.2 Résultats et discussions

**Qualité de restauration :** Cette expérience vise à comparer la qualité de restauration de l'algorithme distribué, contraint par une certaine catégorie de réseau de neurones, à celle d'un algorithme d'optimisation exploitant un réseau bien plus large et performant, mais non adapté à une implémentation distribuée.

La figure 2 compare la qualité des estimations obtenues par l'algorithme distribué proposé à celles de l'algorithme PnP-FB utilisant DRUNet [3]. L'estimation du MMSE en figure 2e correspond à la configuration $B = 4$ GPUs. Le grossissement est situé sur l'une des zones de communication (délimitées par des pointillés blancs), confirmant qu'aucun artefact dû à des effets de bords n'a été généré par l'implémentation distribuée.

À des écarts minimes près dûs à différentes séquences de nombres pseudo-aléatoires, les métriques des configurations $B = 1$ et $B = 4$ restent similaires, respectivement $(\mathrm{rSNR} = 16.29, \mathrm{SSIM} = 0.78)$ contre $(\mathrm{rSNR} = 16.30, \mathrm{SSIM} = 0.78)$. Malgré un visuel moins convaincant dans les zones texturées, l'estimation du MAP obtenue avec PnP-FB utilisant DRUNet donne de meilleures métriques sur l'image entière : rSNR = 17.71 et SSIM = 0.86.

Une implémentation distribuée pour la résolution de problèmes inverses de grande dimension impose l'utilisation de réseaux de neurones moins complexes, menant à des estimations de qualité légèrement inférieure.

**Efficacité :** Le tableau 1 rassemble les résultats des expériences de *scalabilité faible*. L'efficacité de l'algorithme décroît pour de plus grandes configurations. Bien que le coût calculatoire reste le même du point de vue de chaque GPU, les communications effectuées sont plus grandes. Cependant, le profil de décroissance dépend grandement du réseau utilisé et des coûts de communications qu'il induit. Le DDFB utilisé ici contient $K = 19$ couches, impliquant $19 \times 2 = 38$ étapes de communication. Réduire le nombre de couches réduirait les coûts de communication en proportion. Utiliser une instance de DDFB avec un nombre inférieur de couches ne devrait pas réduire drastiquement la qualité de restauration [2].

## 4 Conclusion

Cet article décrit un algorithme PnP-ULA distribué équivalent à sa version séquentielle. Il exploite plusieurs GPU pour traiter plus rapidement des problèmes de même taille (scalabilité forte) ou passer à l'échelle sur des problèmes de très grande taille (scalabilité faible), sans détériorer la qualité des estimations ou renoncer à la quantification des incertitudes. Son efficacité repose sur l'usage d'un débruiteur profond adapté au calcul distribué. Sa structure doit être la plus légère et la plus simple possible pour réduire les communications, dans un compromis entre qualité de restauration et efficacité calculatoire. La méthode proposée permet par exemple de résoudre un problème d'*inpainting* sur une image couleur de taille $1024^2$ en 35 minutes avec 4 GPUs, contre 1h06 avec un seul. Elle permet aussi de traiter le même problème pour une image 4 fois plus grande avec une augmentation du temps de calcul limitée à un facteur de 1.42.

## Références


[1] U. S. KAMILOV et al., : Plug-and-Play Methods for Integrating Physical and Learned Models in Computational Imaging : Theory, Algorithms, and Applications, *IEEE Signal Process. Mag.*, t. 40, n° 1, p. 85-97, jan. 2023.

[2] H. T. V. LE et al., : Unfolded Proximal Neural Networks for Robust Image Gaussian Denoising, *IEEE Trans. Image Process.*, t. 33, p. 4475-4487, 2024.

[3] K. ZHANG et al., : Plug-and-Play Image Restoration with Deep Denoiser Prior, *IEEE Trans. Pattern Anal. Mach. Intell.*, t. 44, n° 10, p. 6360-6376, 2022.

[4] S. H. CHAN et al., : Plug-and-Play ADMM for Image Restoration : Fixed-Point Convergence and Applications, *IEEE Trans. Comput. Imag.*, t. 3, n° 1, p. 84-98, mars 2017.

[5] R. Laumont et al., : Bayesian imaging using Plug & Play priors: when Langevin meets Tweedie, *SIAM J. Imaging Sci.*, vol. 15, no. 2, pp. 701–737, 2022.

[6] F. COEURDOUX et al., : Plug-and-Play Split Gibbs Sampler : Embedding Deep Generative Priors in Bayesian Inference, *IEEE Trans. Image Process.*, t. 33, p. 3496-3507, 2024.

[7] F. DAREMA, : The SPMD Model : Past, Present and Future, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Berlin, Heidelberg, 2001, p. 1-1.

[8] P.-A. THOUVENIN et al., : A Distributed Split-Gibbs Sampler with Hypergraph Structure for High-Dimensional Inverse Problems, *J. Comput. and Graph. Stat.*, t. 33, n° 3, p. 814-832, oct. 2024.

[9] A. DURMUS et al., : Efficient Bayesian Computation by Proximal Markov Chain Monte Carlo : When Langevin Meets Moreau, *SIAM J. Imaging Sci.*, t. 11, n° 1, p. 473-506, 1er jan. 2018.

[10] J. HO et al., : Denoising Diffusion Probabilistic Models, *Adv. in Neural Information Processing Systems*, t. 33, Curran Associates, Inc., 2020, p. 6840-6851.

[11] B. GALERNE et al., : Scaling Painting Style Transfer, *Computer Graphics Forum*, t. 43, n° 4, e15155, 2024.

[12] L. DALCIN et al., : mpi4py : Status Update After 12 Years of Development, *IEEE Comput. Sci. Eng.*, t. 23, n° 4, p. 47-54, 2021.

[13] R. OKUTA et al., : CuPy : A NumPy-Compatible Library for NVIDIA GPU Calculations, *Adv. in Neural Information Processing Systems*.

[14] M. KOWALSKI et al., : Analysis and Synthesis Denoisers for Forward-Backward Plug-and-Play Algorithms, *preprint*, déc. 2024.

[15] Z. WANG et al., : Image Quality Assessment : From Error Visibility to Structural Similarity, *IEEE Trans. Image Process.*, t. 13, n° 4, p. 600-612, avr. 2004.