Rapport de TER - Analyse de l'algorithme `ExtractVector`

BENAMROUCHE Malek

8 mai 2025

1 Introduction

1.1 Contexte et problématique

L'arithmétique en virgule flottante est fondamentale pour les calculs scientifiques modernes, mais la représentation limitée des nombres réels introduit des erreurs d'arrondi qui peuvent compromettre la fiabilité des résultats. Cette instabilité numérique est critique dans les domaines exigeant une haute précision comme la simulation numérique ou l'informatique quantique. Notre travail se concentre sur la sommation précise de grands ensembles de nombres à virgule flottante, opération particulièrement vulnérable aux erreurs d'arrondi lorsque les opérandes présentent des ordres de grandeur disparates [2].

Cette problématique est particulièrement pertinente dans l'algèbre linéaire numérique et le calcul haute performance (HPC), où des milliards d'opérations sont exécutées simultanément sur des clusters distribués. Les simulateurs d'ordinateurs quantiques constituent également un domaine d'application crucial, nécessitant des calculs d'amplitudes complexes d'une précision rigoureuse. Le défi réside dans l'équilibre entre précision numérique et efficacité computationnelle.

1.2 Approche proposée et contributions

L'algorithme `ExtractVector` que nous étudions repose sur une décomposition hiérarchique des opérandes, séparant chaque composante du vecteur d'entrée en une partie "haute" (significative) et une partie "basse" (résiduelle). Cette approche est naturellement adaptable aux systèmes distribués et particulièrement adaptée aux architectures HPC modernes. L'originalité de notre contribution réside dans l'analyse mathématique rigoureuse de l'algorithme et son potentiel d'application dans des domaines émergents. Nous apportons une preuve formelle qui garantit les bornes d'erreur et établit les conditions nécessaires à son fonctionnement optimal, notamment concernant le paramètre σ qui assure l'exactitude des opérations critiques.

1.3 Plan du rapport

La suite de ce rapport s'organise comme suit : la section 2 détaille le fonctionnement de l'algorithme `ExtractVector`. La section 3 présente le cadre théorique de l'arithmétique flottante. La section 4 développe sa preuve de validité, structurée en lemmes distincts pour en faciliter la compréhension. La section 5 expose l'implémentation réalisée. Enfin,

la section 7 synthétise nos contributions et propose des perspectives d'extension pour les systèmes distribués à grande échelle.

Notations. Dans la suite, $p = (p_1, \ldots, p_n)$ désigne un vecteur de flottants de taille $n \leq 2^N$, M l'entier tel que $\forall i, |p_i| \leq 2^M$, P le nombre de bits de précision, et $\sigma = \frac{3}{2} \times 2^{M+N}$. L'opération $\mathrm{fl}(x)$ représente l'arrondi du réel x au flottant le plus proche. On choisit N et M minimaux, et on pose $s = M + N - P + 1$.

2 Algorithme

Algorithm 1 `ExtractVector`(σ, p)

1: $\tau \leftarrow 0$
2: **for** $i = 1$ to n **do**
3: $\quad q_i \leftarrow \mathrm{fl}((\sigma + p_i) - \sigma)$
4: $\quad p'_i \leftarrow \mathrm{fl}(p_i - q_i)$
5: $\quad \tau \leftarrow \mathrm{fl}(\tau + q_i)$
6: **end for**
7: **return** (τ, p')

L'algorithme fonctionne en extrayant des parties "hautes" des p_i qui sont sommables exactement, et en rendant un vecteur p' contenant les parties "basses" des p_i. L'idée étant d'appliquer ExtractVector sur p' avec un nouveau σ par la suite.

L'algorithme `ExtractVector` que nous étudions ici a été initialement proposé par Rump et al. [3] dans le cadre de leurs travaux sur la sommation de haute précision. Notre analyse s'appuie sur leurs fondements théoriques tout en proposant une formalisation et une vérification rigoureuses des propriétés de l'algorithme.

3 Cadre théorique

Définition 3.1. Un *flottant* est un nombre représentable exactement par le type `float`. Il peut s'écrire sous la forme $m \cdot 2^x$, avec x plus grand que l'exposant minimal e et plus petit que l'exposant maximal E, et $|m|$ supérieur à 0 et inférieur à 2^p, p étant le nombre de bits de précision.

Cette définition est conforme à la représentation standardisée des nombres à virgule flottante, telle que décrite dans [5] et formalisée dans la norme IEEE 754 [2].

