ENSTA IP PARIS Lab-STICC COLLEGES BRETAGNE LOIRE SCIENCES POUR L'INGENIEUR ET LE NUMERIQUE

Accélération matérielle de l'arithmétique des intervalles pour la robotique mobile

Thèse de doctorat

Pierre Filiol

28 avril 2026

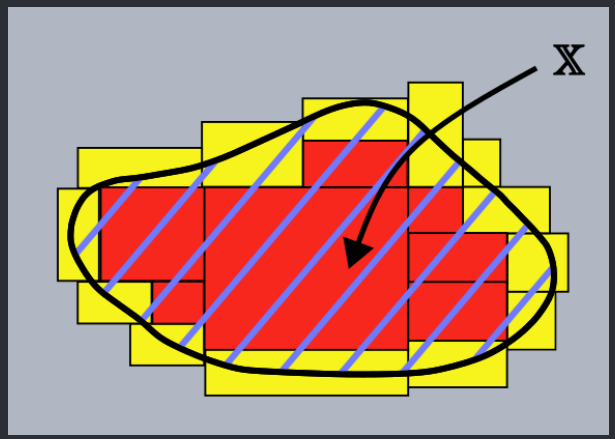
Sommaire

Sommaire

Interêt des méthodes par intervalles en robotique

- Certaines tâches récurrentes en robotique mobile sont modélisables par des problèmes de satisfaction de contraintes.
- Pour des contraintes non-linéaires, la résolution est difficile.
- L'arithmétique des intervalles permet de générer un pavage garanti de l'ensemble solution $\mathbb{X}$.

$$\mathcal{H} : \begin{cases} X : \{x_1, x_2, \ldots, x_n\} \\ D : \{\mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_n\} \\ C : \{C_1, C_2, \ldots, C_m\} \end{cases} \Rightarrow$$



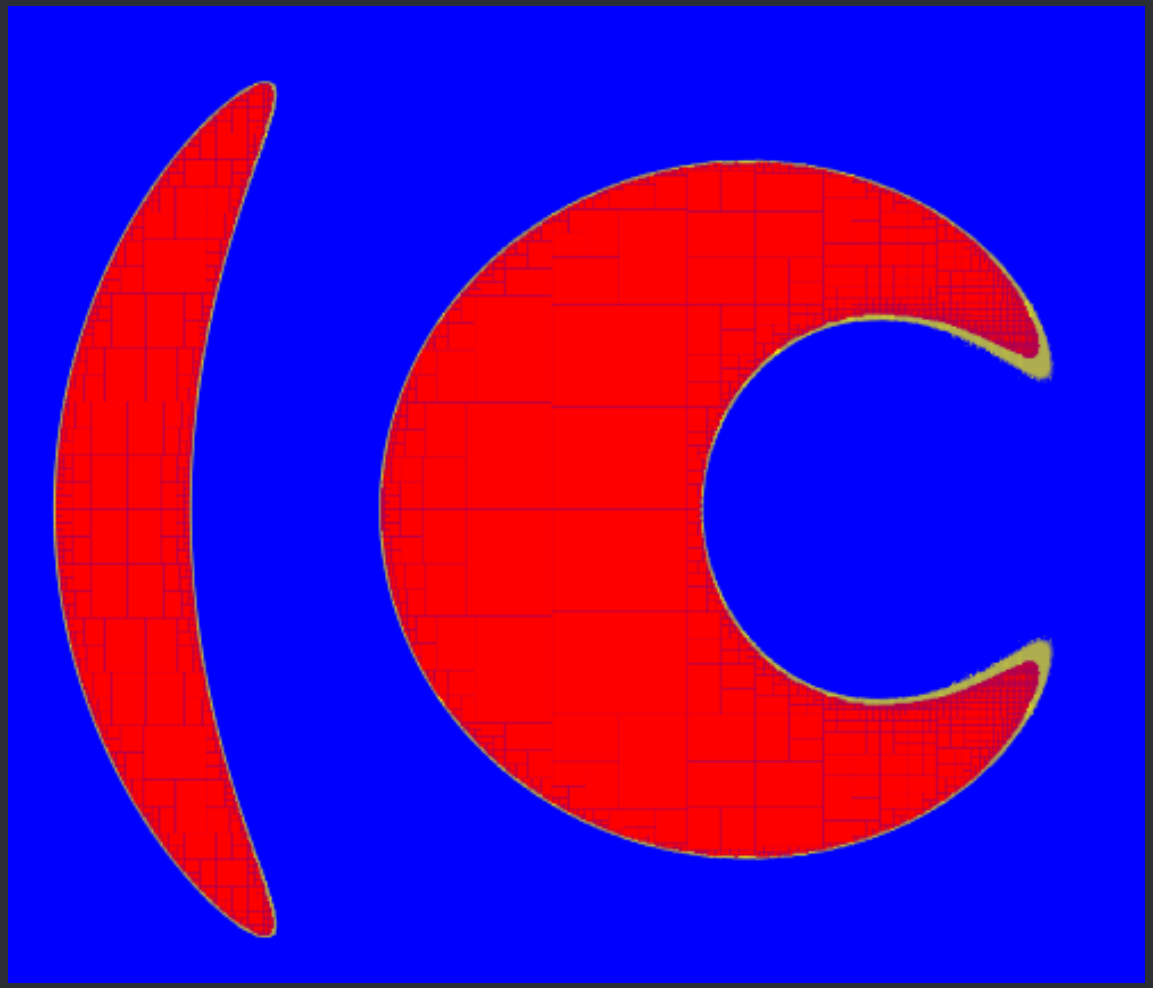
Exemple

Considérons le problème de satisfaction de contraintes suivant

$$\mathcal{H} : \begin{cases} X : (x_1, x_2) \\ D : ([-3.5, 2], [-3, 3]) \\ C : \mathbf{f}(X) \in [0.325, +\infty] \end{cases} \qquad \text{avec } \mathbf{f}(X) = \frac{\sin({x_1}^2 + {x_2}^2)}{e^{x_1} + {x_2}^2}$$

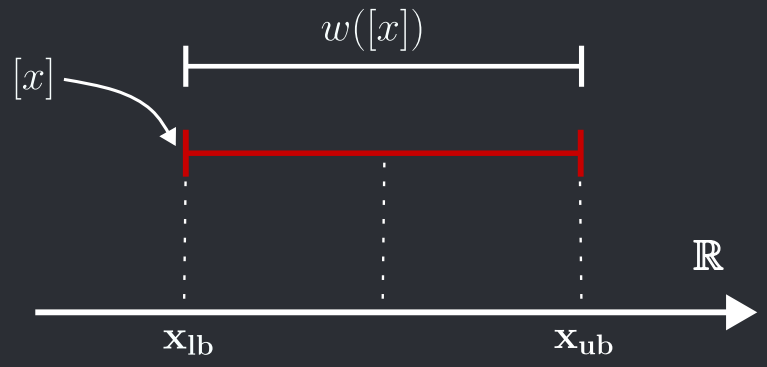
- L'arithmétique des intervalles permet de générer une approximation garantie de l'ensemble des solutions.

Exemple *(suite)*



Bases du calcul avec des intervalles

- Les intervalles sont utilisés comme opérande principal.



- La norme IEEE-1788 définit les régles de calcul pour les intervalles :
  - Opérateurs arithmétiques.
  - Fonctions transcendentales.

Contracteurs - Principe

- Considérons le problème $\mathcal{H}$ ci-dessous et sa solution $\mathbb{S}$:

$$\mathcal{H}: \begin{cases} X: (x_1, x_2, x_3) \\ D: (\mathbb{R}, \mathbb{R}, [0, +\infty]) \text{ où } \\ C: \mathbf{f}(X) = \mathbf{0} \end{cases} \quad \mathbf{f}(X) = \begin{pmatrix} x_1 + \sin(x_2) + x_3 \\ x_1 - x_2 - 3 \end{pmatrix}$$

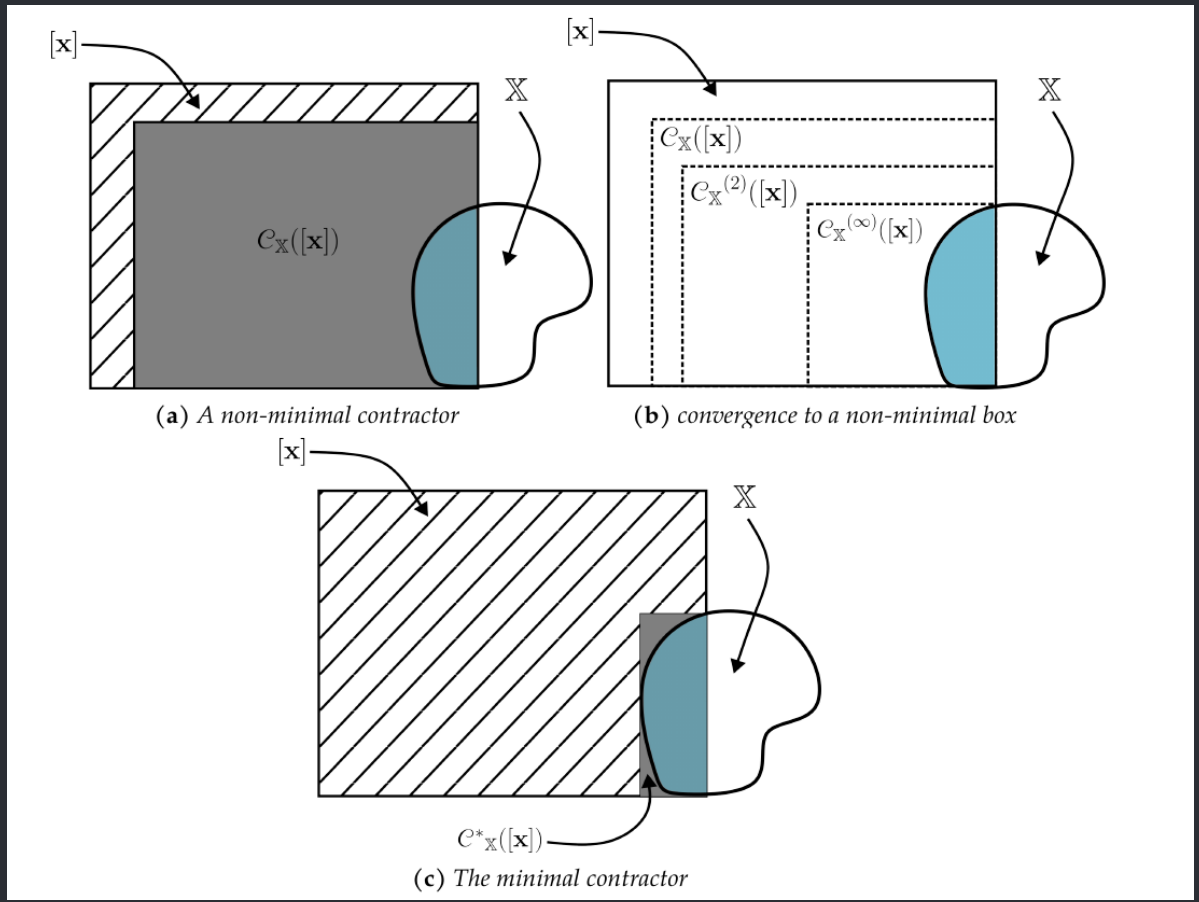
- Approximation garantie de $\mathbb{S}$ par propagation de contraintes:

$$\mathcal{H}: \begin{cases} X: (x_1, x_2, x_3) \\ D: (\mathbb{R}, \mathbb{R}, [0, +\infty]) \\ C: \mathbf{f}(X) = \mathbf{0} \end{cases} \Rightarrow \qquad \mathcal{H}: \begin{cases} X: (x_1, x_2, x_3) \\ D': ??? \\ C: \mathbf{f}(X) = \mathbf{0} \end{cases}$$

- But: réduire D' tq $\boxed{\mathbb{S} \subset D' \subset D}$ sans perdre de solutions.

