

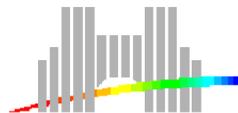
Projet Arénaire

Arithmétique des ordinateurs

CNRS - ENSL - INRIA - LIP - UCBL

Gilles Villard

Visite scientifique de J. Vuillemin - 7 mars 2006



Plan

I - Thématiques générales

Exemples de réalisation

II.1 - Évaluation de fonctions en matériel

II.2 - Efficacité et fiabilité en calcul flottant

II.3 - Complexité algorithmique en algèbre linéaire

III- Perspectives

Plan

I - Thématiques générales

Exemples de réalisation

II.1 - Évaluation de fonctions en matériel

II.2 - Efficacité et fiabilité en calcul flottant

II.3 - Complexité algorithmique en algèbre linéaire

III- Perspectives

15 personnes + S. Torres (30%) + S. Boyer (20%)

Permanents :

N. Brisebarre (U. St-Étienne)
F. de Dinechin (ENSL)
C.-P. Jeannerod (INRIA)
J.-M. Muller (CNRS)
N. Revol (INRIA)
G. Villard (CNRS)

Départs 2005 :

M. Daumas (CNRS, Montpellier-Perpignan)
A. Tisserand (CNRS, Montpellier)
J.-L. Beuchat (FNSRS 2001-2005)

Postdoc : I. Toli (INRIA)

Ingénieur associé : E. Bechetoille (INRIA)

Doctorants : F. Cháves, J. Detrey, C. Lauter, G. Melquiond, R. Michard,
S.K. Raina, N. Veyrat-Charvillon

Opérateurs arithmétiques : algorithmes et propriétés

- **Briques de base** : systèmes de représentation, $+$, $-$, \times , \div , $\sqrt{\quad}$, \cdot/cst , $\cdot \times cst$, entiers, nombres réels et complexes, intervalles, multi-précision, corps finis
- **Fonctions algébriques, élémentaires et spéciales** : $\sqrt[3]{\quad}$, \log , \cos , erf , . . .
- **Opérateurs composites** : $1/\sqrt{x^2 + y^2}$, . . .
- **Spécifiques** : filtres FIR, transformées DCT, opérateurs DSP, cryptographie
- **Propriétés** des opérateurs : représentabilité de résultats, d'erreurs, . . .
- Actions de **normalisation** : IEEE 754, C++

Calculer mieux



Performances

surface, mémoire
vitesse
énergie
testabilité



Qualité

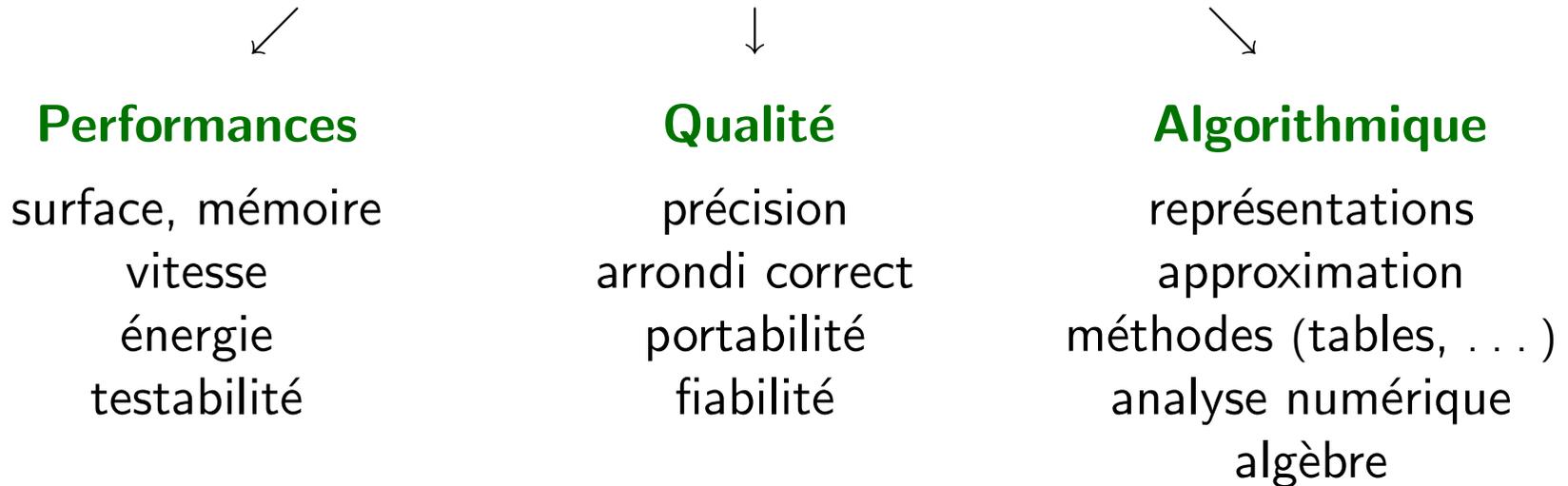
précision
arrondi correct
portabilité
fiabilité