Définition 3.2. $\mathrm{fl}(x)$ désigne le flottant le plus proche du réel x selon l'arrondi de la machine.

Théorème 3.3 (Addition exacte)**.** *Soient a et b deux flottants. Leur somme est exacte (sans erreur d'arrondi) si $-\frac{a}{b} \in \left[\frac{1}{2},\ 2\right]$.*

Théorème 3.4 (Erreur d'arrondi)**.** *Soient a et b deux flottants. $\mathrm{fl}(a+b) - (a+b)$ est un flottant.*

4 Preuve principale

Commençons par rappeler nos hypothèses et définir précisément le cadre de notre analyse.

4.1 Définitions et hypothèses préliminaires

Définition 4.1 (Format flottant)**.** Nous considérons le format flottant binaire avec P bits de précision, un exposant minimal e et un exposant maximal E. Un nombre flottant peut s'écrire sous la forme $m \cdot 2^k$ où $m \in \mathbb{Z}$ avec $0 \leq |m| < 2^{P-1}$, et $e \leq k \leq E$.

Définition 4.2 (Arrondi au plus proche)**.** Pour tout nombre réel x, nous notons $\mathrm{fl}(x)$ l'arrondi au plus proche de x dans le format flottant considéré. En cas d'égalité, nous supposons un arrondi au nombre pair (arrondi de type "round to nearest, ties to even").

Hypothèse 4.3. Nous posons les hypothèses suivantes :

1. Le vecteur d'entrée $p = (p_1, \ldots, p_n)$ comporte au plus 2^N éléments : $n \leq 2^N$.
2. Chaque composante p_i est un flottant vérifiant $|p_i| \leq 2^M$.
3. Le paramètre σ est défini par $\sigma = \frac{3}{2} \times 2^{M+N}$.
4. Le paramètre s est défini par $s = M + N - P + 1$.
5. Le paramètre σ vérifie $2^{P+E-1} \geq \sigma \geq 2^{e+1}$, où E est l'exposant maximal et e l'exposant minimal du format flottant.

Théorème 4.4 (Propriétés fondamentales de `ExtractVector`)**.** *Sous les hypothèses 4.3, l'algorithme* ***`ExtractVector`*** *appliqué au vecteur p avec le paramètre σ produit un scalaire τ et un vecteur p' tels que :*

1. $|p'_i| \leq 2^{s-1}$ *pour tout* $i \in \{1, \ldots, n\}$
2. $\sum_{i=1}^{n} p_i = \tau + \sum_{i=1}^{n} p'_i$

Pour démontrer ce théorème, nous allons établir une série de lemmes qui analysent les propriétés des opérations effectuées dans l'algorithme `ExtractVector`.

Lemme 4.5 (Extraction exacte de la partie haute)**.** *Pour tout $i \in \{1, \ldots, n\}$, on a :*

$$\mathrm{fl}(\mathrm{fl}(\sigma + p_i) - \sigma) = \mathrm{fl}(\sigma + p_i) - \sigma$$

Démonstration. Pour démontrer cette propriété, nous devons prouver que $\mathrm{fl}(\sigma + p_i) - \sigma$ est un flottant, ce qui implique que l'opération d'arrondi $\mathrm{fl}(\mathrm{fl}(\sigma + p_i) - \sigma)$ est exacte.

Distinguons deux cas en fonction de la valeur de p_i.

Cas 1 : $-\sigma \leq p_i \leq -\dfrac{\sigma}{2}$. Dans ce cas :

$$-\sigma \leq p_i \leq -\frac{\sigma}{2} \Rightarrow \frac{p_i}{\sigma} \in \left[-1, -\frac{1}{2}\right] \Rightarrow -\frac{p_i}{\sigma} \in \left[\frac{1}{2}, 1\right]$$

Par le théorème 3.3 (addition exacte sous condition sur le rapport), la condition $-\frac{p_i}{\sigma} \in \left[\frac{1}{2}, 1\right]$ implique que l'addition $\sigma + p_i$ est exacte :

$$\mathrm{fl}(\sigma + p_i) = \sigma + p_i$$

Par conséquent :

$$\mathrm{fl}(\sigma + p_i) - \sigma = (\sigma + p_i) - \sigma = p_i$$

Puisque p_i est un flottant par hypothèse, nous avons :

$$\mathrm{fl}(\mathrm{fl}(\sigma + p_i) - \sigma) = \mathrm{fl}(p_i) = p_i = \mathrm{fl}(\sigma + p_i) - \sigma$$

ce qui prouve l'égalité attendue pour ce cas.