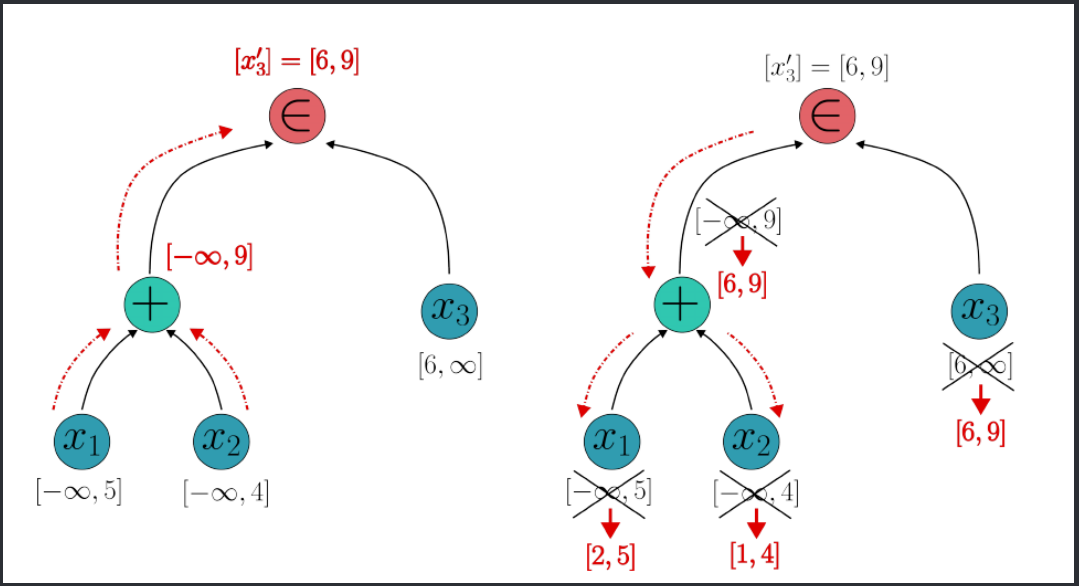
Contracteurs - Visualisation

- Un tel opérateur est appelé *contracteur*: $\mathcal{C}_{\mathbb{S}}$: $\begin{cases} \mathbb{IR}^n \longrightarrow \mathbb{IR}^n \\ [\mathbf{x}] \longrightarrow \mathcal{C}_{\mathbb{S}}([\mathbf{x}]) \end{cases}$



Contracteurs élémentaires

- Prenons l'exemple d'une simple contrainte d'addition:
  - $x_1 + x_2 = x_3$
  - $x_1 \in [-\infty, 5], x_2 \in [-\infty, 4], x_3 \in [6, +\infty]$



- Forward:

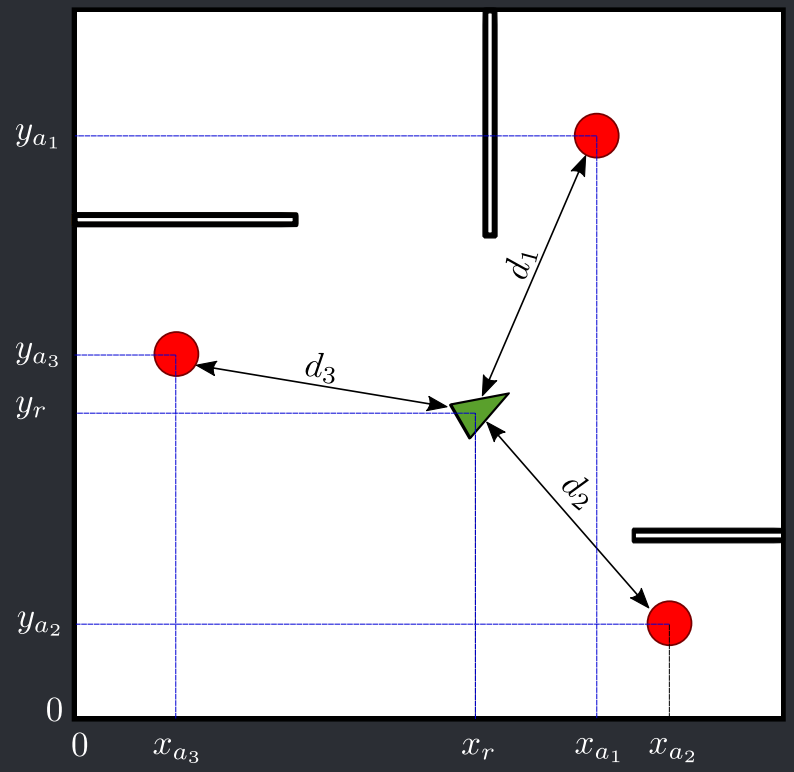
$$x_3' = (x_1 + x_2) \cap x_3$$

- Backward:

$$x_1' = (x_3' - x_2) \cap x_1$$
$$x_2' = (x_3' - x_1) \cap x_2$$

Cas pratique - Problème de localisation

- Localisation d'un robot dans un champs d'amers.



$$\mathcal{H} : \begin{cases} X : (x, y) \\ D : (\mathbb{R}^+, \mathbb{R}^+) \text{ avec} \\ C : \mathbf{f}(X) \in (d_1^2, d_2^2, d_3^2) \end{cases}$$

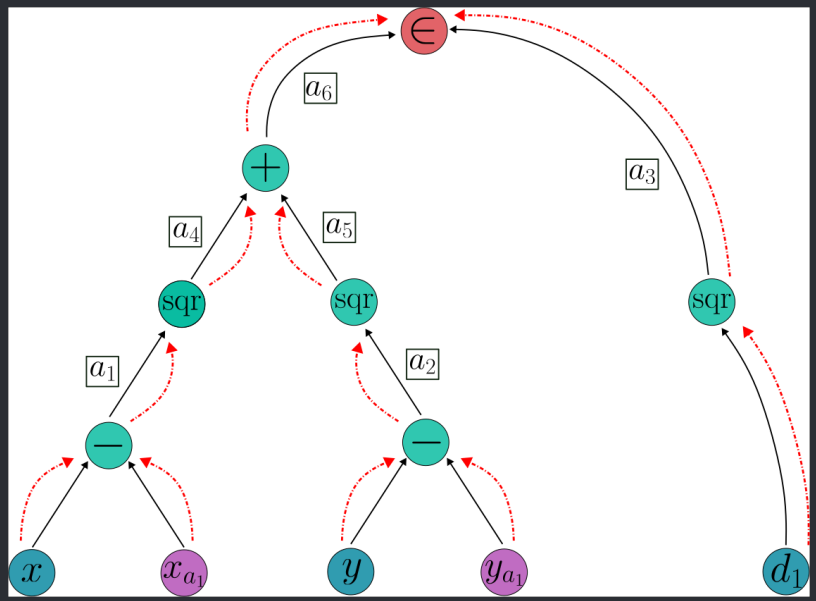
$$\mathbf{f}(X) : \begin{cases} (x - x_{a_1})^2 + (y - y_{a_1})^2 \\ (x - x_{a_2})^2 + (y - y_{a_2})^2 \\ (x - x_{a_3})^2 + (y - y_{a_3})^2 \end{cases}$$

Cas pratique - Contracteur complexe

- Considérons la contrainte liée au premier amer:
  - $C_1 : (x - x_{a_1})^2 + (y - y_{a_1})^2 \in d_1^2$ *(Amer 1)*
- Construction du contracteur grâce à l'algorithme HC4-R.
  - Méthode basée sur la propagation de contraintes.
  - Utilise les contracteurs fwd/bwd élémentaires.

> **Référence**
>
> Benhamou et al. Revising Hull and Box Consistency
> 16th Intl. Conf. on Logic Programming (1999) p.230-244.