Algorithmique

représentations
approximation
méthodes (tables, . . .)
analyse numérique
algèbre

Calculer mieux



⇒ Outils de conception et méthodologie

Approximation de fonctions : polynomiale, rationnelle, coefficients contraints

Qualité et automatisation : approximation, estimation d'erreur, fiabilité

Preuve formelle : preuve de théorèmes, assistants à la preuve

↪ Utilisation des arithmétiques

Complexité algorithmique : calcul flottant, algébrique/binaire

Algèbre linéaire : matrices polynomiales, calcul mathématique

Optimisation globale : intervalles, réduction de l'intervalle de recherche

Cryptographie : clef publique, signature numérique

Traitement du signal, multimédia

↪ Utilisation des arithmétiques

Complexité algorithmique : calcul flottant, algébrique/binaire

Algèbre linéaire : matrices polynomiales, calcul mathématique

Optimisation globale : intervalles, réduction de l'intervalle de recherche

Cryptographie : clef publique, signature numérique

Traitement du signal, multimédia

Supports d'implantation

Matériel : ASIC, FPGA, asynchrone, basse consommation (Verilog, VHDL . . .)

Couches de base : processeurs dédiés, bibliothèques optimisées, compilation

Logiciel : bibliothèques d'opérateurs, programmes (C, C++, Coq, PVS, Maple . . .)

<http://www.ens-lyon.fr/LIP/Arenaire/Ware>

Plan

I - Thématiques générales

Exemples de réalisation

II.1 - Évaluation de fonctions en matériel

II.2 - Efficacité et fiabilité en calcul flottant

II.3 - Complexité algorithmique en algèbre linéaire

III- Perspectives

Hardware Function Evaluation

Projet Arénaire
J. Detrey, F. de Dinechin, R. Michard, A. Tisserand, N. Veyrat-Charvillon

Context

General methods for hardware function evaluation

- Target functions
 - Boolean functions (non-approximated functions)
 - Algebraic functions (e.g., polynomial, T^2)
 - Composed functions (e.g., $T^2 \circ T$)
- Hardware-related algorithms
 - Fast multipliers, adders, subtractors
 - Optimizing error-correcting codes and encoders
 - Block codes (LFSR and PRNG)
 - Applications to digital signal processing algorithms

The precision/performance tradeoff

- Manage approximation errors
 - Fast and slow approximation methods
 - Fast and slow approximation methods
 - Fast and slow approximation methods
- Manage rounding errors
 - Fast and slow approximation methods
 - Fast and slow approximation methods
- Detailed error computation, a priori and a posteriori

Paper: note on function generation

- Define the architectural space
 - Define the set of parameters and constraints
 - Look for the right framework for new hardware solutions
- Navigate in it
 - Optimize via either constraint relaxation, heuristic, SAT
 - Use the new solution framework on the constraints and target

Best Polynomial Approximations with Machine Coefficients

Problems: what is the best polynomial to approximate a function?

