Algorithmes parallèles sur le modèle CGM pour la résolution du problème de k-anonymat

Arsème Vadèle Djeufack Nanfack

Algorithmes parallèles sur le modèle CGM pour la résolution du problème de k-anonymat

Arsème Vadèle Djeufack Nanfack *

Université de Dschang - Cameroun
46 Allée des Fauvettes, 78540 Vernouillet - France
arsenenanfackvadele@yahoo.fr

Résumé
Le problème du k-anonymat introduit par Samarati et Sweeney en 1998 consiste à garantir qu'il est impossible de distinguer les données d'un individu d'au moins $(k-1)$ autres présents dans la même base de données. Cependant, les procédés utilisés pour rendre les bases de données k-anonymes doivent modifier les données de la base, rendant celles-ci moins précises par un procédé de généralisation des données stockées. Il engendre ainsi des pertes d'information qui sont générées pendant le processus de généralisation. Plusieurs solutions ont émergé pour la résolution de ce problème parmi lesquelles celle de Mauger, Le Mahec et Dequen, qui ont proposé un algorithme de k-anonymat de complexité quadratique en temps, donnant de bons résultats tout en minimisant les pertes d'informations à l'aide de métriques. Leur solution, bien que satisfaisante, est très coûteuse en temps de calcul et donc limitée dans son exploitation lorsque la taille des données est grande. Le parallélisme est la meilleure façon de remédier à ce problème de performance. Plusieurs modèles de calcul parallèles permettent de concevoir des algorithmes parallèles comme le modèle CGM (Coarse-Grained Multicomputer), qui permet la conception d'algorithmes efficaces et portables. Dans le présent article, nous proposons deux heuristiques parallèles sur le modèle CGM pour minimiser le temps de calcul pendant le processus de k-anonymat. Le premier nommé PSKawMO (Parallel Search K-anonymization with Metric Optimization) et le second nommé PReKawMO (Parallel Reorganized K-anonymization with Metric Optimization) ont respectivement des accélérations théoriques de l'ordre du nombre de processeurs et le carré du nombre de processeurs. Les résultats sur le dataset *Adult* et la plateforme Matrics ont montré que l'algorithme PSKawMO est 4 fois plus rapide tandis que l'algorithme PReKawMO est 1000 fois plus rapide que l'algorithme séquentiel sur 50 processeurs.

Mots-clés : Parallélisme, Modèle de calculs parallèles, Coarse-Grained Multicomputer, Privacy Preserving Data Publishing, K-anonymat

1. Introduction et état de l'art

Le parallélisme est une solution visant à réduire la durée d'exécution des problèmes gourmands en temps de calculs. Plusieurs solutions parallèles sur le modèle CGM ont été proposé les deux dernières décennies pour la résolution de divers problèmes comme le problème d'arbre de recherche binaire optimal [4], le problème de mise entre parenthèses de coût minimum [5], le problème de prédiction de la structure secondaire de l'ARN [18], etc. Le parallélisme consiste à faire coopérer plusieurs unités de calcul (processeurs) simultanément pour résoudre plus rapidement un problème donné. Les algorithmes parallèles sont conçus suivant un modèle de calcul parallèle précis. La prolifération de ces modèles (PRAM [1, 15], BSP [14], CGM [3], etc.) rend la conception d'algorithmes parallèles difficiles. Cependant le modèle CGM

*. Le texte a été relu par Gaël Le Mahec

(Coarse-Grained Multicomputer) [3] a émergé pour être le modèle le plus adapté à l'architecture des machines parallèles actuelles et permet la conception d'algorithmes parallèles efficaces et portables. Un algorithme CGM est une succession de rondes de calcul et de rondes de communication globales.
Plusieurs solutions ont été proposées au fil des années [2, 6, 8, 10, 11, 12, 17] pour garantir le k-anonymat. Certaines préservent mieux la qualité des données que d'autres. Cependant, l'optimisation de la qualité des données au cours du processus de k-anonymat est un problème NP-difficile [2, 9]. Plusieurs heuristiques ont émergé pour obtenir le k-anonymat et minimiser la quantité de données perdues. Ces algorithmes sont généralement gourmands en temps de calculs parmi lesquels celui de Mauger et al. [8] de complexité en temps de $\mathcal{O}\left(n^2\right)$, qui donne de bons résultats et permet de fournir des garanties fortes de protection des données personnelles tout en maintenant une bonne qualité des données. Toutefois, l'exploitation de leur solution devient limitée quand la base de données est grande. Le problème qui se pose est donc celui de la réduction du temps de calcul de cet algorithme. Les auteurs de [11] ont proposé une parallélisation de leur l'algorithme GCCG en mémoire partagée (multithreading) dont l'altération des données croit lorsqu'on augmente le nombre de processeur. Les expérimentations effectuées sur leur solution parallèle (avec uniquement 4 processeurs) et l'architecture parallèle utilisée (ne correspondant pas à celle d'aujourd'hui) sont limitées. À la lumière de nos connaissances actuelles, il n'existe pas de solution parallèle sur le modèle CGM permettant la résolution du problème de k-anonymat. Dans cet article nous nous proposons d'élaborer une solution permettant de rendre les bases de données de grande taille k-anonyme à l'aide du parallélisme.