Cas 2 : $p_i > -\dfrac{\sigma}{2}$. Dans ce cas, nous devons démontrer que $\mathrm{fl}(\sigma + p_i) - \sigma$ est un flottant, ce qui implique :

$$\mathrm{fl}(\mathrm{fl}(\sigma + p_i) - \sigma) = \mathrm{fl}(\sigma + p_i) - \sigma$$

Posons $a = -\sigma$ et $b = \mathrm{fl}(\sigma + p_i)$. Nous allons montrer que $-\frac{a}{b} \in \left[\frac{1}{2}, 2\right]$, ce qui permettra d'appliquer le théorème 3.3 pour prouver que la soustraction $b + a = \mathrm{fl}(\sigma + p_i) - \sigma$ est exacte.

D'une part, nous avons :

$$\mathrm{fl}(\sigma + p_i) \leq \mathrm{fl}(2\sigma) = 2\sigma$$

car $|p_i| \leq 2^M \leq \sigma$ implique $\sigma + p_i \leq 2\sigma$, et $2\sigma = 3 \times 2^{M+N}$ est exactement représentable.

Cela donne :

$$-\frac{a}{b} = \frac{\sigma}{\mathrm{fl}(\sigma + p_i)} \geq \frac{\sigma}{2\sigma} = \frac{1}{2}$$

D'autre part, comme $p_i > -\frac{\sigma}{2}$, nous avons :

$$\sigma + p_i > \sigma - \frac{\sigma}{2} = \frac{\sigma}{2}$$

Par la propriété de croissance de la fonction d'arrondi :

$$\mathrm{fl}(\sigma + p_i) \geq \mathrm{fl}\left(\frac{\sigma}{2}\right) = \frac{\sigma}{2}$$

car $\frac{\sigma}{2} = \frac{3}{4} \times 2^{M+N}$ est exactement représentable.

Cela donne :

$$-\frac{a}{b} = \frac{\sigma}{\mathrm{fl}(\sigma + p_i)} \leq \frac{\sigma}{\frac{\sigma}{2}} = 2$$

Nous avons donc :

$$\frac{1}{2} \leq -\frac{a}{b} \leq 2$$

Par application du théorème 3.3 avec $a = -\sigma$ et $b = \mathrm{fl}(\sigma + p_i)$, nous concluons que la soustraction est exacte :

$$\mathrm{fl}(b + a) = b + a$$

Soit :

$$\mathrm{fl}(\mathrm{fl}(\sigma + p_i) - \sigma) = \mathrm{fl}(\sigma + p_i) - \sigma$$

Ce qui complète la preuve du lemme. □

Lemme 4.6 (Calcul exact de la partie basse)**.** *Pour tout $i \in \{1, \ldots, n\}$ et avec $q_i = \mathrm{fl}(\mathrm{fl}(\sigma + p_i) - \sigma)$, on a :*

$$\mathrm{fl}(p_i - q_i) = p_i - q_i$$

Démonstration. Par le lemme 4.5 et la définition de q_i, nous avons :

$$q_i = \mathrm{fl}(\mathrm{fl}(\sigma + p_i) - \sigma) = \mathrm{fl}(\sigma + p_i) - \sigma$$

Par conséquent :

$$p_i - q_i = p_i - (\mathrm{fl}(\sigma + p_i) - \sigma) = p_i + \sigma - \mathrm{fl}(\sigma + p_i) = (\sigma + p_i) - \mathrm{fl}(\sigma + p_i)$$

D'après le théorème 3.4, l'erreur d'arrondi $\mathrm{fl}(\sigma + p_i) - (\sigma + p_i)$ est un flottant pour toute opération d'arrondi d'une somme. Par conséquent, son opposé :

$$p_i - q_i = (\sigma + p_i) - \mathrm{fl}(\sigma + p_i) = -(\mathrm{fl}(\sigma + p_i) - (\sigma + p_i))$$

est également un flottant.