Cas pratique - Contracteur complexe *(forward)*



Algorithm 1: Contractor $\overrightarrow{\mathcal{C}_1}$

Data: x, y

1 $a_1 = \overrightarrow{\mathcal{C}_-}(x, x_{a_1})$;

2 $a_2 = \overrightarrow{\mathcal{C}_-}(y, y_{a_1})$;

3 $a_3 = \overrightarrow{\mathcal{C}_{sqr}}(d_1)$;

4 $a_4 = \overrightarrow{\mathcal{C}_{sqr}}(a_1)$;

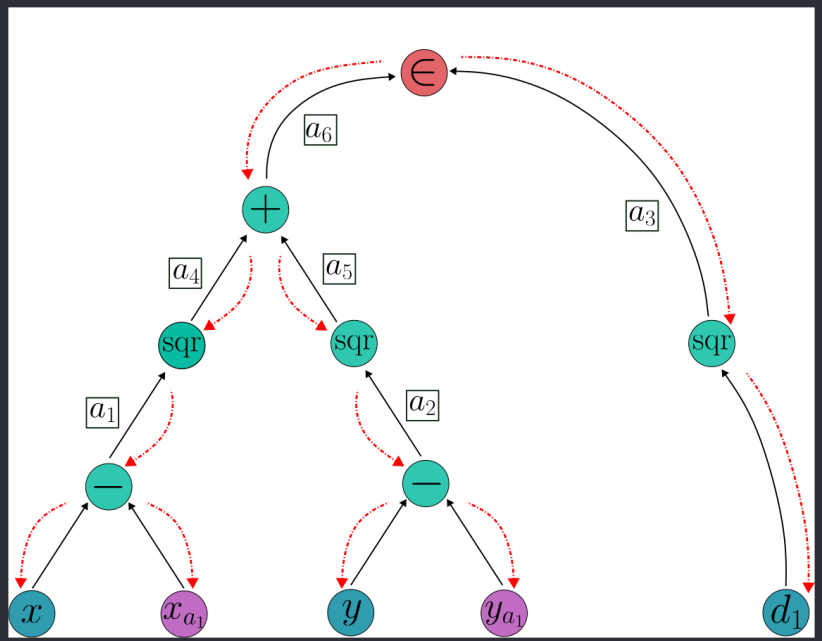
5 $a_5 = \overrightarrow{\mathcal{C}_{sqr}}(a_2)$;

6 $a_6 = \overrightarrow{\mathcal{C}_+}(a_6, a_3)$;

7 $a_6 = a_6 \cap a_3$;

8 **return** x, y, a_{1-6}

Cas pratique - Contracteur complexe *(backward)*



Algorithm 2: Contractor $\overleftarrow{\mathcal{C}_1}$

Data: x, y, a_{1-6}

1 $a_4, a_5 = \overleftarrow{\mathcal{C}_+}(a_4, a_5, a_6)$;

2 $a_1 = \overleftarrow{\mathcal{C}_{sqr}}(a_1, a_4)$;

3 $a_2 = \overleftarrow{\mathcal{C}_{sqr}}(a_2, a_5)$;

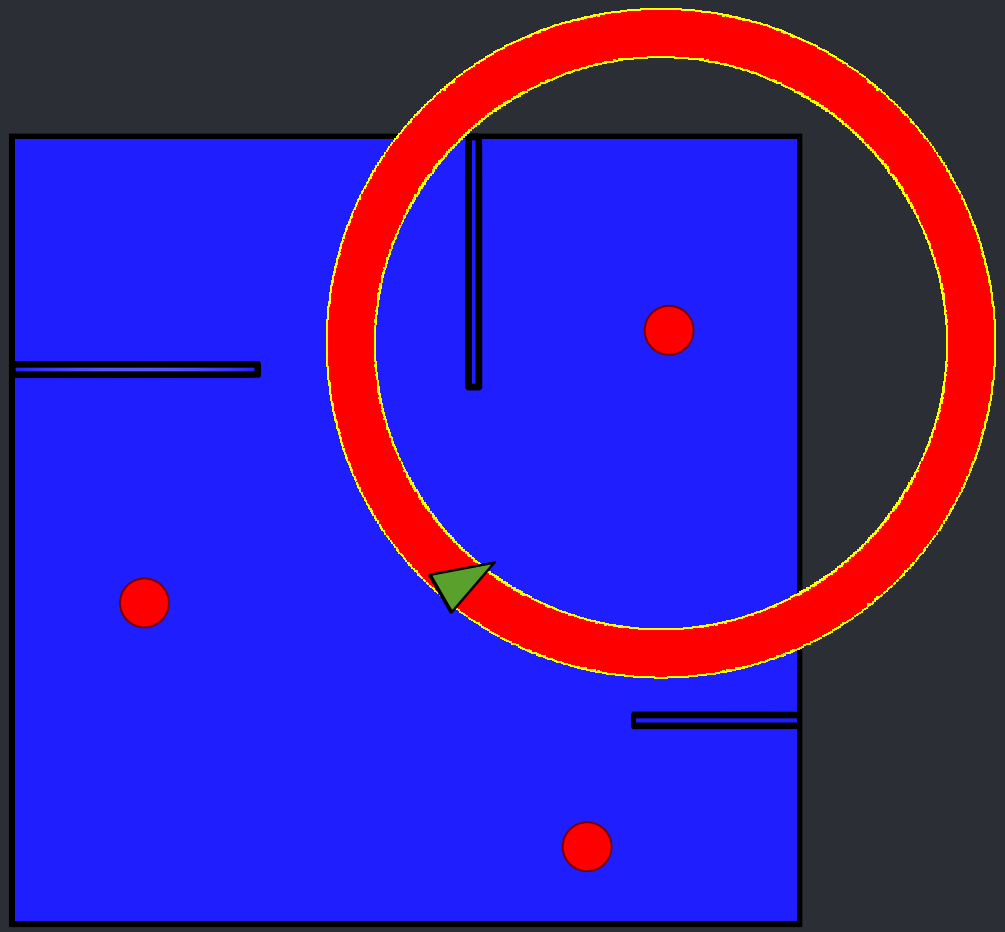
4 $x = \overleftarrow{\mathcal{C}_-}(x, x_{a_1}, a_1)$;

5 $y = \overleftarrow{\mathcal{C}_-}(y, y_{a_1}, a_2)$;

6 **return** x,y

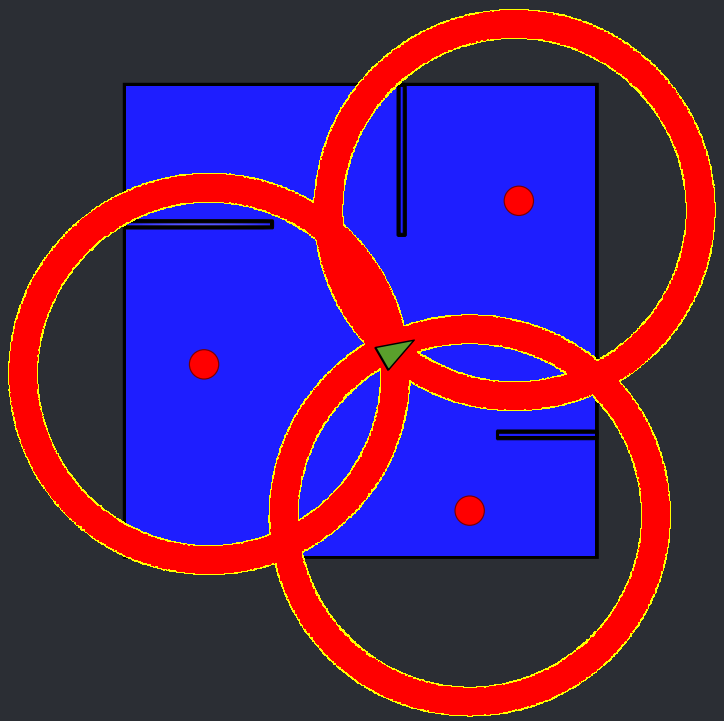
Cas pratique - Génération du pavage pour C_1

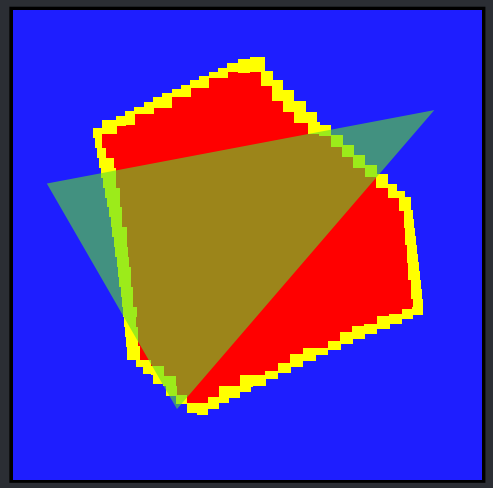
- Algorithme SIVIA (Set Inverter via Interval Analysis)
  (Jaulin & Walter, 1993)



Cas Pratique - Résolution du problème complet

- Pour résoudre le problème de localisation initial:
  - On procède de même pour C_2 et C_3.
  - $\boxed{C_{\mathcal{H}}([\mathbf{x}]) = \mathcal{C}_1([\mathbf{x}]) \cap \mathcal{C}_2([\mathbf{x}]) \cap \mathcal{C}_3([\mathbf{x}])}$





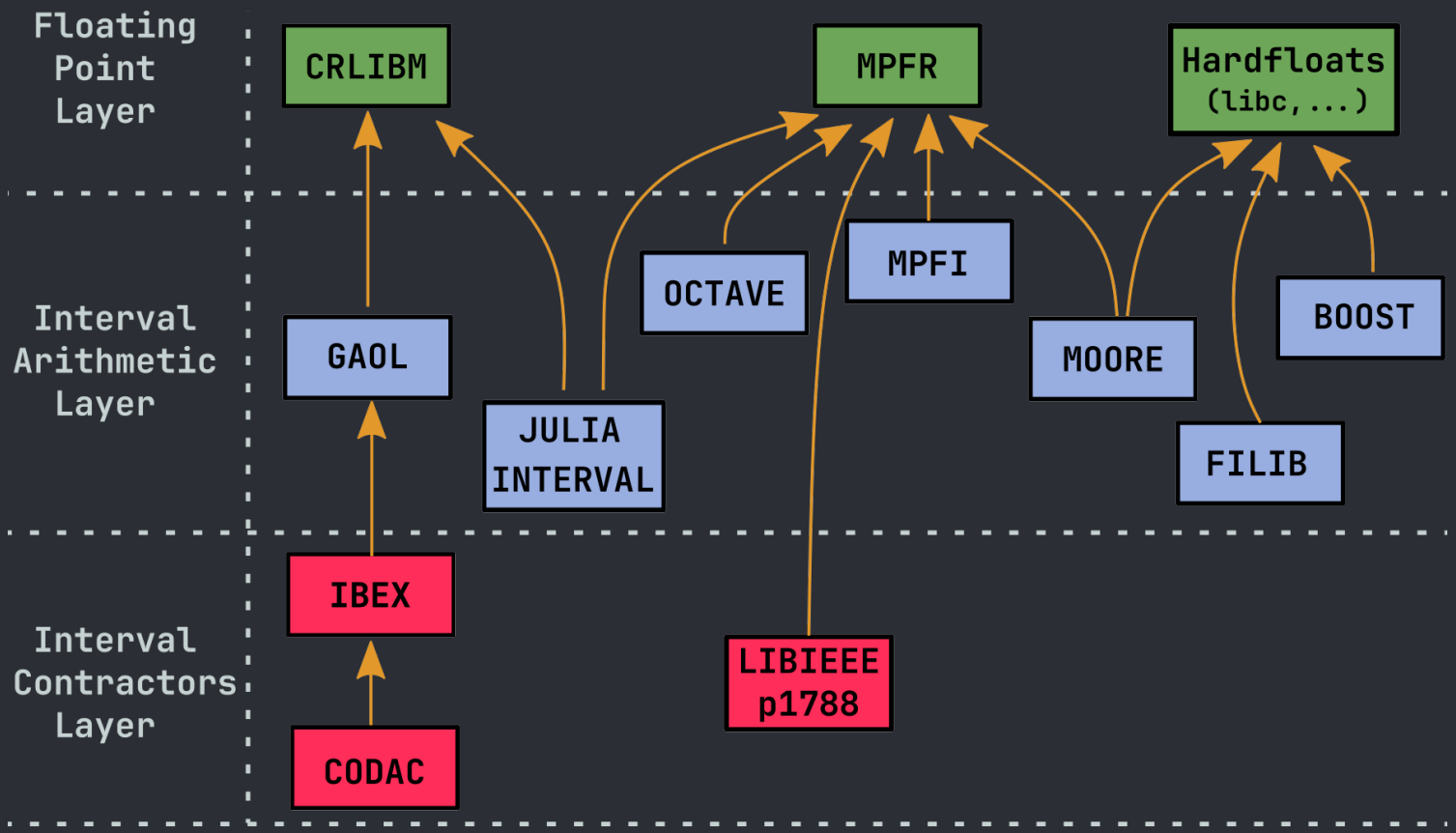
Sommaire

Défis des approches par intervalles

- Les méthodes par intervalles sont utiles à la robotique.
  - Garanties de calcul pour des applications critiques.
- Leur mise en place pose des problèmes pratiques:
  - Méthodes gourmandes en ressources.
  - Calculs en temps contraint.
- Cela rend le déploiement embarqué complexe.
  - Besoin de hardware performant.
  - Problématique de l'autonomie.