- Minimize the error (maximize the number of approximation errors)
- Minimize the coefficient and operations in the approximation

Solution: **LEPFA** [22] finds the best polynomial with constraints on the coefficients

- Procedure: solve with linear solver, then fix coefficients and solve
- Minimize the error (maximize the number of approximation errors)
- Minimize the coefficient and operations in the approximation

Example: square coefficient polynomial approximation of $\sin(x)$

Constraints: square coefficient, fixed degree, fixed cost

Linear solver: $\min_{a_0, a_1, a_2} \max_{x \in [-\pi/2, \pi/2]} |\sin(x) - (a_0 + a_1x + a_2x^2)|$

Result: $a_0 = 0.7071067811865475244, a_1 = 0.7071067811865475244, a_2 = 0$

Table-and-Addition Methods

A family of **multipliers** allow approximation methods

- Reduce table sizes by reducing number of input bits
- Reduce the number of adders
- Reduce the number of adders

The generalized multiplier method is optimal among other methods [22]

Higher Order Table-Based Method

Recursive polynomial approximation

- Large precision and low error rates
- Fast architecture
- Reduced hardware cost
- Fast multipliers

Philosophy: fewer more accurate than needed

Fastest architecture

Reduced hardware cost

Applicable up to 32 bits

Small Polynomial Approximation without Multiplier

Goal: replace multipliers by a small number of additions

- Determination of the target polynomial approximation
- Step 1: generate table of the coefficients, add, division table
- Step 2: generate table of the coefficients, add, division table
- Step 3: fix the table of the coefficients to improve accuracy

Example: sine function

Target: $\sin(x)$, $x \in [-\pi/2, \pi/2]$

Example polynomial: $\sin(x) \approx a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$

Constraints: add, subtract, multiply, divide, shift, rotate

Example: $a_0 = 0.7071067811865475244, a_1 = 0.7071067811865475244, a_2 = 0$

Table-and-Addition Methods

Efficient hardware using function-based on tables and adders (and small tables, even 1-bit)

- Fast architecture
- Reduced hardware cost
- Fast multipliers

Example: $\sin(x) \approx a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$

Constraints: add, subtract, multiply, divide, shift, rotate

Example: $a_0 = 0.7071067811865475244, a_1 = 0.7071067811865475244, a_2 = 0$

Floating Point Elementary Functions for FPGAs

Objectives

- Fast and accurate approximation
- Fast architecture
- Reduced hardware cost
- Fast multipliers

Example: logarithmic function

Target: $\ln(x)$, $x \in [0.5, 2]$

Example polynomial: $\ln(x) \approx a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$

Constraints: add, subtract, multiply, divide, shift, rotate

Example: $a_0 = 0.6931471805599453092, a_1 = 0.6931471805599453092, a_2 = 0$

Selected publications

[22] Michard, A., de Dinechin, F., Tisserand, A., and Veyrat-Charvillon, N. "Fast polynomial approximation for hardware implementation." In *International Conference on Design, Systems and Control*. IEEE, November 2016.

[23] Michard, A., de Dinechin, F., Tisserand, A., and Veyrat-Charvillon, N. "Fast polynomial approximation for hardware implementation." *ACM Transactions on Embedded Computing Systems*, 2016.

[24] Detrey, J., de Dinechin, F., and Tisserand, A. "Fast polynomial approximation for hardware implementation." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[25] Detrey, J., de Dinechin, F., and Tisserand, A. "Fast polynomial approximation for hardware implementation." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[26] Detrey, J., de Dinechin, F., and Tisserand, A. "Fast polynomial approximation for hardware implementation." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[27] Detrey, J., de Dinechin, F., and Tisserand, A. "Fast polynomial approximation for hardware implementation." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[28] Michard, A., de Dinechin, F., Tisserand, A., and Veyrat-Charvillon, N. "Fast polynomial approximation for hardware implementation." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[29] Michard, A., de Dinechin, F., Tisserand, A., and Veyrat-Charvillon, N. "Fast polynomial approximation for hardware implementation." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[30] Michard, A., de Dinechin, F., Tisserand, A., and Veyrat-Charvillon, N. "Fast polynomial approximation for hardware implementation." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

Fonctions algébriques et élémentaires

- génération automatique d'architectures
- gestion et estimation des erreurs d'approximation et d'arrondi
- méthodes pour ASIC et FPGA
- applications en traitement du signal

Expertise et résultats :

Compromis précision/performance
Approximation polynomiale économique/meilleure
Tables multi-partites
Additions/décalages, additions/tables
Fonctions élémentaires en flottant