Ainsi, $p_i - q_i$ étant déjà un flottant, son arrondi est lui-même :

$$\mathrm{fl}(p_i - q_i) = p_i - q_i$$

Ce qui complète la preuve du lemme. □

Lemme 4.7 (Quantification des q_i)**.** *Pour tout $i \in \{1, \ldots, n\}$, q_i est un multiple exact de 2^s où $s = M + N - P + 1$.*

Démonstration. Par définition, $\sigma = \frac{3}{2} \times 2^{M+N}$ et $|p_i| \leq 2^M$.

Pour toute valeur de p_i dans l'intervalle $[-2^M, 2^M]$, nous avons :

$$\sigma - 2^M \leq \sigma + p_i \leq \sigma + 2^M$$

Comme $\sigma = \frac{3}{2} \times 2^{M+N}$ et $N \geq 1$ (par hypothèse), nous avons $2^M \ll \sigma$, d'où :

$$\frac{3}{2} \times 2^{M+N} - 2^M \leq \sigma + p_i \leq \frac{3}{2} \times 2^{M+N} + 2^M$$

Pour $N \geq 1$, cette inégalité implique :

$$2^{M+N} < \sigma + p_i < 2^{M+N+1}$$

Cela signifie que l'ufp (unit in the first place) de $\sigma + p_i$ est exactement 2^{M+N}.

Par conséquent, l'ufp de $\mathrm{fl}(\sigma + p_i)$ est également 2^{M+N}. Dans un format flottant avec P bits de précision, l'ulp (unit in the last place) de $\mathrm{fl}(\sigma + p_i)$ est exactement :

$$\mathrm{ulp}(\mathrm{fl}(\sigma + p_i)) = 2^{M+N} \times 2^{-(P-1)} = 2^{M+N-P+1} = 2^s$$

Cela signifie que $\mathrm{fl}(\sigma + p_i)$ est nécessairement un multiple exact de 2^s.

Par ailleurs, $\sigma = \frac{3}{2} \times 2^{M+N} = 3 \times 2^{M+N-1}$ est un multiple exact de 2^{M+N-1}. Comme $s = M+N-P+1$ et $P \geq 2$ pour tout format flottant standard, nous avons $s \leq M+N-1$, ce qui implique que σ est également un multiple exact de 2^s.

Par conséquent, $q_i = \mathrm{fl}(\sigma + p_i) - \sigma$ est la différence de deux multiples exacts de 2^s, donc q_i est lui-même un multiple exact de 2^s. □

Lemme 4.8 (Monotonie de l'approximation)**.** *Pour tout entier $x \geq s$ et pour tout $i \in \{1, \ldots, n\}$:*

1. *Si $p_i \leq 2^x$, alors $q_i \leq 2^x$*

2. Si $p_i \geq -2^x$, alors $q_i \geq -2^x$

Démonstration. Nous démontrons la propriété par contraposition.

Supposons qu'il existe $x \geq s$ tel que $q_i > 2^x$. Nous allons prouver que cela implique $p_i > 2^x$.

Par définition, $q_i = \text{fl}(\sigma + p_i) - \sigma$. Donc $q_i > 2^x$ implique $\text{fl}(\sigma + p_i) > \sigma + 2^x$.

Par la définition de l'arrondi au plus proche, si $\text{fl}(\sigma + p_i) > \sigma + 2^x$, cela signifie que $\sigma + p_i$ est plus proche de $\text{fl}(\sigma + p_i)$ que de $\sigma + 2^x$, ou en d'autres termes :

$$|\text{fl}(\sigma + p_i) - (\sigma + p_i)| < |(\sigma + 2^x) - (\sigma + p_i)| = |2^x - p_i|$$

Comme $\text{fl}(\sigma + p_i) > \sigma + 2^x > \sigma + p_i$ (car $2^x > 0$ et nous supposons $q_i > 2^x$), nous avons :

$$|\text{fl}(\sigma + p_i) - (\sigma + p_i)| < |2^x - p_i| = 2^x - p_i$$

où la dernière égalité suppose $p_i \leq 2^x$.

