Conversion Automatique des Codes du Flottant au Fixe

Jean Tedros, Kakpto Yehossou, Romain Laudier, Mlaraha Hassani

January 17, 2021

Université de Versailles Saint-Quentin-en-Yvelines
ISTY - Master Calcul Haute Performance, Simulation
PPN

Encadrant Dr. Matei Istoan

Contents

List of Figures

1 Introduction et Aperçu historique

Représenter les nombres réels pour l'arithmétique est nécessaire pour le calcul, néanmoins le choix de la représentation n'est pas évident. L'émulation d'un ensemble continu infini par un ensemble fini se fait au prix de la précision, la portée et la facilité d'utilisation. Le développement historique des représentations des nombres est une histoire fascinante. Les mathématiciens babyloniens ont largement utilisé une notation sexagésimale développée dès 1750 AV. La notation était particulièrement pratique pour la multiplication et la division, car l'alignement des points de base n'avait aucun effet sur la réponse. Cette notation était unique en ce qu'elle était en fait une forme de représentation à virgule flottante avec omission d'exposants, parsuite tous les nombres dont le rapport est une puissance de 60 partagent la même représentation. Notre notation décimale, qui diffère des formes les plus anciennes principalement de son point de base fixe, ainsi que de son symbole pour le nombre zéro, a été développée pour la première fois en Inde dans la culture hindy vers l'an 600 JC. Les principes hindous de l'arithmétique décimale ont été amenés en Perse vers 750 après JC. Peu de temps après, al-Khawarizmi a écrit son manuel d'Arabe sur le sujet, qui a été traduit en Latin et a eu une forte influence sur Leonardo Pisano de Fibonacci dont le livre d'arithmétique a joué un rôle majeur dans la diffusion des chiffres Hindous-Arabes en Europe en AD 1202. .

La notation décimale n'était d'abord appliquée qu'aux nombres entiers, pas aux fractions. Les astronomes Arabes, qui avaient besoin de fractions dans leurs cartes stellaires et autres tableaux, ont continué à utiliser la notation de Ptolémée, une notation basée sur les unités de degrés, minutes et secondes basée sur la notation sexagésimale babylonienne. Le premier exemple connu de fractions décimales en Europe se trouve dans un texte du XVe siècle de Simon Stevin de Belgique. Le fait que tout entier supérieur à 1 puisse servir de base a été d'abord énoncé par Blaise Pascal dans Numeric Multiplicibus en 1685, où il a déclaré la nécessité d'un système duodécimal. La première apparition connue de la notation binaire remonte à 1605 par Thomas Harriot, où il définit les opérations de décalage sur les nombres binaires. Finalement, un article de G. W. Leibniz illustre l'addition, la soustraction, la multiplication et la division binaires; son article était la naissance de l'arithmétique radix-2. Une grande partie de l'histoire récente des systèmes numériques est liée au développement des machines à calcule. L'intérêt accru pour les dispositifs mécaniques pour l'arithmétique, en particulier pour la multiplication, a conduit plusieurs personnes dans les années 1930 à envisager le système binaire. Néanmoins, les premiers ordinateurs à grande vitesse Américains, construits au début des années 40, utilisaient l'arithmétique décimale. Mais en 1946, un mémorandum important d'AW Burks, HH Goldstine et J. von Neumann, à propos de la conception du premier ordinateur à programme stocké, a donné des raisons détaillées pour s'éloigner de la tradition radiacale et utiliser la notation en base deux. Le monde du calcul numérique a beaucoup changé en 1985, lorsque la norme IEEE 754-1985 pour l'arithmétique binaire à virgule flottante a été publiée.

2 Représentation en Virgule Fixe

Presque tous les ordinateurs modernes destinés au calcul scientifique incluent l'arithmétique à virgule flottante dans le cadre du répertoire d'opérations intégrées compatibles avec IEEE 754-1985. Cette norme spécifie différents formats, le comportement des opérations de base et des conversions, et des conditions exceptionnelles [1]. Cela s'est traduit par des améliorations significatives

en termes de précision, de fiabilité et de portabilité des logiciels numériques. Cependant, il y a une surcharge de processeur importante requise pour effectuer des calculs en virgule flottante résultant du manque de prise en charge matérielle de la virgule flottante. Les calculs peuvent être effectués à l'aide de représentations en virgule fixe signées en complément à deux.