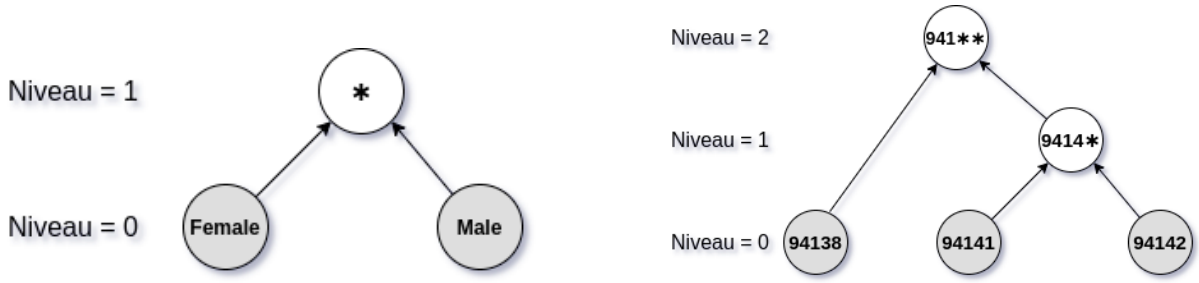
2. K-anonymisation et métriques de perte d'information

le problème de k-anonymisation consiste à garantir l'impossibilité de distinguer les données d'un individu parmi k autres au sein de la même base de données suivant leurs attributs quasi-identifiant. Dans une base de données, on distingue trois types d'attributs notamment les attributs *identifiant*, *quasi-identifiant* et *sensible*. Les attributs identifiants permettent d'identifier les enregistrements de façon unique comme le nom, prénoms etc. Les attributs quasi-identifiants ne renseignent aucune informations sur les usagers mais couplés à des sources des données externes permettent de révéler l'identité des personnes de la base de données (par exemple le code postale, le sexe, la nationalité, etc.). Les attributs sensibles sont les attributs qui sont ne renseignent aucune information sur les individus et sont généralement que l'on souhaite diffuser (par exemple une pathologie médicale, le salaire, etc.). Soit le tableau 1 représentant les données d'une base T_1 à k-anonymiser dont les attributs quasi-identifiants sont *race*, *sexe* et *code postal*.

TABLE 1 – Ensemble de données sur les individus à diffuser

Identifiants		**Quasi-identifiants**			**Attribut sensible**
Id	**Nom**	**Race**	**Sexe**	**Code ZIP**	**Pathologie**
1	Sorelle	Black	Female	94141	Shortness of breath
2	Sylvia	Black	Female	94141	Obesity
3	Sali	White	Male	94138	Chest pain
4	Mark	White	Male	94138	Obesity
5	Venessa	White	Female	94142	Shortness of breath

On peut généraliser les attributs sexe et code postal comme on peut voir sur la figure 1. La figure 4 (voir Annexe) illustre comment partir de l'ensemble de données du tableau 1 pour obtenir une base de données 2-anonyme.



(a) *sexe* (b) *code postal*

FIGURE 1 – Généralisation des attributs quasi-identifiant

Les métriques de perte d'information permettent d'évaluer le taux d'informations d'une base de données k-anonyme par rapport à sa version initiale. Plusieurs métriques existent dans la littérature notamment *Distorsion* [6], *NCP* (Normilized Certainty Penality) [17], *Total* [2], *LLM*, *NLLM*, *WLLM* et *WNLLM* [8]. Dans la suite on notera par μ l'une des sept métriques présentées ci-dessus.

3. Contribution parallèle à la résolution du problème de k-anonymat

Le graphe de dépendances est un élément essentiel pour paralléliser un problème donné. Il consiste à décrire les dépendances qu'il y a entre les sous problèmes d'un problème. Deux itérations consécutives de résolution du problème de k-anonymat [8] sont fortement dépendantes. En effet, étant donné l'itération i et $(i+1)$ l'itération suivante ; l'itération $(i+1)$ nécessite comme entrées, les données produites à la sortie de l'itération i. Cette forte dépendance entre les données de deux itérations consécutives de calcul réduit les sources du parallélisme du problème, ce qui rend difficile la parallélisation du problème de k-anonymat. Dans la suite nous présentons deux solutions parallèles sur le modèle CGM pour minimiser le temps de calcul du problème posé.

3.1. Algorithme parallèle CGM basé sur le partitionnement de l'espace de recherche de la classe de coût de fusion minimal

La solution séquentielle proposée par Mauger et al. [8] (GkAA) se compose de deux étapes principales :

1. La recherche de deux classes d'équivalences dont le coût de fusion est minimal au cours de chaque itération i ;
2. La fusion des deux classes d'équivalence trouvées, puis la répétition de l'étape 1 à l'itération $(i+1)$ avec comme entrée le nouvel ensemble de classes d'équivalences formées à l'itération précédente i.