Cette inégalité implique :

$$\text{fl}(\sigma + p_i) - (\sigma + p_i) < 2^x - p_i$$

En réarrangeant :

$$\text{fl}(\sigma + p_i) - 2^x < \sigma + p_i - p_i = \sigma$$

Soit :

$$\text{fl}(\sigma + p_i) < \sigma + 2^x$$

Mais cela contredit notre hypothèse $\text{fl}(\sigma + p_i) > \sigma + 2^x$. Cette contradiction montre que l'hypothèse $p_i \leq 2^x$ est incompatible avec $q_i > 2^x$. Par contraposition, nous concluons :

$$\text{Si } q_i > 2^x \text{ alors } p_i > 2^x$$

Ou de manière équivalente :

$$\text{Si } p_i \leq 2^x \text{ alors } q_i \leq 2^x$$

La preuve du deuxième point est similaire, en considérant l'hypothèse $q_i < -2^x$ et en montrant qu'elle implique $p_i < -2^x$. □

Lemme 4.9 (Propriété d'approximation optimale)**.** *Pour tout $i \in \{1, \ldots, n\}$, q_i est le multiple de 2^s le plus proche de p_i parmi tous les réels.*

Démonstration. Soit r un multiple quelconque de 2^s (réel ou flottant). Nous devons prouver que $|q_i - p_i| \leq |r - p_i|$.

Cas 1 : r est un flottant représentable. Considérons $r' = r + \sigma$. Comme r est un multiple de 2^s et σ est un multiple de 2^s (d'après le lemme 4.7), r' est également un multiple de 2^s.

Si r' est un flottant représentable, alors par la propriété de l'arrondi au plus proche et la définition de q_i :

$$|\text{fl}(\sigma + p_i) - (\sigma + p_i)| \leq |r' - (\sigma + p_i)|$$

Ce qui est équivalent à :

$$|(\sigma + q_i) - (\sigma + p_i)| \leq |r' - (\sigma + p_i)|$$

Soit :

$$|q_i - p_i| \leq |r - p_i|$$

Cas 2 : r est un flottant mais r' n'est pas un flottant. Si r est un flottant multiple de 2^s mais $r + \sigma$ dépasse la capacité de représentation, alors $|r|$ doit être extrêmement grand.

D'après le lemme 4.8, pour tout entier $x \geq s$, si $|p_i| \leq 2^x$, alors $|q_i| \leq 2^x$.

Comme $|p_i| \leq 2^M$ par hypothèse et $M = s + P - N - 1$, nous avons $|q_i| \leq 2^{s+P-N-1} \leq 2^{s+P-2}$ pour $N \geq 1$.

Pour qu'un flottant r multiple de 2^s soit tel que $r + \sigma$ ne soit pas représentable, il faut que $|r| > 2^E - \sigma$, où E est l'exposant maximal. Or, $2^E - \sigma \gg 2^{s+P-2}$ pour tout format flottant standard.

Donc, pour un tel r :

$$|r - p_i| \geq |r| - |p_i| > (2^E - \sigma) - 2^M \gg |q_i - p_i|$$

Ce qui montre que r ne peut pas être plus proche de p_i que q_i.

Cas 3 : r n'est pas un flottant. Si r est un multiple de 2^s non représentable comme flottant, alors sa valeur absolue doit être au moins 2^E (où E est l'exposant maximal) ou sa représentation nécessite plus de P bits.

Or, $|p_i| \leq 2^M \ll 2^E$ et $|q_i| \leq 2^{s+P-2} \ll 2^E$.

Ainsi, pour un tel r :

$$|r - p_i| \geq |r| - |p_i| \geq 2^E - 2^M \gg |q_i - p_i|$$

Par conséquent, dans tous les cas, nous avons prouvé que :

$$|q_i - p_i| \leq |r - p_i|$$

pour tout multiple r de 2^s, ce qui démontre que q_i est le multiple de 2^s le plus proche de p_i. □

Lemme 4.10 (Borne sur les résidus)**.** *Pour tout $i \in \{1, \ldots, n\}$, on a $|p'_i| \leq 2^{s-1}$.*

Démonstration. D'après les lemmes 4.6 et 4.9, nous avons :

$$p'_i = \mathrm{fl}(p_i - q_i) = p_i - q_i$$

et q_i est le multiple de 2^s le plus proche de p_i.