Implémentations logicielles

- Un riche écosystème de bibliothèques logicielles.



Interêt d'une approche matérielle

Thèse défendue

La robotique mobile bénéficie d'une approche matérielle de l'arithmétique des intervalles.

- Les architectures matérielles généralistes sont un frein à la performance des opérateurs intervalles.

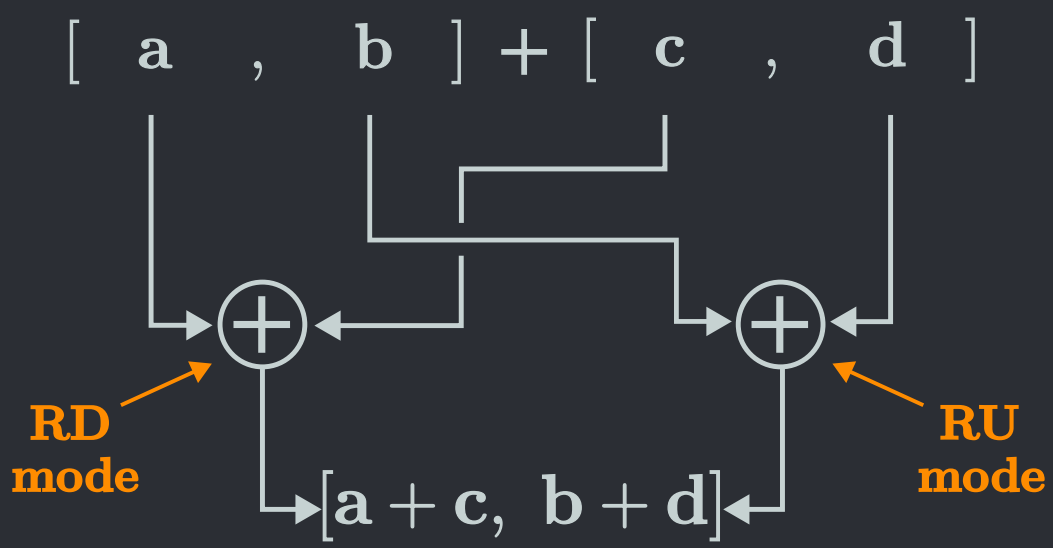
- Les stratégies proposées par les bibliothèques logicielles intervalles en terme de calcul flottant n'offrent pas un compromis adapté à la robotique.

Les limites du hardware généraliste

- Limitation majeure: la gestion des modes d'arrondi flottant.

- Les bornes inférieures et supérieures doivent être évaluées dans les modes d'arrondi IEEE-754 appropriés.
  - Condition pour un résultat garanti sur un opérateur intervalle.

- Exemple de l'addition:



Mitigation du problème

- GAOL résoud partiellement ce problème sur hardware généraliste.
- Exploitation de la symétrie entre les modes d'arrondi:

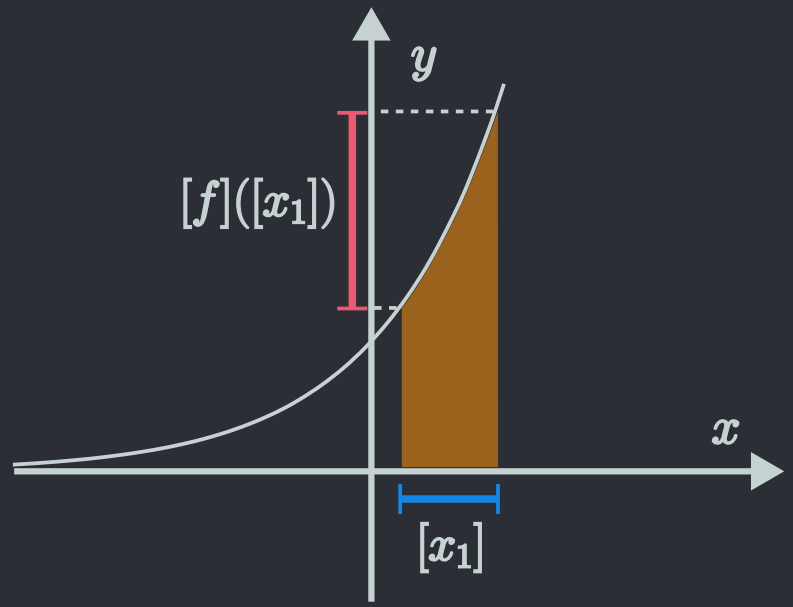
$$\boxed{rd_{+\infty}(x) = -rd_{-\infty}(-x)}$$

- Utilisation d'instructions SIMD pour une évaluation concurrente.
  - Astuce limitée aux opérateurs supportés.
  - Pas d'exponentielle, logarithme ou trigonométrie.

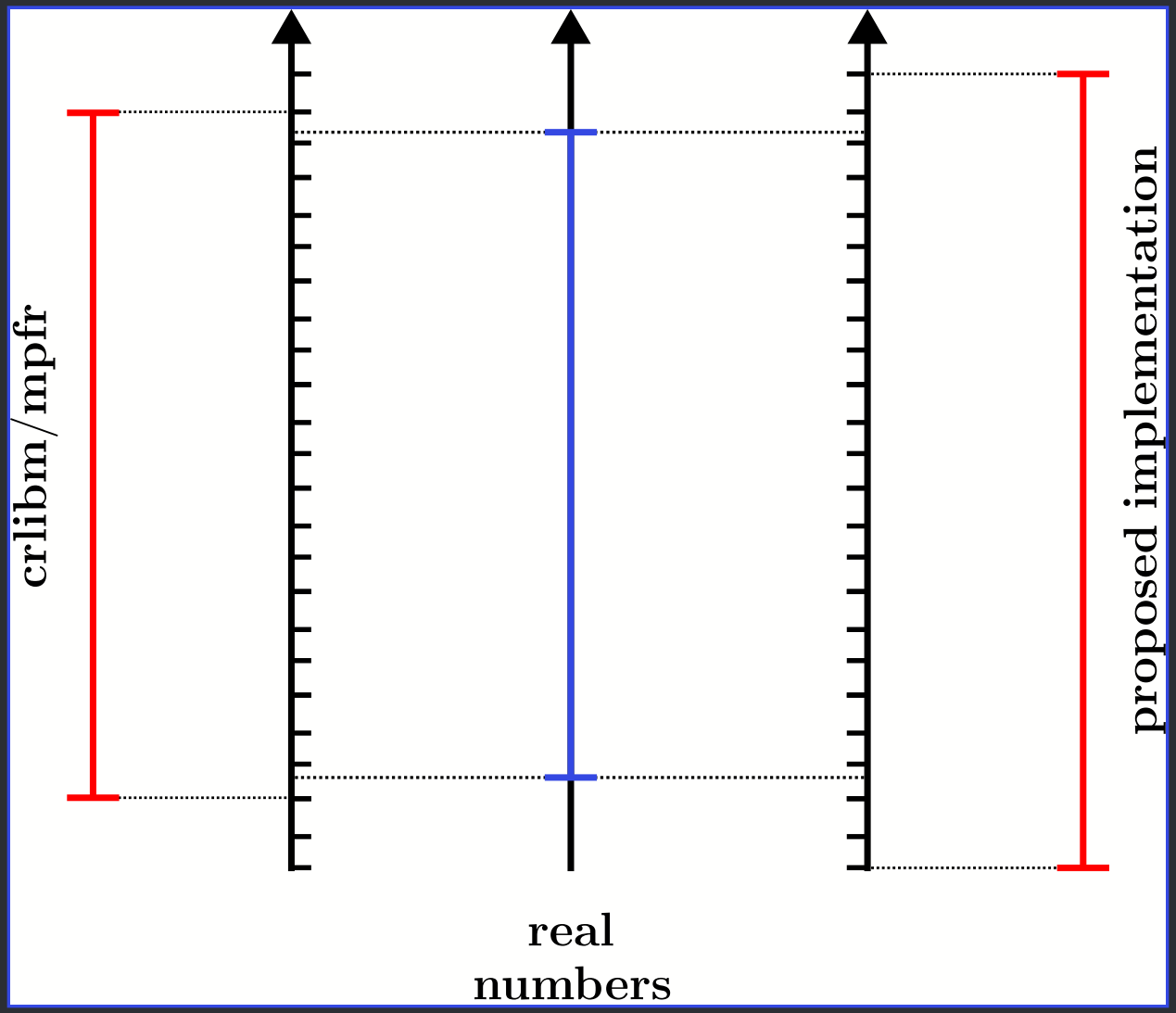
> **Référence**
>
> Goualard, Frédéric. (2010). Fast and Correct SIMD Algorithms for Interval Arithmetic.

Stratégie des calculs en virgule flottante

- Approches similaires dans les couches flottantes des librairies itv:
  - **crlibm:** arrondis garantis en double précision.
  - **mpfr:** idem en précision arbitraire, softfloat.
- La robotique peut accepter plus de pessimisme dans les calculs:
  - Il est essentiel de maintenir les garanties de calcul.
  - Le critère de *tinyness* est secondaire.



Garanties et tinyness



Contributions scientifiques

1. Analyse du socle théorique de l'arithmétique des intervalles.
   - Sélection de primitives d'accélération adaptées à la robotique.
   - Contribution au socle théorique - Algèbre du iota.