Les processeurs modernes intègrent une unité d'exécution appelée ALU (unité logique arithmétique) qui effectue des opérations élémentaires sur les entiers, y compris +, - et, ×. La haute efficacité de l'arithmétique entière la rend idéale pour les calculs symboliques et pour la manipulation de valeurs énumérables. En pratique, la plupart des ALU implémentent des opérations sur les entiers à l'aide de l'arithmétique modulaire [4]. Dans cette arithmétique, les opérations sont effectuées modulo N et émulent les entiers tant que les valeurs calculées sont inférieures á N. Sur les machines binaires, choisir N comme puissance de 2 accélère les opérations élémentaires et réduit le coût matériel. Par exemple, de nombreux microcontrôleurs bas de gamme utilisent $N = 2^8$ word-length, alors que sur les ordinateurs de bureau $N = 2^{16}$, 2^{32}, 2^{64} word-lengths correspondant au types de variables (char, int, long, int) de C/C^{++} couramment implémentés dans les compilateurs.

Contrairement à la virgule flottante, un nombre à virgule fixe a un point décimal virtuel de position fixe. Contrairement à l'exposant qui est codé dans une représentation numérique à virgule flottante, le facteur d'échelle en virgule fixe n'est pas codé dans les données du programme, il est donc dit virtuel. Sa valeur n'est connue que du programmeur qui écrit le logiciel. Une conséquence de cette nature implicite est que chaque détail arithmétique, y compris les alignements et la prévention des débordements, doit être géré de manière statique et fourni par le programmeur. Ce n'est que lorsqu'il est nécessaire de récupérer un résultat, de l'interpréter ou de l'afficher à un utilisateur que le programmeur est obligé d'appliquer le facteur de mise à l'échelle. Chaque nombre est représenté par un nombre de bits pour la partie entière et pour la partie fractionnaire, ainsi que par un bit pour le signe. Dans la notation Q, un nombre à virgule fixe est dit format $Q_{IWL,FWL}$ avec les parties entières IWL et fractionnaires IWF. Le nombre de bits attribués à la partie entière est appelé longueur de mot entier, et celui attribué á la fraction est la longueur de mot fractionnaire et 1 bit pour le signe [3]. Ainsi, la longueur de mot (WL) correspond á IWL + FWL + 1. Une représentation en virgule fixe est caractérisée par le set le range nombres exprimables, définie comme la distance entre le plus grand et le plus petit nombre représentable, et le pas de précision ou de quantification, défini comme la distance entre deux nombres adjacents. Le range (R) et le pas de quantification (Q) dépendent respectivement de l'IWL et du FWL: $-2^{IWL} \leq R \leq 2^{IWL}$ et $Q = 2^{-FWL}$.

Le nom virgule fixe est dû au fait que le facteur d'échelle d'une variable à virgule fixe ne change pas au moment de l'exécution. Cela contraste avec les variables à virgule flottante oú l'exposant croît et se rétrécit dynamiquement de manière à garder une mantisse normalisée. Une conséquence de la nature dynamique de l'exposant est que les nombres à virgule flottante ne sont pas séparés de manière équidistante. En effet, l'écart entre un nombre binaire à virgule flottante x et le prochain nombre binaire à virgule flottante plus grand que x est donné par $ulp(x) = \epsilon \cdot 2^E$ o ϵ est une constante qui dépend du word-lenght de la mantisse et E est l'exposant de x [3]. Par comparaison, les nombres à virgule fixe sont séparés de manière équidistante par un pas de 2^{-FWL}.

Avec l'introduction du bit de signe, la représentation la plus couramment utilisée pour les nombres signés est le complément à deux. Contrairement au bit signé dans la représentation à virgule flottante, la représentation du complément à deux comprend un bit de signe implicite. Le complément

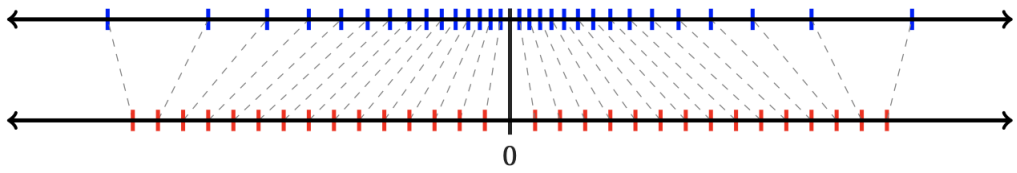

Figure 1: Distributions typiques des nombres á virgule flottante (en haut) et á virgule fixe (en bas)

à deux est formé en flippant tous les bits du nombre, puis en ajoutant 1 tout en supprimant tout report du bit le plus significatif. Puisque tous les modèles de bits sont valides, il y a 256 nombres différents qui peuvent être représentés, contrairement à la représentation one-complement avec un total de 255 représentations valides en raison du zéro ayant deux représentations.