Une *classe d'équivalence* est une ensemble constitué des enregistrements ayant les mêmes valeurs quasi-identifiant. Lorsqu'on fusionne deux classes d'équivalences, on obtient une nouvelle classe avec pour dimension la somme des dimensions des classes fusionnées. L'étape 1 est un problème de recherche et la parallélisation est possible. En effet le calcul du coût de fusion de deux couples disjoints de classes d'équivalences est complètement indépendant.
L'ensemble des classes d'équivalences $\mathcal{C}(T)$ (avec T l'ensemble de données à k-anonymiser) est partitionné en r sous-ensembles de classes d'équivalences à chaque itération pour effectuer la recherche. En outre chaque processeur effectue la recherche sur un sous-ensemble de classes d'équivalences $\delta\mathcal{C}(T)$ de dimension au plus égale à $\left(\frac{r}{p}+1\right)$, où p est le nombre de processeurs.
L'algorithme 1 résume les grandes étapes de notre solution parallèle. Chaque processeur partitionne l'ensemble des classes d'équivalences $\mathcal{C}(T)$ et sélectionne le sous-ensemble de classes d'équivalences $\delta\mathcal{C}(T)$ pour la phase de calculs locaux. Ensuite chaque processeur effectue la

Algorithme 1 PSKawMO : Algorithme CGM basé sur le partitionnement de l'espace de recherche

1: **procedure** PSKAWMO(T, k, μ, p)
2: Ranger dans l'ordre croissant de taille les $C \in \mathcal{C}(T)$
3: **tant que** $\mathcal{C}_k(T)$ n'est pas vide **faire**
4: Choisir arbitrairement une classe C_{small} de $\mathcal{C}(T)$
5: Chaque processeur détermine son $\delta\mathcal{C}(T)$
6: Recherche de $\delta C \in \delta\mathcal{C}(T) \setminus C_{small}$ par chaque processeur
7: Communication All-To-One de δC au processeur maître
8: Recherche de C par le processeur maître parmi tous le δC reçus
9: Communication One-To-All de C à tous les processeurs
10: $C_{merge} \leftarrow Merge(C_{small}, C)$
11: $\mathcal{C}(T) \leftarrow \mathcal{C}(T) \setminus \{C_{small}, C\} \cup C_{merge}$

recherche dans ce sous-ensemble de classes d'équivalences. A la fin de chaque étape de recherche, les processeurs envoient la classe d'équivalences de coût de fusion minimal au processeur maître (communication du type **All-to-One**). Le processeur maître après la fusion, envoie le nouvel ensemble de classes d'équivalences aux autres processeurs pour la synchronisation (communication du type **One-to-All**). Ainsi chaque itération nécessite deux rondes de communications.

Proposition 1. *L'algorithme CGM PSKwaMO s'exécute dans le pire des cas en* $\mathcal{O}\left(\frac{n^2}{p}\right)$ *avec* $\mathcal{O}(np)$ *rondes de communications.*

Remarque 1. *Le nombre de ronde de communication de notre algorithme* PSKwaMO *dépend de la taille des données du problème* n*, rendant celui-ci peu extensible avec l'accroissement de la taille des données.*

3.2. Algorithme parallèle CGM basé sur le partitionnement de l'ensemble de données

L'objectif de cette solution est de réduire le nombre de ronde de communication engendré par l'algorithme PSKwaMO, tout en limitant tant que possible la dispersion des enregistrements dont on peut supposer qu'ils appartiendront à la même classe d'équivalence. Pour cela, nous introduisons la **réorganisation des données** qui consiste à trier les lignes en fonction de leur *distance* suivant la métrique utilisée dans GkAA. Nous n'avons pas de garantie que le tri ne perturbe pas de manière conséquente les résultats obtenus. Il est probable que l'altération soit un peu plus importante avec notre solution. Cependant, nous espérons que le rapport gain de temps par rapport à l'altération soit très en faveur de notre méthode. L'algorithme 2 décrit les grandes étapes du partitionnement de l'ensemble de données.

Algorithme 2 Procédure de partitionnement de l'ensemble de données

1: **procedure** PARTITIONNEMENT(T, p)
2: Initialiser l'ensemble devant contenir les sous-ensemble ΔT
3: **tant que** T n'est pas vide **faire**
4: Initialiser le sous-ensemble δT
5: Choisir le premier élément E_0 de T comme sentinelle
6: **tant que** le sous-ensemble n'est pas au moins de taille $\left\lfloor \frac{n}{p} \right\rfloor$ **faire**
7: Ajouter l'élément le plus proche de E_0 et tous ceux qui lui sont similaires
8: Ajouter δT à ΔT

Algorithme 3 PReKawMO : Algorithme CGM avec réorganisation basé sur le partitionnement de l'ensemble de données