2. Implémentation des primitives matérielles.
   - Design en VHDL et synthèse sur FPGA.

3. Intégration au sein de l'architecture RISC-V.
   - Proposition d'une extension de l'ISA *xinterval*.

4. Proposition d'une stratégie *Hardware-in-loop* pour:
   - La validation/évaluation des performances du jeu d'instructions.
   - Facteur d'accélération de l'ordre de 3× à 8×.

Contributions scientifiques

- *Filiol et al "A New Interval Arithmetic To Generate the Complementary of Contractors". Acta Cybernetica (2024)*

- *Filiol et al. “RISC-V Based Hardware Acceleration of Interval Contractor Primitives in the Context of Mobile Robotics.” Acta Cybernetica (2024)*

- *Filiol et al. "Efficient Hardware Primitives for Interval Contractors in Robotics and Integration to a Custom RISC-V ISA Extension, NEWCAS Conference (2025)"*

- *Filiol Pierre et al. "An Iterative Design and Validation Methodology for RISC-V Custom Extension" ICECS Conference (2025)*

Sommaire

Objectifs de l'implémentation

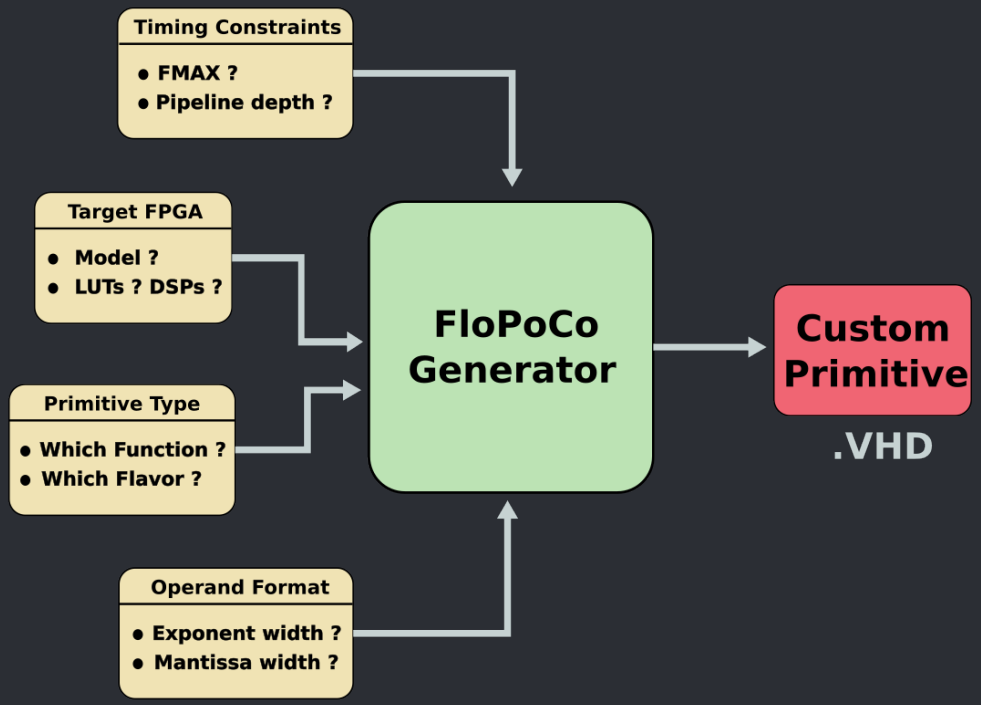
- Implémentation hardware d'opérateurs intervalle.
  - Exploitation de parallélisme spatial sur le calcul des bornes.
  - Proposer une architecture adaptée aux changements d'arrondis.
  - Maitriser plus finement la couche calcul flottant.

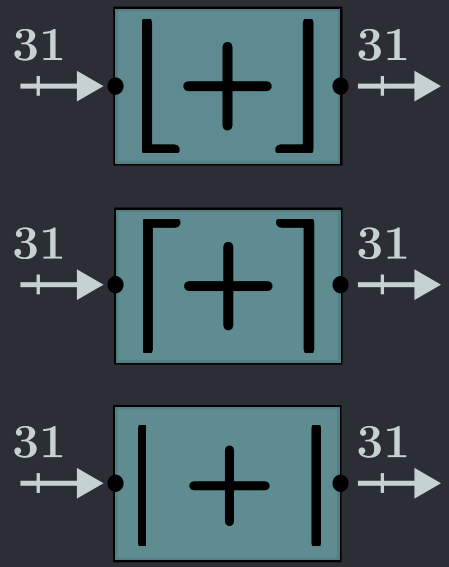
- Focus sur les contracteurs forward/backward élémentaires:
  - Opérateurs arithmétiques: $+, -, \times, /, x^2, \sqrt{x}$.
  - Fonctions réelles: $\exp, \log, \sin, \cos$

> **Référence**
>
> Kirchner, Kulisch, U.W. Hardware Support for Interval Arithmetic.
> Reliable Comput 12, 225-237 - 2006.

Un mot sur les primitives flottantes

- Utilisation de FloPoCo pour générer les opérateurs flottants.





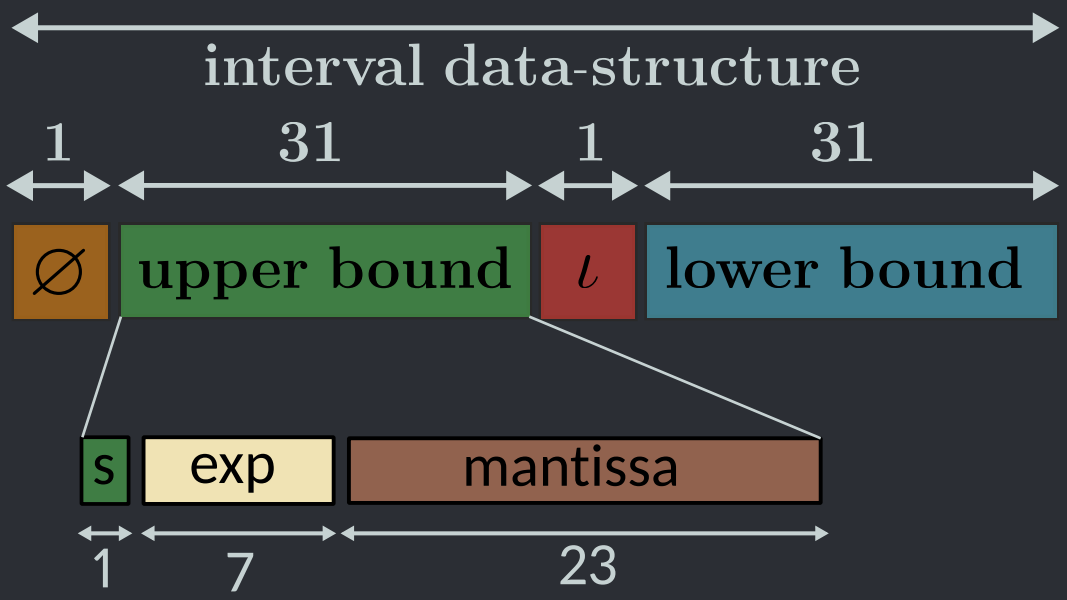
Référence

Dinechin, Pasca, Designing custom arithmetic data paths with FloPoCo. IEEE Design Test of Computers, 18-27, 2011

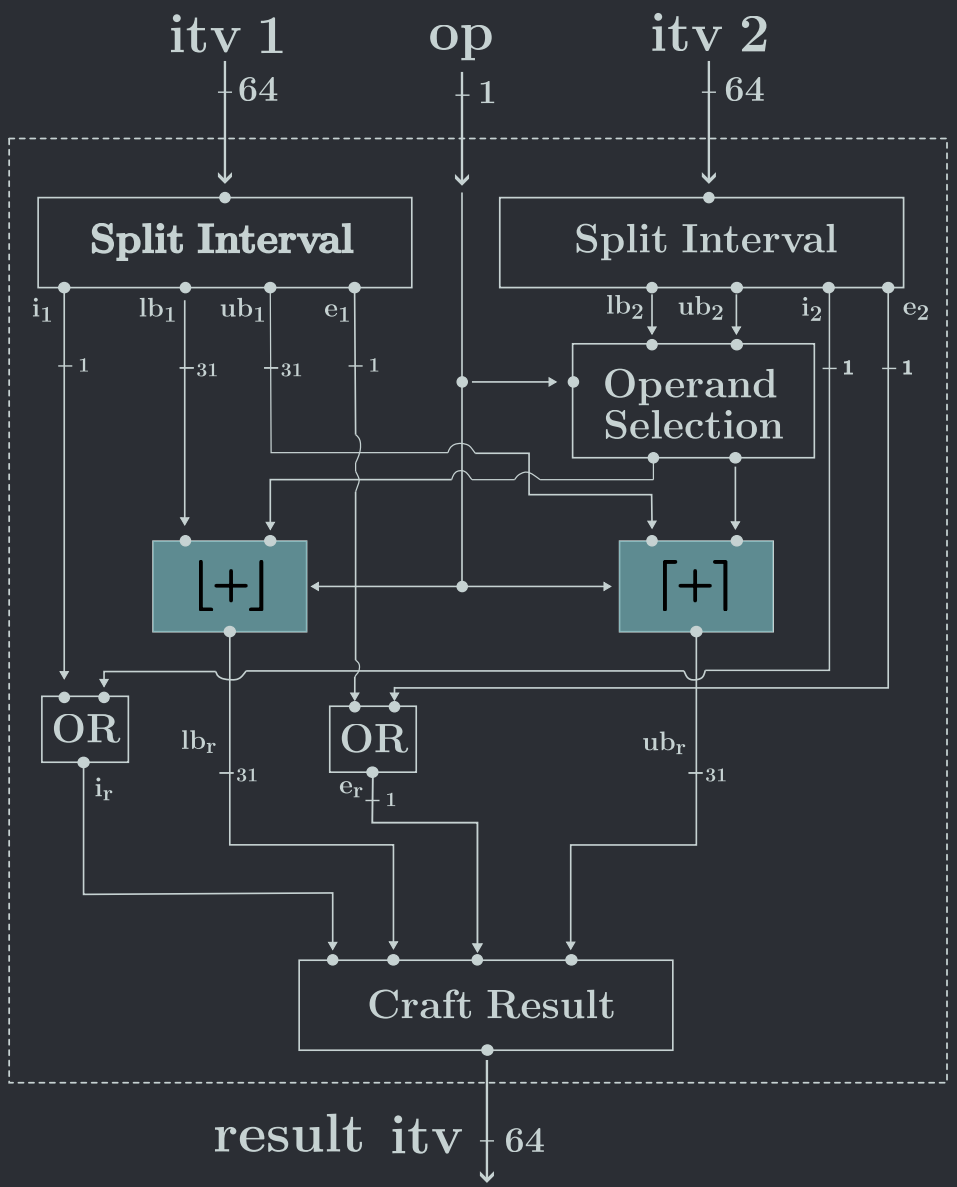
Représentation matérielle d'un intervalle

- Contrainte: insertion dans les registres 64-bit du RISC-V.
  - 2 bits pour les flags *empty* et *iota*.
  - 2 bornes en format Float31 custom.