2.1 Arithmétque

Les opérations arithmétiques de base sur les nombres à virgule fixe, l'addition, la soustraction, la multiplication et la division, peuvent être mises en uvre en utilisant des opérations arithmétiques ordinaires sur les nombres entiers et le décalage de bits. L'addition et la soustraction sont effectuées par arithmétique entière, à condition que le FWL des deux opérandes soit identique. Si les FWL sont différents, alors l'opérande avec le FWL le plus bas doit être décalé juste avant l'addition, c'est-à-dire avant d'effectuer l'addition ou la soustraction d'un nombre 8,24 et d'un nombre 16,16, le premier nombre à décaler vers la droite de 8 places. Une multiplication de deux nombres IWL1.FWL1 et IWL2.FWL2 se traduira par un nombre avec la représentation (IWL1+IWL2).(FWL1+FWL2); évidemment ce nombre ne rentrera pas dans le format WL choisi et donc à la fin les bits appropriés doivent être découpés. Lorsqu'un produit de nombres à virgule fixe est calculé, au moins une des valeurs doit être étendue de signe à $2 \times$ WL pour faire la multiplication correctement, sinon seule la moitié inférieure du résultat sera renvoyée. En pratique, cela se fait en convertissant le multiplicande en un type de variable $2 \times$ WL. L'arrondi est nécessaire à la fois après une multiplication, ou avant l'addition et la soustraction de valeurs avec différents FWL.

2.1.1 Addition et Soustraction

Nous considérons sans perte de généralité que les deux opérandes sont alignés dans un format Q donné. En fonction de la valeur, la sortie nécessite l'un des formats suivants:

1. $Q_{i,f}$, si output $\subseteq$ range($Q_{i,f}$).

2. $Q_{i+1,f-1}$, si output $\nsubseteq$ range($Q_{i,f}$), overflow.

Le cas (1) se produit lorsque la somme rentre dans $range(Q_{i,f})$. Le cas (2) se produit lorsque range $(Q_{i,f}) \subseteq$ output $\subseteq$ range $(Q_{i+1,f-1})$. Dans ce cas, la somme ou la soustraction des deux variables provoque un débordement car le résultat est potentiellement trop grand pour tenir dans une variable $Q_{i,f}$. La solution consiste à aligner les deux opérandes avant d'effectuer l'addition ou la soustraction. C'est-à-dire que l'addition ou la soustraction est transformée en une opération entre deux variables au format $Q_{i+1,f-1}$ au moyen d'un décalage vers la droite.

2.1.2 Multiplication

Contrairement à l'addition et à la soustraction, il n'y a aucune contrainte sur les formats des multiplicandes. Le résultat dans une variable à virgule fixe au format $Q_{i,f}$ o $i = i1 + i2$ et $f = f1 + f2$ où i et f sont les IWL et FWL correspondants des opérandes. La multiplication par une puissance de 2 est un cas particulier intéressant puisqu'elle peut être transformée en un décalage virtuel consistant à changer le format en virgule fixe de l'opérande. Par conséquent, la multiplication par une puissance de 2 est gratuite et n'a aucun impact sur la latence d'évaluation du programme. Quant au format de sortie, la multiplication par 2^p aboutit à une variable à virgule fixe au format $Q_{i,f}$ avec $i = i_1 + p$ et $f = f_1 - p$ o i_1 et f_i sont les IWL et FWL des premiers opérandes.

3 Représentation en virgule flottante

Pour de nombreuses raisons, il est considérablement plus pratique de laisser la position du point de base être dynamiquement variable ou "flottante" pendant qu'un programme est en cours d'exécution, et de porter avec chaque nombre une indication de sa position actuelle de point de base. Un nombre à virgule flottante standard est représenté par 3 valeurs: un signe s, un exposant e, et un significande m. Ses valeurs sont données par

$$x = (-1)^s \cdot m \cdot \beta^e$$

La norme spécifie différentes contraintes sur s, e et m pour $\beta \in 2, 10$: Le signe s $\in 0, 1$ est une valeur binaire. Le nombre est positif si son signe est zéro et négatif dans le cas contraire. L'exposant e $\in e_{min}, e_{max}$ où e_{min} et e_{max} sont spécifiés pour chaque format standardisé. le signifiant $m = m_0 \cdot m_1 \cdot m_2 \cdot ... \cdot m_{p-1}$. Le résultat exact des opérations arithmétiques telles que l'addition et la multiplication sur des nombres à virgule flottante ne sont pas nécessairement des nombres à virgule flottante [4]. Pour faire face à cela, la norme spécifie quatre modes d'arrondi. Ces modes d'arrondi, arrondi vers zéro, arrondi vers plus et moins l'infini, et arrondi au plus proche même spécifient de manière unique le nombre à virgule flottante à renvoyer lorsque le résultat d'une opération n'est pas un nombre à virgule flottante.