1: **procedure** PREKAWMO(T, k, μ, p)
2: Réorganiser l'ensemble de données T
3: Partitionner l'ensemble de données T en p sous-ensembles ΔT
4: Chaque processeur détermine son sous-ensemble de données $\delta T \in \Delta T$
5: KawMO $(\delta T, k, \mu)$
6: Communication All-To-One des $\delta T_{\mu,k}$ au processeur maître
7: Construction de $T_{\mu,k}$ par le processus maître

Chaque processeur réorganise l'ensemble de données et le partitionne en p sous-ensembles, puis choisit un sous-ensemble et exécute l'algorithme séquentiel de Mauger et al. [8] sur celui-ci. A la fin de l'exécution de l'algorithme, chaque processeur envoi la table k-anonyme obtenue localement au processeur maître pour la construction de la table k-anonyme résultante. L'algorithme 3 résume les grandes lignes de notre solution.

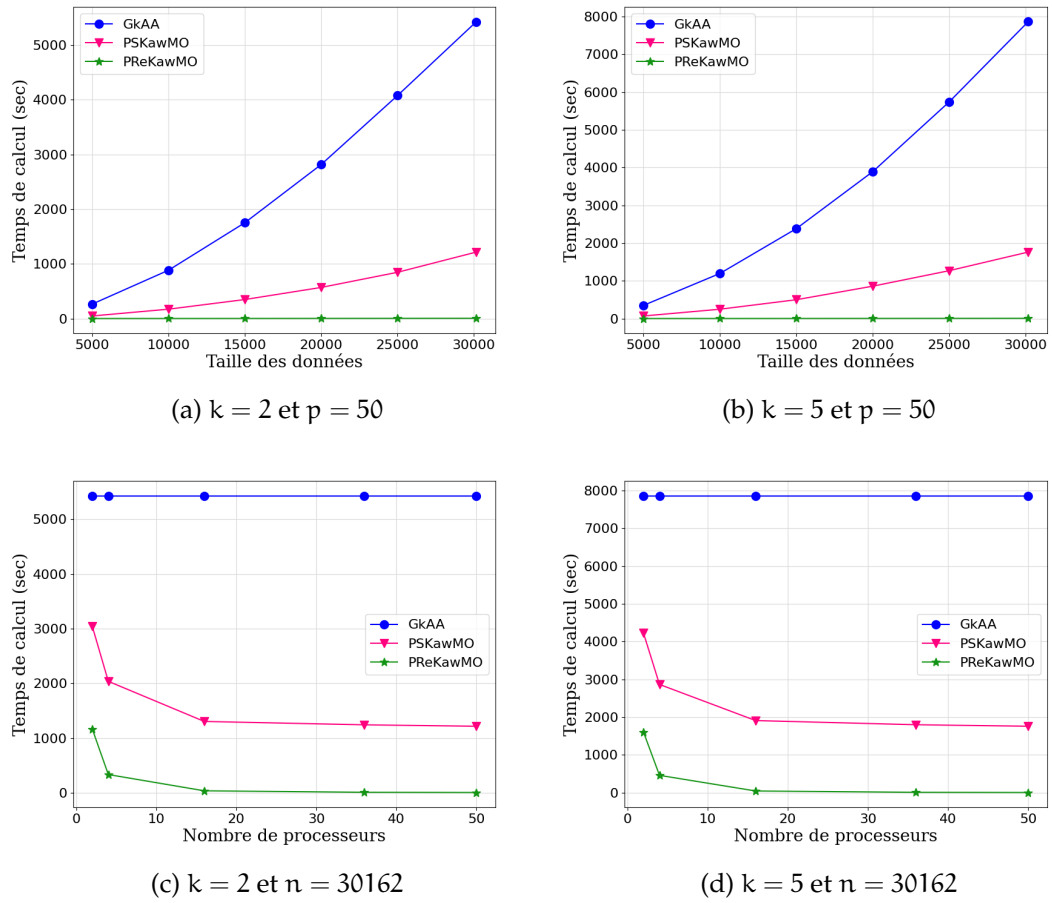
Proposition 2. *l'algorithme CGM* PReKwaMO *s'exécute dans le pire des cas en* $\mathcal{O}\left(\frac{n^2}{p^2}\right)$ *avec* $\mathcal{O}(p)$ *rondes de communications.*

4. Résultats Expérimentaux

Nous avons comparés les résultats de nos algorithmes parallèles avec la solution séquentielle de Mauger et al. [8]. Ils ont été implémentées en C++/Python2.7, puis exécutés sur un système d'exploitation CentOS Linux du cluster de calcul Matrics de l'université de Picardie Jules Verne [7]. Pour l'implémentation des communications entre les processeurs, nous avons utilisé la distribution mpi4py pour python de la bibliothèque MPI (Message Parsing Interface). Pour nos expérimentations, nous avons choisi de travailler avec l'ensemble de données *Adult* disponible sur le dépôt d'apprentissage machine de l'université de Californie à Irvine (UCI) [13]. Il est généralement utilisé pour effectuer des expérimentations sur les algorithmes de k-anonymat. Les résultats présentés sont en fonction de différentes valeurs du triplet (n, p, k). n correspond à la taille des données dont les valeurs sont prises dans $\{5000, 10000, 15000, 20000, 25000, 30162\}$; p est le nombre de processeurs avec les valeurs dans $\{2, 4, 8, 16, 28, 36, 50\}$; k est le facteur d'anonymat dont les valeurs sont prises dans $\{2, 5\}$. Lorsque $p = 1$, les algorithmes exécutent la solution séquentielle de Mauger et al. [8].