- Sacrifice d'un bit d'exposant ⇒ plage normale ≈ $[10^{-19}, 10^{38}]$



Opérateur d'addition/soustraction



Operations:

$$[z] = [x] + [y]$$

$$[z] = [x] - [y]$$

Etapes:

1. Extract itv fields.
2. Select operands.
3. Concurrent adders.
4. Craft Result.

Opérateur de multiplication

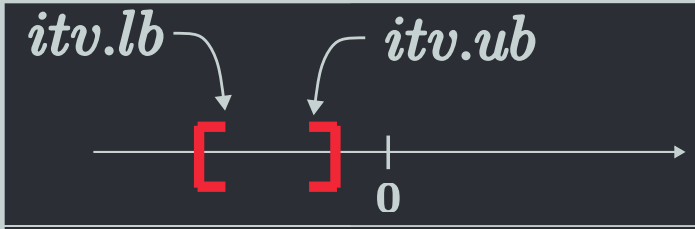
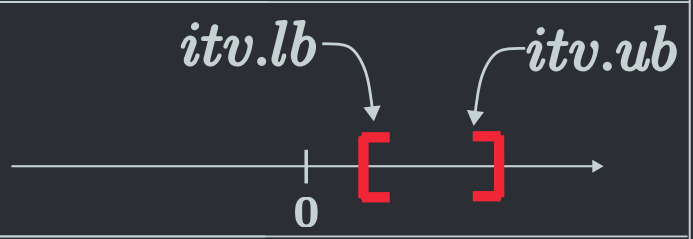
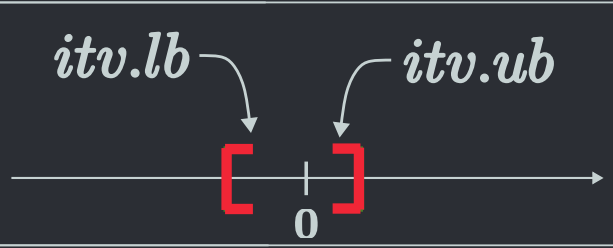
- Règle pour la multiplication d'intervalles:

$$\begin{aligned}[x] \times [y] &= [x_{lb}, x_{ub}] \times [y_{lb}, y_{ub}] \\ &= [\underbrace{\{x_{lb}y_{lb}, x_{lb}y_{ub}, x_{ub}y_{lb}, x_{ub}y_{ub}\}}_{\mathrm{L}}] \\ &= [\min(L), \max(L)]\end{aligned}$$

- En logiciel, l'implémentation requiert:
  - 4 multiplications flottantes (probablement séquentielles).
  - Tri croissant des produits obtenus.
  - Sélection du min/max.

Opérateur de multiplication

- Le matériel permet de classifier les opérandes à bas coût.
  - 2 comparateurs et un MUX.

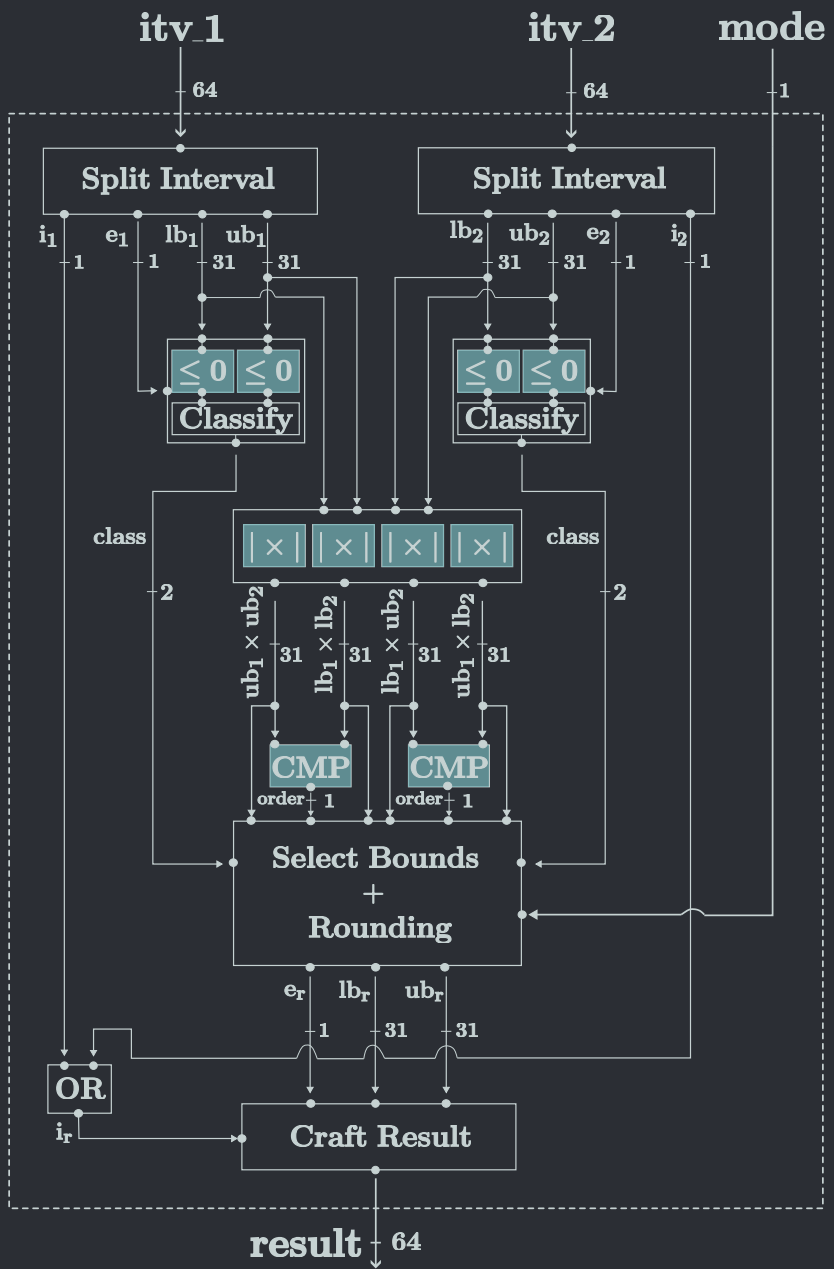
code	format	code	format
000	$\varnothing$	011	
001	$\{0\}$	100	
010	$[-\infty, +\infty]$	101	

Opérateur de multiplication

- Cette classification permet de prédire la forme du résultat.
  - multiplications réalisées de façon concurrente.
  - une comparaison en plus dans le pire des cas.

itv2 \ itv1	011	100	101
011	$[\mathbf{ub_1ub_2}, \mathbf{lb_1lb_2}]$	$[\mathbf{ub_1lb_2}, \mathbf{lb_1ub_2}]$	$[\mathbf{ub_1lb_2}, \mathbf{lb_1lb_2}]$
100	$[\mathbf{lb_1ub_2}, \mathbf{ub_1lb_2}]$	$[\mathbf{lb_1lb_2}, \mathbf{ub_1ub_2}]$	$[\mathbf{lb_1ub_2}, \mathbf{ub_1ub_2}]$
101	$[\mathbf{lb_1ub_2}, \mathbf{lb_1lb_2}]$	$[\mathbf{ub_1lb_2}, \mathbf{ub_1ub_2}]$	$[\min(\mathbf{lb_1ub_2}, \mathbf{ub_1lb_2}), \max(\mathbf{lb_1lb_2}, \mathbf{ub_1ub_2})]$

Opérateur de multiplication

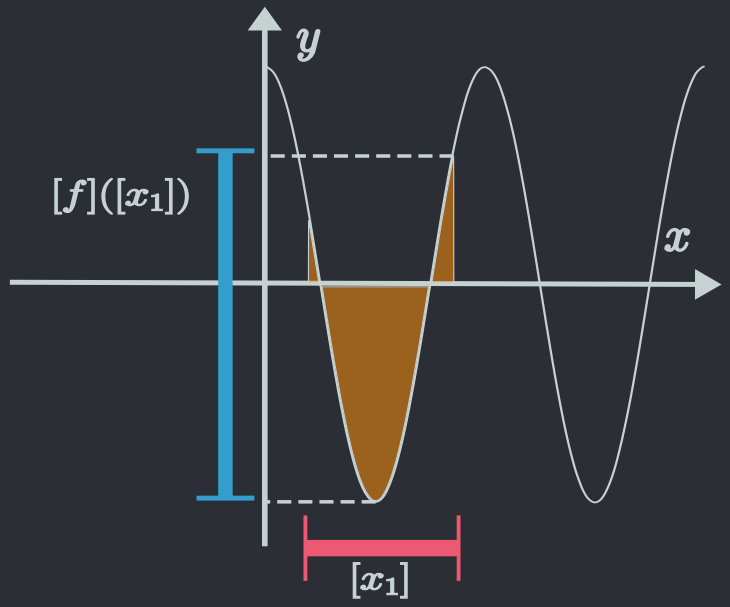


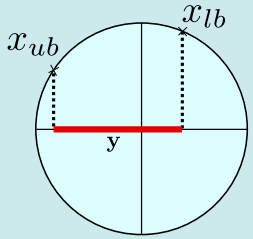
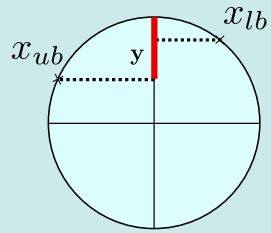
Etapes:

1. Operand classification.
2. Products computation.
3. Tie-break (worst case).
4. Pick result bounds.
5. Craft final result.

Opérateurs de trigonométrie

- Très utiles dans les applications robotiques.