Standard name	**Common name**	**Bits distribution (sign, exponent, mantissa)**	**Range of e $[e_{min}, e_{max}]$**
Binary32	Single precision	(1, 8, 23)	[−126, 127]
Binary64	Double precision	(1, 11, 52)	[−1022, 1023]
Binary128	Quadruple precision	(1, 15, 112)	[−16382, 16383]

Figure 2: Spécifications standard pour 3 formats binaires IEEE-754

3.1 Conversion

La conversion entre les nombres à virgule fixe et flottante est simple. En général, la conversion d'un nombre à virgule flottante, a, au format à virgule fixe est effectuée en déplaant d'abord à

gauche le nombre par le FWL cible, en arrondissant au nombre entier le plus proche, puis en convertissant le résultat en une variable entière.

4 Conversion de virgule flottante en virgule fixe

Sans outils appropriés, la programmation en virgule fixe est sujet aux erreurs. Plus de 50 pourcent du temps de conception est consacré à la cartographie manuelle du prototype à virgule flottante dans un programme à virgule fixe. Le développement de programmes à virgule fixe est considéré comme fastidieux et difficile car il nécessite une mise à l'échelle appropriée pour chaque donnée et difficile car il nécessite mise à l'échelle appropriée pour chaque déplacement de données et opération arithmétique pour éviter les débordements tout en maintenant la précision. Ce qui nous intéresse, ce sont les méthodologies basées sur la transformation du code source. L'outil prend en entrée le code source d'un programme en virgule flottante et procède soit en simulant le code, soit en interpolation pour déterminer les formats en virgule fixe qui doivent tre affectés à chaque variable du calcul. à la fin de cette procédure, un nouveau code est sorti qui ne contient aucune référence aux types de données à virgule flottante. Les outils qui automatisent ce processus, souvent appelés outils float-to-fix, utilisent des techniques de compilation qui incluent l'analyse, l'annotation des représentations intermédiaires et la génération de code [2].

Comme indiqué précédemment, un format Q doit tre choisi pour être le format de la conversion résultante qui à son tour détermine à la fois le range et la précision de la variable convertie pendant toute la durée d'exécution du code. Certaines techniques anciennes suggèrent un cadre basé sur des informations statistiques déduites de simulations en virgule flottante. D'autres suggèrent d'annoter le code source par le programmeur, étant donné que les simulations ne donnent qu'une clôture de la plage des variables et peuvent donc manquer des cas d'angle qui apparaissent rarement lors des simulations. La seconde technique n'est pas entièrement automatisée en ce sens que le programmeur doit annoter les plages et la précision pour chacune des variables. Cependant, comme il a une connaissance plus approfondie du problème mathématique, il s'appuie généralement sur des méthodes analytiques plutôt que sur de longues simulations. En tant que tel, il peut fournir des limites d'erreur strictes et un temps de synthèse plus rapide. Ces techniques comprennent à la fois la détermination IWL et FWL, c'est-à-dire des analyses de range et de précision.

4.1 Verificarlo

Verificarlo est un compilateur basé sur LLVM qui remplace automatiquement les opérations en virgule flottante par leur équivalent dans un autre format. Cette transformation se fait généralement sur trois sections: le front-end qui transforme le code source en un autre intermédiaire, puis le middle-end qui demande des optimisations, et enfin le back-end qui effectue les modifications.

Verificarlo utilise le frontal de clang qui prend en charge de nombreux langages C/C^{++} et FORTRAN représentant la majorité des langages utilisés dans les codes HPC qui sont ensuite transformés en une représentation intermédiaire en utilisant clang. Après la traduction initiale, toutes les opérations arithmétiques à virgule flottante $+, -, \times$ sont identifiées et abstraites dans des structures qui sont ensuite utilisées par le backend pour charger toute autre fonction équivalente.

Une fois le programme compilé avec Verificarlo, il peut tre instrumenté avec différents backends en virgule flottante. En fait, il existe 6 backends implémentant chacun un type différent d'arithmétique en virgule flottante. L'objectif est d'écrire un nouveau backend pour la conversion du flottant vers la représentation fixe.

References

[1] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

[2] Y. Chatelain. *Outils de débogage et d'optimisation des calculs flottants dans un contexte HPC*. Theses, Université Paris-Saclay, Université Versailles Saint-Quentin-en-Yvelines, Dec. 2019.

[3] E. Oberstar. Fixed-point representation and fractional math (this is the old previous copy see updated link in abstract). 1, 01 2007.

[4] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.