4.1. Évaluation du temps de calcul des différentes solutions parallèles

Les résultats montrent que les algorithmes PSKa*w*MO, PReKa*w*MO sont respectivement 4 et 1000 fois plus rapide que l'algorithme séquentiel de Mauger et al. GkAA [8] sur 50 processeurs pour $n = 30162$ et $k = 2$. En outre, on observe que le temps d'exécution diminue le nombre de processeurs augmente (voir figures 2c et 2d). PReKa*w*MO minimise le temps d'exécution en gardant les processeurs occupés le plus longtemps possible. En effet la taille des sous ensembles de données constituant les sous problèmes de k-anonymat, diminue avec l'accroissement du nombre de processeur. Une telle accélération s'explique par la complexité quadratique en fonction du nombre de lignes de la base de données de l'algorithme séquentiel. En découpant la base de données en autant de parties que le nombre de processeurs, on obtient une accélération théorique de l'ordre du nombre de processeurs au carré. Le risque étant alors d'altérer de manière rédhibitoire les données. Dans la section nous analyserons l'influence de la parallélisation sur la qualité des données de la base k-anonyme.
Le temps de communication entre processeurs est négligeable devant le temps de calculs pour la solution PReKa*w*MO présentée ci-dessous, et décroît avec l'augmentation du nombre de processeurs. En effet, sur le modèle CGM, le temps de communication dépend non seulement



(a) k = 2 et p = 50

(b) k = 5 et p = 50

(c) k = 2 et n = 30162

(d) k = 5 et n = 30162

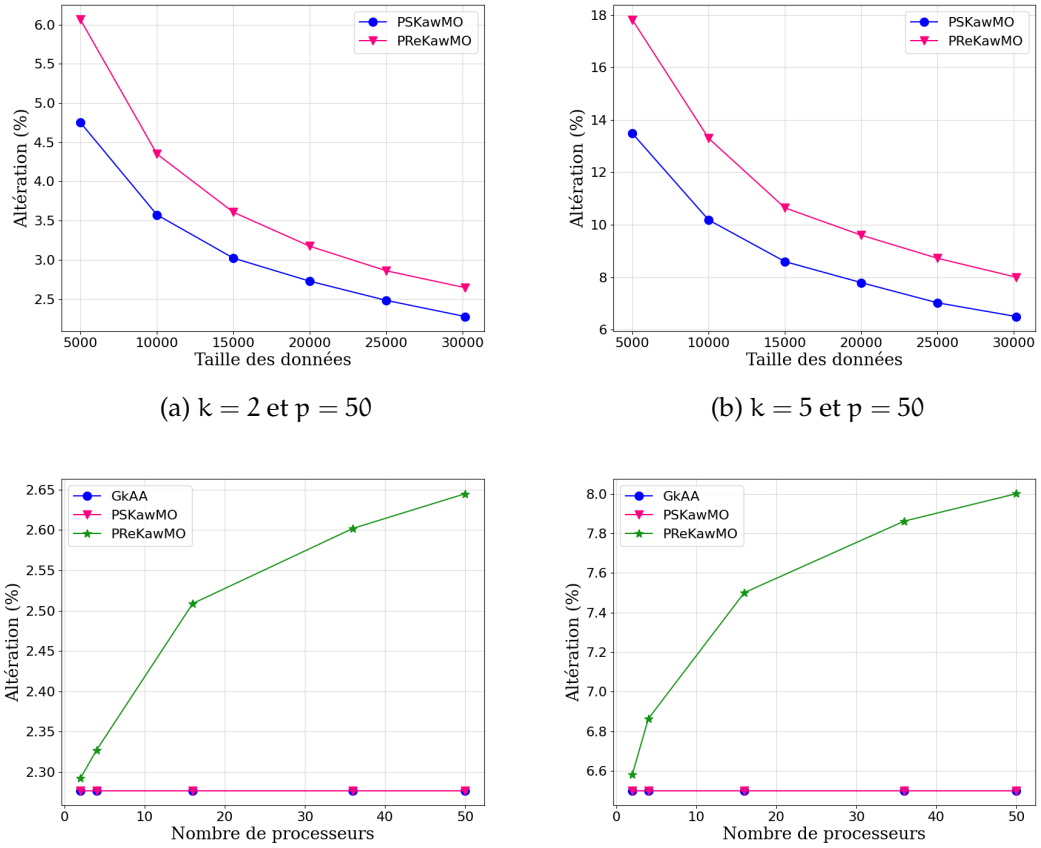
FIGURE 2 – Évolution du temps de calcul avec l'accroissement du nombre de processeurs et de la taille des données

du nombre de processeurs en jeu mais aussi de la taille des données envoyées. Ainsi plus le nombre de processeurs est grand, plus la taille des données à transmettre est petite. En particulier, l'algorithme PReK*aw*MO a un temps de communication de $0,01$ sec lorsque $n = 30162$, $p = 50$ et $k = 5$.