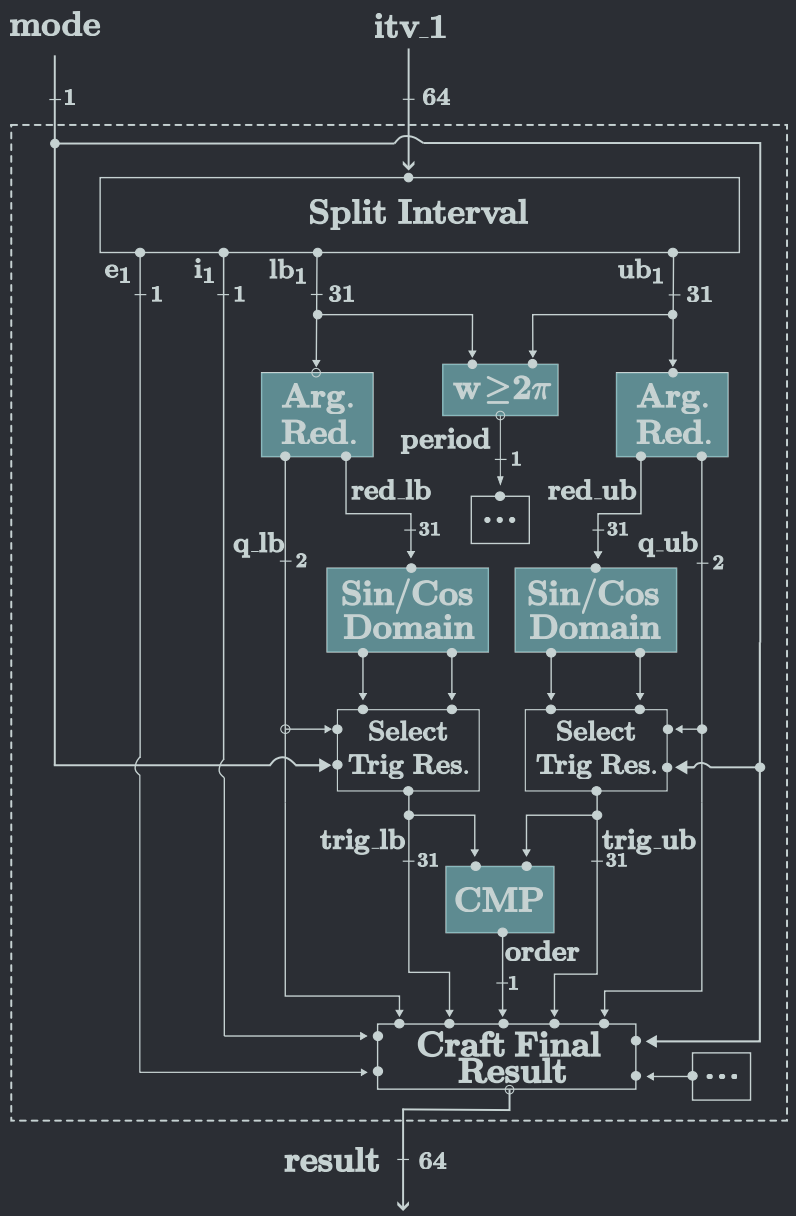
$\mathbf{01}$ $\mathbf{q_{lb}}\|\mathbf{q_{ub}}$	cos	sin
	 $y_{lb} = cos(x_{ub})$ $y_{ub} = cos(x_{lb})$	 $y_{lb} = min(sin(x_{lb}), sin(x_{ub}))$ $y_{ub} = 1$

- On se limite aux contracteurs élémentaires sur le cosinus/sinus.

$$[y] = \cos([x])$$

$$[y] = \sin([x])$$

Opérateurs de trigonométrie



Etapes:

1. Quadrant retrieval.
2. Argument reduction.
3. Convert to Q1.23.
4. Concurrent CORDICs.
5. Convert to Float31.
6. Apply truth table.
7. Craft final result.

Résultats de synthèse sur Xilinx XC7Z020@100MHz

Hardware Entity	LUTs	FFs	DSPs	Latency (cycles)
itv_make	0.4%	0.3%	0%	**5**
itv_extract	0.2%	0.2%	0%	**1**
itv_set	0.8%	0.4%	0%	**4**
itv_add	2.3%	1%	0%	**8**
itv_mul	1.8%	0.9%	3.6%	**7**
itv_div	6.5%	3%	0%	**16**
itv_sqrt	1.5%	0.6%	0%	**12**
itv_exp	1.2%	0.8%	0.9%	**13**
itv_log	2.6%	1.6%	2.7%	**12**
itv_trig	18.4%	5.9%	7.2%	**42**
itv_trig_inv	10.7%	4%	3.6%	**42**

Sommaire

Exploitation des primitives matérielles

- Quelle est la meilleure façon de les exposer au développeur ?
  - Approche purement matérielle ?
  - Coprocesseur spécialisé ?
  - Extension du jeu d'instructions CPU ?

- On opte pour une extension de jeu d'instructions:
  - Flexibilité du modèle "CPU".
  - Transparent pour le développeur.
  - Réutilisation de code existant.

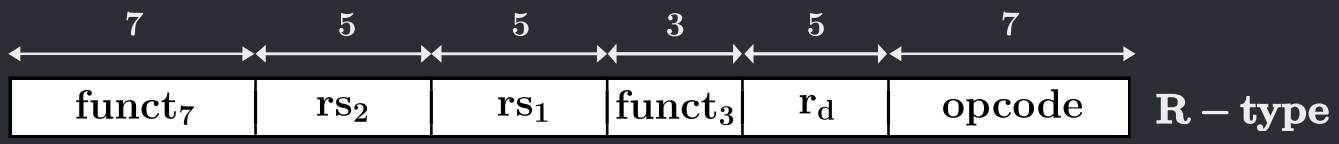
Choix de l'architecture RISC-V

- Le standard RISC-V possède plusieurs avantages:
  - ISA ouvert et prévu pour les extensions customs.
  - Ecosystème logiciel clé en main.

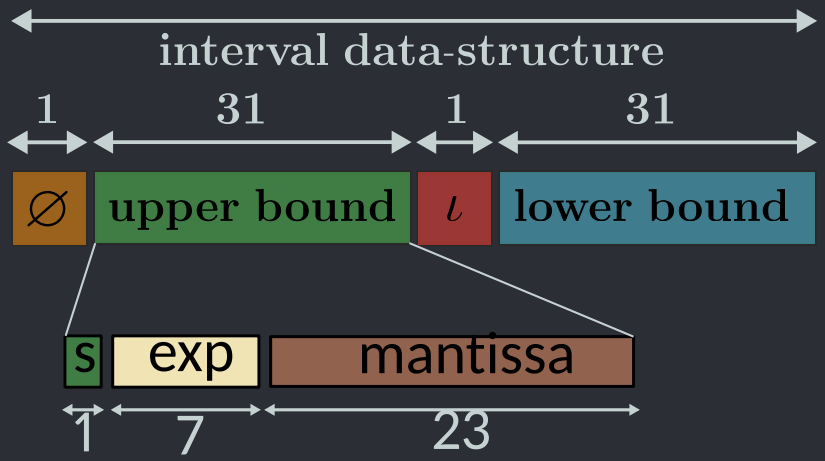
		opcode[4 : 2]							
opcode[6 : 5]		000	001	010	011	100	101	110	111
	00	LOAD	LOAD-FP	CUSTOM 0	MISC MEM	OP-IMM	AUIPC	OP-IMM 32	48b
	01	STORE	STORE-FP	CUSTOM 1	AMO	OP	LUI	OP-32	64b
	10	MADD	MSUB	NMSUB	NMADD	OP-FP	OP-V	CUSTOM 2	48b
	11	BRANCH	JARL	reserved	JAL	SYSTEM	OP-VE	CUSTOM 3	≥ 80b

L'Extension Custom *xinterval*

- Minimiser l'impact de l'intégration sur l'architecture existante:
  - S'appuyer sur les extensions F et D.
  - Insertion des intervalles dans les registres double-précision.
  - Utilisation de l'opcode *custom-0* et du format d'instruction R.



- Pour rappel:



Les instructions de *xinterval*

register content interpretation	
reg **interval**	reg **integer**
reg **float32**	reg **float64**

- **Création d'intervalle**

asm	r_d	rs_1	rs_2	effect	description
itv.mk	✓	✓	✓	$\{r_d\} = \text{itv}(\{rs_1\}, \{rs_2\})$	Create interval from float bounds
itv.mki	✓	✓	✓	$\{r_d\} = \text{itv}(\{rs_1\}, \{rs_2\}) \cap \iota$	Create interval from float bounds + iota

- **Extraction des champs**

asm	r_d	rs_1	rs_2	effect	description
itv.ext.e	✓	✓	✗	$\{r_d\} = 1$ if $\{rs_1\}$ is empty else 0	Check empty state of selected itv
itv.ext.i	✓	✓	✗	$\{r_d\} = 1$ if $\{rs_1\}$ is iota else 0	Check iota state of selected itv
itv.ext.lb	✓	✓	✗	$\{r_d\} = \text{lb}(\{rs_1\})$	Get lower bound of selected itv
itv.ext.ub	✓	✓	✗	$\{r_d\} = \text{ub}(\{rs_1\})$	Get upper bound of selected itv

Les instructions de *xinterval*

register content interpretation	
reg interval	reg integer
reg float32	reg float64

- **Opérations ensemblistes**

asm	r_d	rs_1	rs_2	effect	description
itv.set.u	✓	✓	✓	$\{r_d\} = \{rs_1\} \cup \{rs_2\}$	Union of two itv
itv.set.i	✓	✓	✓	$\{r_d\} = \{rs_1\} \cap \{rs_2\}$	Intersection of two itv

- **Demi-contracteurs**

asm	r_d	rs_1	rs_2	effect	description
itv.add	✓	✓	✓	$\{r_d\} = \{rs_1\} + \{rs_2\}$	add two itv
itv.sub	✓	✓	✓	$\{r_d\} = \{rs_1\}$ - $\{rs_2\}$	subtract two itv
itv.mul	✓	✓	✓	$\{r_d\} = \{rs_1\} \times \{rs_2\}$	multiply two itv
itv.div	✓	✓	✓	$\{r_d\} = \{rs_1\} / \{rs_2\}$	divide two itv