La distance maximale entre un nombre quelconque et le multiple de 2^s le plus proche (lui-même inclus si c'en est un) est inférieur où égal à la moitié de la différence entre deux multiples de 2^s, soit $\frac{2^s}{2} = 2^{s-1}$. Par conséquent :

$$|p'_i| = |p_i - q_i| \leq 2^{s-1}$$

Ce qui prouve le lemme. □

Lemme 4.11 (Sommation exacte des parties hautes)**.** *Pour tout $i \in \{1, \ldots, n\}$ et avec τ_i la valeur de τ au début de l'itération i, on a :*

$$\mathrm{fl}(\tau_i + q_i) = \tau_i + q_i$$

Démonstration. Nous allons démontrer par récurrence que pour tout $i \in \{0, 1, \ldots, n\}$:

1. τ_i est un multiple exact de 2^s
2. $|\tau_i| \leq i \cdot 2^M$

Base : Pour $i = 0$, nous avons $\tau_0 = 0$ qui est trivialement un multiple de 2^s et vérifie $|\tau_0| = 0 \leq 0 \cdot 2^M$.

Hérédité : Supposons que l'hypothèse est vraie pour un certain i avec $0 \leq i < n$. Montrons qu'elle est vraie pour $i + 1$.

D'après le lemme 4.7, q_{i+1} est un multiple de 2^s. Par l'hypothèse de récurrence, τ_i est également un multiple de 2^s. Par conséquent, $\tau_i + q_{i+1}$ est un multiple de 2^s.

D'autre part :

$$|\tau_i + q_{i+1}| \leq |\tau_i| + |q_{i+1}| \leq i \cdot 2^M + |q_{i+1}|$$

D'après le lemme 4.8 et le fait que $|p_{i+1}| \leq 2^M$, nous avons $|q_{i+1}| \leq 2^M$. Donc :

$$|\tau_i + q_{i+1}| \leq i \cdot 2^M + 2^M = (i + 1) \cdot 2^M$$

Nous avons ainsi établi que $\tau_{i+1} = \tau_i + q_{i+1}$ est un multiple de 2^s et vérifie $|\tau_{i+1}| \leq (i + 1) \cdot 2^M$, ce qui complète l'hérédité.

Maintenant, démontrons que l'addition $\tau_i + q_i$ est calculée exactement :

$$|\tau_i + q_i| \leq n \cdot 2^M = 2^N \cdot 2^M = 2^{M+N}$$

Comme $\tau_i + q_i$ est un multiple de 2^s avec $s = M + N - P + 1$, ce nombre nécessite au plus $P - 1$ bits significatifs pour sa représentation :

$$\frac{2^{M+N}}{2^s} = \frac{2^{M+N}}{2^{M+N-P+1}} = 2^{P-1}$$

Par conséquent, $\tau_i + q_i$ est exactement représentable avec P bits de précision, et :

$$\mathrm{fl}(\tau_i + q_i) = \tau_i + q_i$$

Ce qui prouve le lemme. □

Lemme 4.12 (Décomposition additive)**.** *Pour tout $i \in \{1, \ldots, n\}$, on a :*

$$p_i = q_i + p'_i$$

Démonstration. D'après le lemme 4.6, on a :

$$p'_i = \mathrm{fl}(p_i - q_i) = p_i - q_i$$

Par réarrangement :

$$p_i = q_i + p'_i$$

Ce qui prouve le lemme. □

Preuve du Théorème 4.4. Grâce aux lemmes que nous avons établis, nous pouvons maintenant prouver les deux propriétés du théorème principal.

Propriété (i) : Le lemme 4.10 établit directement que $|p'_i| \leq 2^{s-1}$ pour tout $i \in \{1, \ldots, n\}$.

Propriété (ii) : D'après le lemme 4.12, nous avons $p_i = q_i + p'_i$ pour tout i. Par sommation :

$$\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} (q_i + p'_i) = \sum_{i=1}^{n} q_i + \sum_{i=1}^{n} p'_i$$

D'après le lemme 4.11 et par construction de l'algorithme, chaque addition dans la boucle est exacte, ce qui implique que $\tau = \sum_{i=1}^{n} q_i$ exactement. Par conséquent :

$$\sum_{i=1}^{n} p_i = \tau + \sum_{i=1}^{n} p'_i$$

Ce qui prouve la propriété (ii) et achève la démonstration du théorème. □

5 Implémentation

Après avoir établi les fondements théoriques de l'algorithme `ExtractVector`, nous présentons maintenant son implémentation en OCaml. Nous nous concentrerons sur les aspects essentiels du code, en expliquant les choix d'implémentation et leur relation avec la théorie développée précédemment.

5.1 Implémentation de l'algorithme

L'implémentation de l'algorithme `ExtractVector` comprend deux fonctions principales : une fonction auxiliaire `compute_sigma` qui calcule le paramètre σ optimal selon les propriétés du vecteur d'entrée, et la fonction principale `extract_vector` qui réalise l'extraction proprement dite.