4.2. Évaluation de l'altération des données

On observe sur la figure 3 que l'altération des données diminue avec l'accroissement de la taille des données pour les solutions séquentielle et parallèles. Cela montre que nos solutions parallèles sont extensibles à l'accroissement des données. De plus l'algorithme PReK*aw*MO a une altération de 8% tandis que l'algorithme PSK*aw*MO a la même altération que l'algorithme séquentiel (soit 5,5%) sur 50 processeurs lorsque $k = 5$ et $n = 30162$. En effet, on observe sur les figures 3c et 3d que l'algorithme PReK*wa*MO apporte un peu plus d'altération que la solution séquentielle. Cette dégradation est à dû au fait que les sous ensembles obtenu lors du partitionnement peuvent être de petite taille et les valeurs des quasi-identifiant prennent alors pour généralisation des valeurs les plus élevée dans hiérarchie de généralisation, causant l'augmentation du coût d'altération.

Les pré-traitements effectués sur les données nous permettent donc de distribuer les calculs sur un grand nombre de processeurs sans détérioration de la qualité des données de la base k-anonyme finale.



(a) k = 2 et p = 50

(b) k = 5 et p = 50

(c) k = 2 et n = 30162

(d) k = 5 et n = 30162

FIGURE 3 – Évolution de l'altération des données avec l'accroissement du nombre de processeurs et de la taille des données

5. Conclusion et perspectives

En somme, nous avons proposé deux algorithmes parallèles sur le modèle CGM qui minimisent le temps de calculs lors d'un processus de k-anonymat. Chacune de ces solutions basée sur l'heuristique de Mauger et al. [8], utilise des approches de partitionnements différentes. Les résultats obtenus montrent bien que l'accélération très élevée de l'algorithme PReK*aw*MO (1000 fois plus rapide que la solution séquentielle sur 50 processeurs) compense largement le supplément de pertes d'information (environ 3% sur 50 processeurs pour k = 5) par rapport à la solution séquentielle [8]. L'algorithme PReK*aw*MO est *efficace* et *extensible* avec l'accroissement de la taille des données.

Comme perspectives de ce travail, nous comptons trouver des procédés, nous permettant d'améliorer la qualité des données de la solution parallèle PReK*aw*MO, en réduisant le nombre de découpage de l'ensemble de données d'une part et l'utilisation des travaux récents de Viton et al. [16] sur l'amélioration de l'utilité des données des bases de données k-anonymes d'autre part. De plus nous comptons également améliorer l'accélération de nos solutions parallèles en se servant du parallélisme automatique en mémoire partagée.

Bibliographie

1. Akl (S.). – Parallel synergy or can a parallel computer be more efficient than the sum of its parts.
2. Byun (J.-W.), Kamra (A.), Bertino (E.) et Li (N.). – Efficient k-anonymization using clustering techniques. *E., eds., Advances in Databases : Concepts, Systems and Applications*, 2007, pp. 188–200.
3. Dehne (F.), Fabri (A.) et Rau-Chaplin (A.). – Scalable parallel geometric algorithms for coarse grained multicomputers. *In Proceedings of the ninth annual symposium on Computational geometry*, 1993.
4. Kengne Tchendji (V.) et Lacmou Zeutouo (J.). – An efficient cgm-based parallel algorithm for solving the optimal binary search tree problem through one-to-all shortest paths in a dynamic graph. *Data Science and Engineering*, vol. 4, n42, 6 2019.
5. Lacmou Zeutouo (J.), Kengne Tchendji (V.) et Myoupo (J. F.). – High-performance cgm-based parallel algorithms for minimum cost parenthesizing probleme. *The Journal of Supercomputing*, 2021.
6. Li (J.), Wong (R. C.-W.), Fu (A. W.-C.) et Pei (J.). – Achieving k-anonymity by clustering in attribute hierarchical structures. *Data Warehousing and Knowledge Discovery*, 2006, pp. 405–416.
7. Matrics. – Matrics platform, 2021.
8. Mauger (C.), Le Mahec (G.) et Dequen (G.). – Modeling and evaluation of k-anonymization metrics. *Privacy Preserving Artificial Intelligence Workshop of AAAI20*, 2020, pp. 1–8.
9. Meyerson (A.) et Williams (R.). – On the complexity of optimal k-anonymity. *In Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, vol. 4, 2004, pp. 223–228.
10. Samarati (P.). – Protecting respondents identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, n6, 11 2001, pp. 1010–1027.
11. Sang (N.), Mengbo (X.) et Quan (Q.). – Clustering based k-anonymity algorithm for privacy preservation. *International Journal of Network Security*, vol. 19, n6, 11 2017, pp. 1062–1071.
12. Sweeney (L.). – Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Puzziness and Knowledge-Based Systems*, vol. 10, n5, 2002, pp. 571–588.
13. UCIrvine. – Machine learning repository, 1987.
14. Valiant (L. G.). – Bulk-synchronous parallel computers. *Parallel Processing and Artificial Intelligence*, 1989.
15. Valiant (L. G.). – A bridging model for parallel computation. *Communications of the ACM*, vol. 33, n8, 1990.
16. Viton (F.), Mauger (C.), Dequen (G.), Guérin (J.-L.) et Le Mahec (G.). – Proportional representation to increase data utility in k-anonymous tables. – In *26th IEEE Symposium on Computers and Communications*, Athènes, Greece, septembre 2021.
17. Xu (J.), Wang (W.), Pei (J.), Wang (X.), Shi (B.) et Fu (A. W.-C.). – Utility-based anonymization using local recoding. *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, 2006, pp. 785–79.
18. Y. KAMGA (F. I.). – Predicting rna secondary structure : An efficient computation based on automatic parallelization techniques, 8 2020.