Plan

I - Thématiques générales

Exemples de réalisation

II.1 - Évaluation de fonctions en matériel

II.2 - Efficacité et fiabilité en calcul flottant

II.3 - Complexité algorithmique en algèbre linéaire

III- Perspectives

Calculs en nombres flottants

Réduire les **temps d'exécution**

Maîtriser la **précision**

⇐ Assurer la **fiabilité**

Accroître la **portabilité**

Participer à la **normalisation**

Calculs en nombres flottants

Réduire les **temps d'exécution**

Maîtriser la **précision**

⇐ Assurer la **fiabilité**

Accroître la **portabilité**

Participer à la **normalisation**



Bibliothèques, algorithmique sophistiquée

Approximations et erreurs d'arrondi

Outils d'aide à la conception et à la validation

↳ **Bibliothèque flip**

[Jeannerod, Raina, Tisserand, Rhône-Alpes/STMicroelectronics]

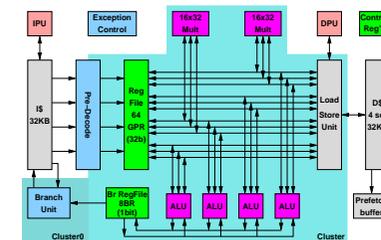
Couche flottante pour processeurs entiers/virgule fixe

Opérateurs arithmétiques de base

\pm , \times , \div , $\sqrt{\quad}$, fma , $1/x$, $1/\sqrt{\quad}$

Adéquation algorithmes/architecture

Processeur embarqué ST200
multimédia, traitement du signal
sans support flottant matériel



Arénaire - Efficacité et fiabilité du calcul flottant

↳ **Bibliothèque flip**

[Jeannerod, Raina, Tisserand, Rhône-Alpes/STMicroelectronics]

Couche flottante pour processeurs entiers/virgule fixe

Opérateurs arithmétiques de base

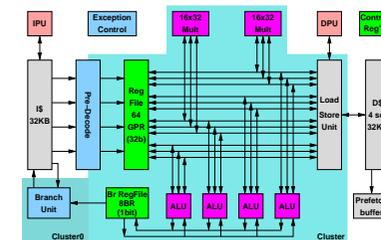
\pm , \times , \div , $\sqrt{\quad}$, fma , $1/x$, $1/\sqrt{\quad}$

Adéquation algorithmes/architecture



gain en vitesse d'un facteur 1.25 à 4

Processeur embarqué ST200
multimédia, traitement du signal
sans support flottant matériel



Répercussions :

- diffusion de l'arithmétique flottante dans le monde de l'embarqué
- architectures et style de développement simples et peu coûteux
- la question du portage en virgule fixe d'un code flottant ne se pose plus

État courant : C LGPL, intégrée dans les compilateurs de production des ST200

Futur : extension à d'autres cibles (DSP TI, Motorola, Analog Device, . . .) [Bechetoille]

↳ **Bibliothèque `crlibm`**

[de Dinechin, Defour, Lauter, Loirat, Muller]

Arrondi correct pour fonctions élémentaires

Peut nécessiter plus de bits que les 53 de la double précision IEEE-754

ex : $\cos(0.83748295564781876292227949) \approx 0.7429607293621962$ **0000000032**

⇒ représentait un obstacle sérieux à la performance

Absence de spécification ⇒ erreurs de quelques fois à plusieurs centaines de fois la précision avec les bibliothèques mathématiques courantes [`acos` Pentium 4/`libc`, `cosh` `libsunmath`]

↳ **Bibliothèque crlibm**

[de Dinechin, Defour, Lauter, Loirat, Muller]

Arrondi correct pour fonctions élémentaires

Peut nécessiter plus de bits que les 53 de la double précision IEEE-754

ex : $\cos(0.83748295564781876292227949) \approx 0.7429607293621962$ **0000000032**

⇒ représentait un obstacle sérieux à la performance

Absence de spécification ⇒ erreurs de quelques fois à plusieurs centaines de fois la précision avec les bibliothèques mathématiques courantes [acos Pentium 4/libc, cosh libsunmath]

Équipe Arénaire 1997-2006 :

analyse théorique de la précision requise
calculs exhaustifs de pires cas
précision adaptative
réduction d'argument

méthodes à base de tables
optimisation de caches
précision double étendue, fma
techniques pour l'arrondi correct