Calcul du paramètre σ

```
(** Calcule la valeur de sigma appropriee pour l'algorithme
   ExtractVector *)
let compute_sigma p =
  let n = Array.length p in
  (* Calcul de M: tel que |p_i| ≤ 2^M *)
  let max_abs_p = Array.fold_left (fun acc x → max acc (abs_float
     x)) 0. p in
  let m = ceil (log (max_abs_p +. 1e-10) /. log 2.) in

  (* Calcul de N: tel que n ≤ 2^N *)
  let n_float = float_of_int n in
  let n_power = ceil (log n_float /. log 2.) in

  (* Sigma = 3/2 * 2^(M+N) comme defini dans la preuve *)
  1.5 *. (2. ** (m +. n_power))
```

La fonction `compute_sigma` détermine la valeur optimale du paramètre $\sigma = \frac{3}{2} \times 2^{M+N}$ en calculant :

— M : le plus petit entier tel que $|p_i| \leq 2^M$ pour tout i

— N : le plus petit entier tel que $n \leq 2^N$, où n est la taille du vecteur

En OCaml, l'opérateur `fold_left` réalise une réduction sur le tableau pour trouver la valeur absolue maximale, tandis que les fonctions `ceil` et `log` permettent de calculer la puissance de 2 minimale nécessaire. L'ajout d'une petite constante (10^{-10}) prévient les erreurs numériques potentielles lors du calcul logarithmique.

Implémentation de l'algorithme ExtractVector

```
(** Implementation de l'algorithme ExtractVector *)
let extract_vector p =
  let n = Array.length p in
  let sigma = compute_sigma p in

  let tau = ref 0. in
  let p_prime = Array.make n 0. in

  Array.iteri (fun i pi →
    let qi = (sigma +. pi) -. sigma in
    p_prime.(i) <- pi -. qi;
    tau := !tau +. qi
  ) p;

  (!tau, p_prime)
```

La fonction `extract_vector` implémente directement l'algorithme présenté dans la section 2. Notons quelques particularités propres à OCaml :

- `ref 0.` crée une référence mutable initialisée à zéro, permettant de modifier la valeur de τ durant l'itération
- `!tau` accède à la valeur contenue dans la référence
- `tau := !tau +. qi` met à jour la valeur de la référence
- `Array.iteri` applique la fonction anonyme à chaque élément du tableau avec son indice

L'implémentation suit fidèlement les étapes théoriques :

1. Calcul du paramètre σ optimal
2. Extraction de la partie haute q_i de chaque composante p_i via l'opération $((\sigma + p_i) - \sigma)$
3. Calcul de la partie basse $p'_i = p_i - q_i$
4. Sommation exacte des q_i pour obtenir τ

5.2 Validation expérimentale

Pour valider notre implémentation, nous avons réalisé une série de tests sur différents types de vecteurs présentant des caractéristiques numériques variées :

- **Vecteurs standards** : contenant des valeurs flottantes arbitraires
- **Vecteurs avec annulations partielles** : contenant des valeurs opposées différant légèrement, susceptibles de causer des pertes de précision
- **Vecteurs mal conditionnés** : présentant un grand écart entre les valeurs minimales et maximales
- **Séquence de Kahan** : cas classique conçu par Kahan [1] pour mettre en défaut les algorithmes de sommation traditionnels

— **Vecteurs avec différences d'échelle** : incluant à la fois des valeurs très grandes (10^{20}) et très petites (10^{-10})

Pour chaque vecteur de test, nous avons vérifié les deux propriétés fondamentales démontrées dans la section 4 :

1. $\sum_{i=1}^{n} p_i = \tau + \sum_{i=1}^{n} p'_i$ (conservation de la somme)
2. $|p'_i| \leq 2^{s-1}$ pour tout i (réduction de magnitude des composantes résiduelles)

Les résultats ont confirmé la validité de notre implémentation et des propriétés théoriques établies. En particulier, pour les vecteurs avec de grandes différences d'échelle, l'algorithme a réussi à préserver les composantes de faible magnitude qui auraient été perdues avec une sommation directe.

6 Conséquences

L'algorithme `ExtractVector` et les propriétés que nous avons démontrées entraînent plusieurs conséquences importantes tant sur le plan théorique que pratique.