Annexes

Exemple d'obtention d'une base k-anonyme

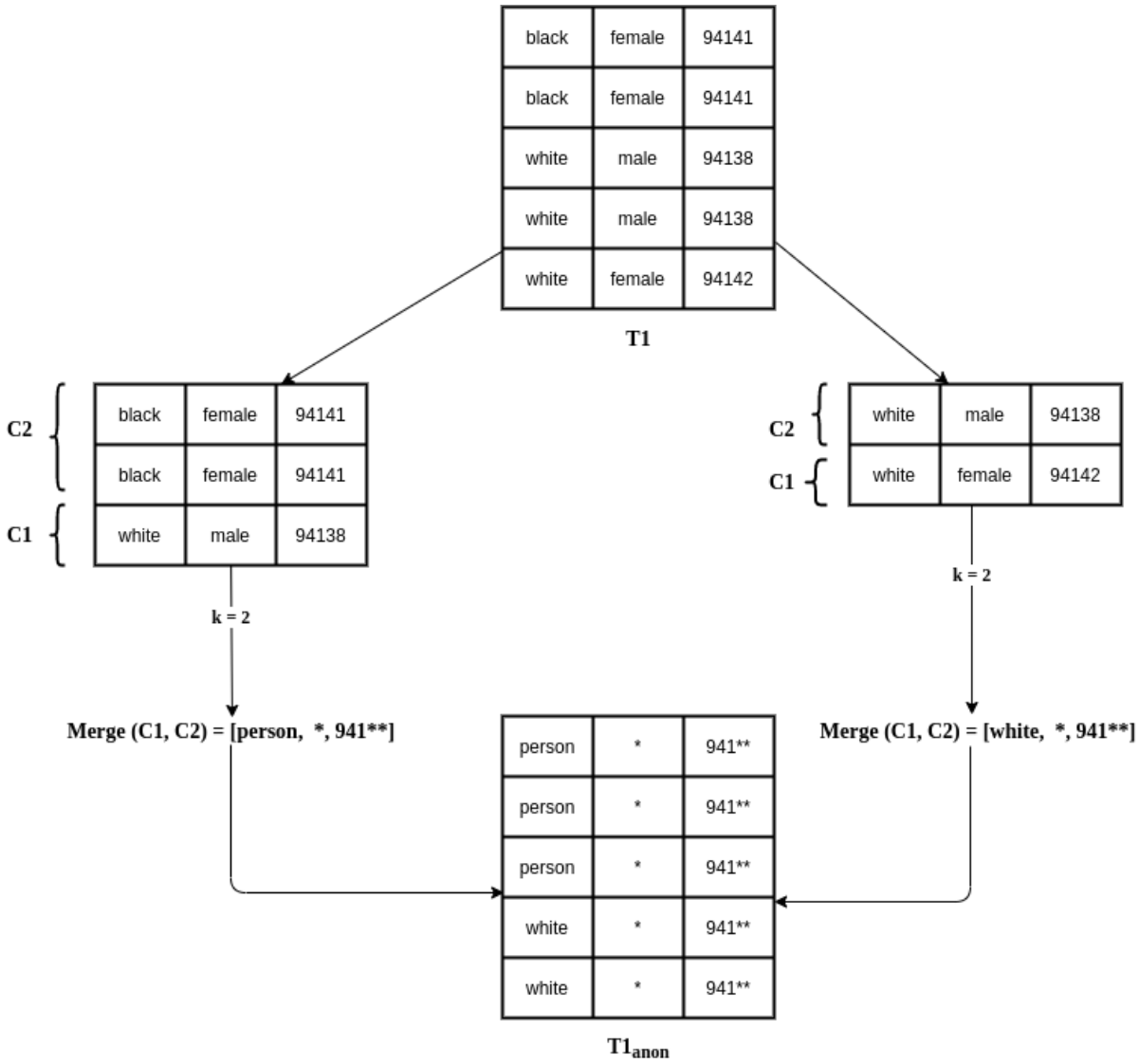


FIGURE 4 – 2-anonymisation d'une bases de données

Preuves

Preuve 1. *(Complexité de l'algorithme 1)*
Chaque processeur effectue $\left(\frac{r-1}{p}\right)$ *étapes pour la recherche de la classe de coût de fusion minimal* δC *à chaque itération, où* r *est le nombre de classes d'équivalences à cette itération. Ainsi le temps de calculs pour un processeur est donné par :*

$$t_{cals}(n) = \left(\frac{r-1}{p}\right) + \left(\frac{r-2}{p}\right) + \cdots + \left(\frac{r-s}{p}\right) = \frac{1}{p}\sum_{i=1}^{s}(r-i) = \frac{s(2r-s-1)}{2p}$$

Ainsi, $r = n$ *et* $s \leq n$ *dans le pire des cas. On obtient donc :*

$$t_{cals}(n) = \frac{n(n-1)}{2p}$$

D'où la complexité en temps est de $\mathcal{O}\left(\frac{n^2}{2p}\right)$, *soit* $\mathcal{O}\left(\frac{n^2}{p}\right)$. *Chaque itération de la boucle nécessite* 2 *rondes de communication All-to-one et One-to-all, de* $(p-1)$ *communications chacune. Ainsi on a* $2(p-1)$ *communications par itération. Dans le pire des cas,* $k = n$ *et on a :*

$$t_{comms}(n) = \sum_{i=1}^{n} 2(p-1) = 2n(p-1)$$

D'où cet algorithme nécessite $\mathcal{O}(2np)$ *communications, soit* $\mathcal{O}(np)$ *ronde de communications.*

Preuve 2. *(Complexité de l'algorithme 3)*
Chaque processeur effectue $(r-1)$ *nombres d'étapes pour la recherche de la classe de coût de fusion minimal à chaque itération. Ainsi le temps d'exécution pour un processeur est donné par :*

$$t_{cals}(n) = (r-1) + (r-2) + \cdots + (r-s) = \sum_{i=1}^{s} (r-i) = \frac{s(2r-s-1)}{2}$$