Répercussions : efficacité, transfert, normalisation

Quatre modes d'arrondis corrects

11 fonctions

	crlibm/libm	
	moyenne	max
log, Pentium 4 DLM05	1.05	0.57
exp, Itanium 2 Zimmermann 05	1.06	1.81

$\max \text{libultim} \approx 100 \times \max \text{crlibm}$

Utilisateurs

CERN/LHC@home : reproductibilité des simulations
plus de 36000 machines
processeurs AMD et Intel (Windows, Linux)

Objectif à moyen terme : GNU glibc

L'arrondi correct de certaines fonctions élémentaires est inclus dans l'ébauche actuelle de la révision de la norme IEEE 754 (été 2005)

Mise en œuvre algorithmique

approximation automatisée de fonctions

[Brisebarre, Muller, Tisserand, Torres]

Aide à la conception et fiabilité

estimation automatique des erreurs d'arrondi

preuve formelle

[Boldo, Daumas, Chávez, Melquiond, Revol]

Plan

I - Thématiques générales

Exemples de réalisation

II.1 - Évaluation de fonctions en matériel

II.2 - Efficacité et fiabilité en calcul flottant

II.3 - Complexité algorithmique en algèbre linéaire

III- Perspectives

Matrices polynomiales, entières et par intervalles

[Jeannerod, Revol, Villard]

Comparaison modèle algébrique/modèle binaire

Complexité du calcul certifié

- Réductions au produit de matrices autres que sur un corps abstrait
- Algorithme quasi-optimal d'inversion d'un polynôme matriciel
- Meilleure complexité pour le déterminant sans division
- Algorithmique exacte pour les très grandes matrices creuses

Applications et Bibliothèques :

- Contrôle linéaire, optimisation globale, calcul mathématique
- Programmation générique, par composants
- MPFI : <http://perso.ens-lyon.fr/nathalie.revol/software.html>
- LinBox : <http://www.ens-lyon.fr/LIP/Arenaire/Ware>

Plan

I - Thématiques générales

Exemples de réalisation

II.1 - Évaluation de fonctions en matériel

II.2 - Efficacité et fiabilité en calcul flottant

II.3 - Complexité algorithmique en algèbre linéaire

III - Perspectives

Forte demande en **qualité du calcul** :

- coûts réduits
- appréciation des erreurs, augmentation de la précision
- portabilité
- confiance/sûreté des codes

[CEA, IBM, Airbus, Intel]



Côté Arénaire :

- conception des opérateurs arithmétiques
- propriétés du calcul
- calcul scientifique
- maîtrise ad hoc des étapes de conception
- potentiel en transfert de connaissances

Un défi pour le **calcul intensif efficace et fiable** :

la maîtrise combinée

- des coûts de développement
- des aspects matériels
- des bibliothèques de programmes
- des aspects mathématiques et algorithmiques
- de méthodes de validation

Un défi pour le **calcul intensif efficace et fiable** :

la maîtrise combinée

{ des coûts de développement
des aspects matériels
des bibliothèques de programmes
des aspects mathématiques et algorithmiques
de méthodes de validation

“Nouveaux outils automatiques pour le calcul flottant validé”

Comment

↪ évaluer une formule mathématique (voire un programme) en calcul flottant ?

↪ automatiser la génération de code pour cette évaluation ?

↪ assurer la qualité des résultats ?

Axe 1. Opérateurs arithmétiques et évaluation de fonctions

- Applications embarquées
- Approximation et évaluation, évaluation d'expression (compilation)
- Opérateurs matériels : virgules fixe et flottante, FPGA
- Opérateurs logiciels : arrondi correct, processeurs entiers, augmentation de la précision

Axe 2. Propriétés, validation, calcul certifié

- Propriétés des opérateurs, fma, opérateurs composites
- Bornes d'erreurs, erreur de méthode/d'arrondi
- Liens avec la preuve formelle
- Complexité algorithmique, modèles de nombres et d'intervalles
- Algorithmes en calcul certifié

Axe 3. Nouveaux outils automatiques

- Méthodologie de développement
- Adaptabilité au matériel
- Génération de code
- Génération de preuve formelle
- Interopérabilité