Corollaire 6.1 (Décomposition exacte)**.** *L'algorithme `ExtractVector` permet de transformer un vecteur p en une somme $\tau + \sum p'_i$ où τ est une valeur flottante contenant la partie haute sommable exactement et p' est un vecteur contenant les parties résiduelles de faible magnitude.*

Cette décomposition présente plusieurs avantages significatifs :

1. **Sommation à précision contrôlée** : En appliquant itérativement l'algorithme sur le vecteur résiduel p', on peut atteindre une précision arbitraire. L'erreur d'arrondi à chaque itération est réduite d'un facteur d'environ 2^{P-N}, où P est le nombre de bits de précision du format flottant et N l'entier le plus petit tel que 2^N est plus grand que le nombre de flottants à sommer.
2. **Robustesse face aux annulations** : Comme démontré par nos tests sur le vecteur avec différences d'échelle, l'algorithme préserve les composantes de faible magnitude même en présence de grands nombres qui s'annulent. Cette propriété est cruciale dans les calculs d'algèbre linéaire impliquant des matrices mal conditionnées.
3. **Indépendance de l'ordre de sommation** : Contrairement aux algorithmes de sommation traditionnels, le résultat de `ExtractVector` est invariant par rapport à l'ordre des composantes du vecteur, ce qui garantit la reproductibilité des calculs numériques.
4. **Adaptation naturelle aux architectures parallèles** : La décomposition effectuée localement sur des sous-vecteurs peut être fusionnée sans perte de précision, rendant l'algorithme particulièrement adapté aux calculs distribués dans les systèmes HPC.

De plus, le lemme 4.9 garantit que l'information extraite à chaque étape est la meilleure possible étant donné les contraintes du format flottant, maximisant ainsi l'efficacité de la décomposition.

Dans le contexte des simulations quantiques, où la précision des amplitudes de probabilité est cruciale, cette méthode permet de réduire significativement l'accumulation d'erreurs d'arrondi, préservant ainsi l'intégrité des calculs d'états quantiques sur de longues séquences d'opérations.

7 Conclusion

Dans ce rapport, nous avons présenté une analyse mathématique rigoureuse de l'algorithme `ExtractVector`, démontrant formellement qu'il permet une décomposition exacte d'un vecteur flottant en une partie haute sommable exactement et un vecteur résiduel de faible magnitude. Nous avons établi les conditions précises sur le paramètre σ garantissant l'exactitude des opérations critiques, et proposé une implémentation efficace en OCaml validée par des tests sur divers vecteurs numériquement difficiles.

L'algorithme `ExtractVector` résout élégamment le problème d'accumulation d'erreurs d'arrondi dans les sommations flottantes, préservant la précision même avec des composantes de magnitudes très différentes. Pour son application dans les environnements HPC, la synchronisation de la valeur du paramètre σ entre tous les cœurs est essentielle pour assurer la précision et la reproductibilité des calculs parallèles. La technique "1-Reduction" de Demmel et Nguyen [4] pourrait être adaptée pour déterminer une valeur σ commune ou standardisée compatible avec l'architecture distribuée.

Plusieurs perspectives s'ouvrent pour des travaux futurs, notamment : l'extension aux opérations matricielles, l'optimisation pour architectures parallèles, l'application aux simulations quantiques nécessitant une précision rigoureuse, et le développement d'une bibliothèque d'arithmétique à précision multiple fondée sur ces principes d'extraction et d'accumulation exacte.

En définitive, l'algorithme `ExtractVector` constitue une avancée significative dans l'arithmétique flottante de précision, permettant des calculs numériques plus fiables dans des domaines scientifiques exigeants tout en restant compatible avec les architectures de calcul modernes.

Références

[1] Kahan, W. (1965). Further remarks on reducing truncation errors. Communications of the ACM, 8(1), 40.

[2] IEEE Standard for Floating-Point Arithmetic. (2008). IEEE Std 754-2008, 1-70.

[3] Rump, S. M., Ogita, T., & Oishi, S. (2008). Fast High Precision Summation. (soumis le 27 juillet 2008).

[4] Demmel, J., & Nguyen, H. D. (2013). Parallel Reproducible Summation. IEEE Transactions on Computers. (soumis le 27 juillet 2008).

[5] Manuel de Référence sur l'Arithmétique à Virgule Flottante. (2023). Chapitres 1 et 2 : Introduction et Définitions de Base.