Dans le pire des cas, $r = \frac{n}{p}$ *et* $k = \frac{n}{p}$ *(c'est-à-dire que* $s \leq \frac{n}{p}$*). On obtient donc :* $t_{cals}(n) = \frac{n}{2p} \times \left(\frac{n}{p} - 1\right)$. *D'où la complexité en temps est de* $\mathcal{O}\left(\frac{n^2}{2p^2}\right)$, *soit* $\mathcal{O}\left(\frac{n^2}{p^2}\right)$.
Une ronde de communication All-to-one est nécessaire à la fin des calculs locaux. Ainsi on obtient $(p-1)$ *communications. D'où un nombre de communications de l'ordre de* $\mathcal{O}(p-1)$, *soit* $\mathcal{O}(p)$.

Tableaux des accélérations des algorithmes parallèles proposés
Nous représentons dans les tableaux 1 et 2, les valeurs des accélérations obtenues respectivement pour les algorithmes 1 et 3.

Valeur de k	Processeurs								
	1	2	4	8	16	20	28	36	50
	n =25000								
2	.	2.01	2.73	3.63	4.21	4.29	4.34	4.42	4.54
3	.	2.0	2.59	3.36	3.91	4.05	4.1	4.17	4.25
5	.	2.0	2.52	3.35	4.1	4.19	4.17	4.26	4.33
	n =30162								
2	.	2.01	3.0	3.78	4.35	4.45	4.44	4.54	4.62
3	.	2.03	2.71	3.51	3.91	4.08	4.12	4.19	4.22
5	.	2.02	2.66	3.41	3.86	4.15	4.13	4.21	4.29

TABLE 2 – Facteur d'accélération de l'algorithme CGM basé sur le partitionnement de l'espace de recherche PSK*a*w*M*O

Valeur de k	Processeurs								
	1	2	4	8	16	20	28	36	50
	n =25000								
2	.	4.65	16.2	58.09	151.8	218.87	434.08	605.25	1129.23
3	.	4.63	16.23	58.82	158.21	230.36	451.8	656.62	1196.81
5	.	4.66	16.5	59.83	164.02	238.26	466.2	686.65	1254.68
	n =30162								
2	.	4.6	15.88	56.34	149.48	211.25	419.22	611.39	1128.82
3	.	4.69	16.23	58.78	158.4	225.38	444.25	670.1	1227.93
5	.	4.73	16.51	59.33	163.44	233.98	461.99	697.41	1273.87

TABLE 3 – Facteur d'accélération de l'algorithme CGM basé sur le partitionnement de l'ensemble de données PReKawMO