Les instructions de *xinterval*

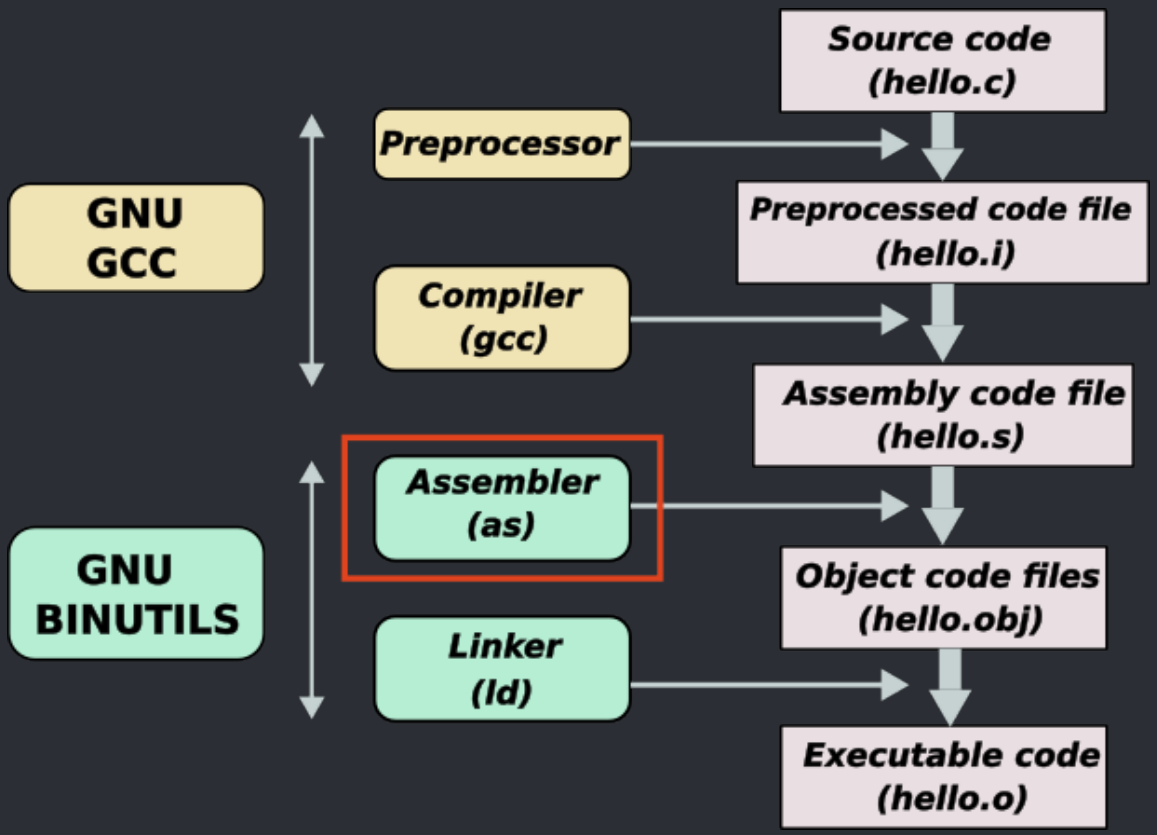
register content interpretation	
reg **interval**	reg **integer**
reg **float32**	reg **float64**

- **Demi-contracteurs**

asm	r_d	rs_1	rs_2	effect	description
itv.sqr	✓	✓	✗	$\{r_d\} = \text{sqr}(\{rs_1\})$	square of the selected itv
itv.sqr_revh	✓	✓	✗	$\{r_d\} = \text{itv}(-\sqrt{lb(\{rs_1\})}, \sqrt{ub(\{rs_1\})})$	helper for backward square ctc
itv.sqrt	✓	✓	✗	$\{r_d\} = \sqrt{\{rs_1\}}$	square-root of the selected itv
itv.exp	✓	✓	✗	$\{r_d\} = \exp(\{rs_1\})$	exponential of the selected itv
itv.log	✓	✓	✗	$\{r_d\} = \log(\{rs_1\})$	logarithm of the selected itv
itv.cos	✓	✓	✗	$\{r_d\} = \cos(\{rs_1\})$	cosine of the selected itv
itv.sin	✓	✓	✗	$\{r_d\} = \sin(\{rs_1\})$	sine of the selected itv
itv.acos	✓	✓	✗	$\{r_d\} = \arccos(\{rs_1\})$	arccosine of the selected itv
itv.asin	✓	✓	✗	$\{r_d\} = \arcsin(\{rs_1\})$	arcsine of the selected itv

Utilisation en langage C

- Le compilateur doit avoir connaissance des nouvelles instructions:
  - On s'attaque à *binutils* plutôt qu'à GCC.
  - Interêt de la toolchain open-source.



Utilisation en langage C

- Wrapping des instructions de *xinterval*.

```c
/* intervals are renamed double */
typedef double interval_t;

/* inline function to bind xinterval instruction itv.add */
inline interval_t __attribute__((always_inline))
itv_add(interval_t itv1, interval_t itv2) {
  interval_t result;
  asm("itv.add %0, %1, %2"
    : "=f"(result)
    : "f"(itv1), "f"(itv2));
  return result;
}
```

Utilisation en langage C

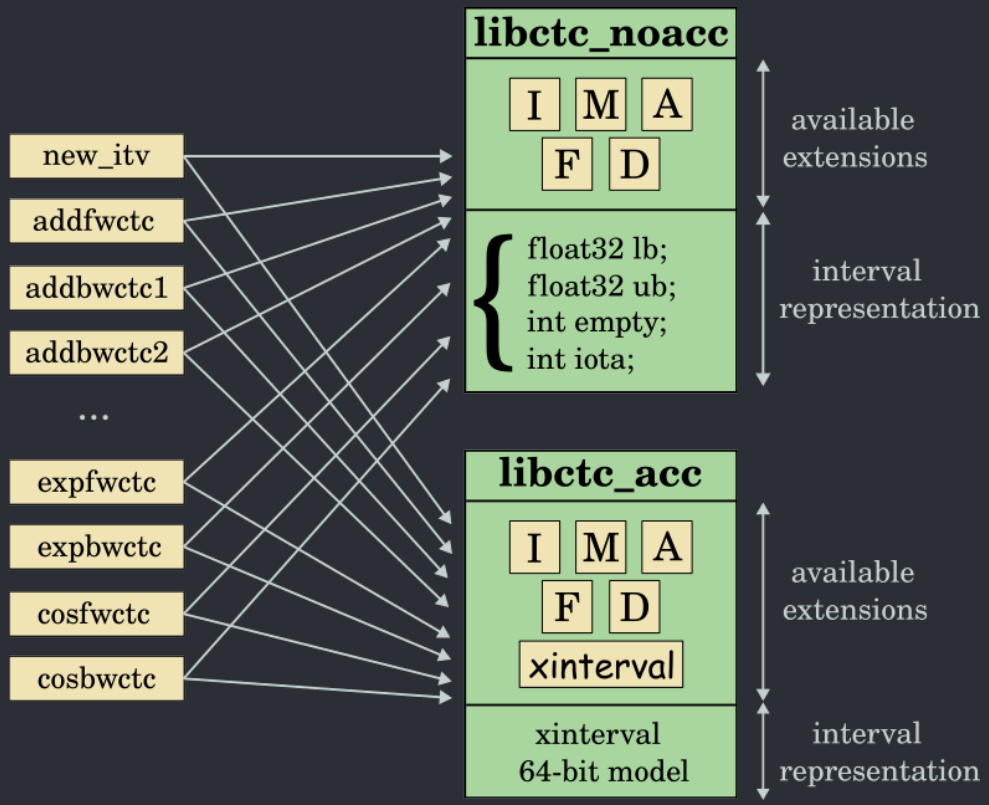
- On construit ensuite les contracteurs élémentaires.

```
/* Addition forward contractor */
interval_t addfwctc(interval_t x, interval_t y) {
    return itv_add(x, y);
}

/* Addition backward contractor 1 */
/* Given that Z = X + Y */
/* This function return Xc such as : */
/* Xc = X inter (Z - Y) */
interval_t addbwctc1(interval_t x, interval_t y, interval_t z) {
    interval_t subtraction = itv_sub(z, y);
    interval_t intersected = itv_intersection(x, subtraction);
    return intersected;
}
```

Création d'une Bibliothèque de Contracteurs

- Nécessaire pour les comparaisons à venir:
  - Interface commune, centrée autour des contracteurs.
  - 2 backends de calculs: avec ou sans *xinterval*.



Sommaire

Les difficultés liées à la validation

- On doit pouvoir tester et estimer les performances de *xinterval*.
- Problématique de l' architecte de jeu d'instructions:
  - Contours de l'extension non figés.
  - Les primitives matérielles évoluent.
  - L'interface logiciel/matériel et les toolchains aussi.

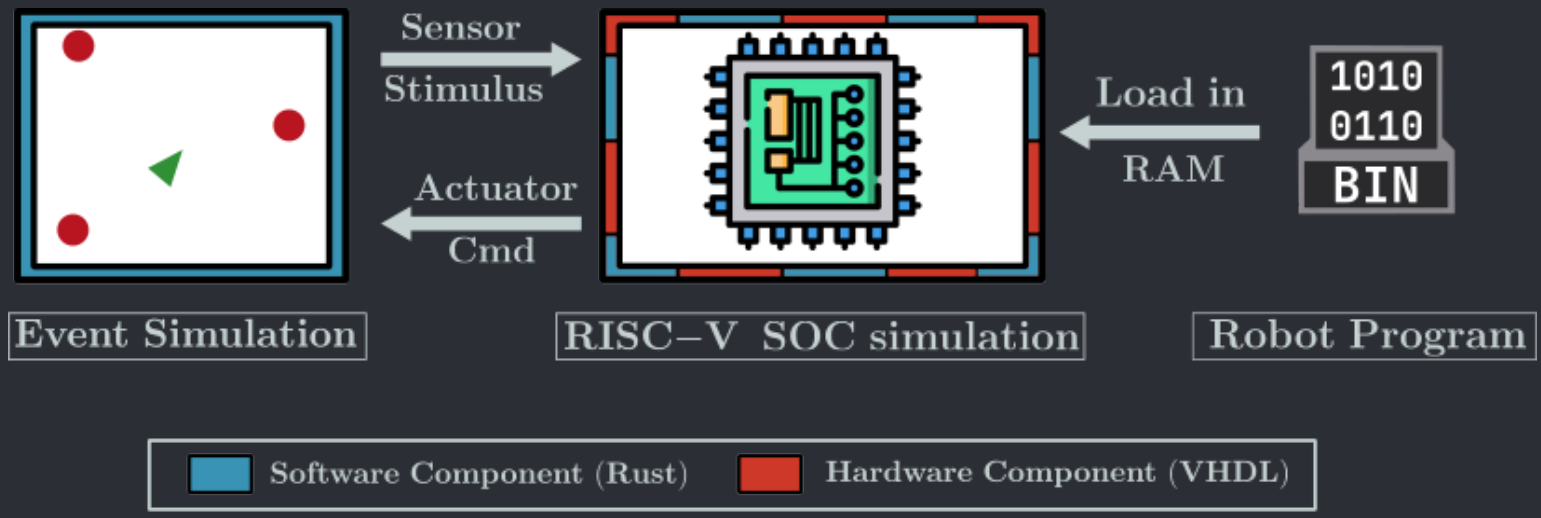
Défis à relever

- Espace de conception très étendu.
- Observabilité restreinte.

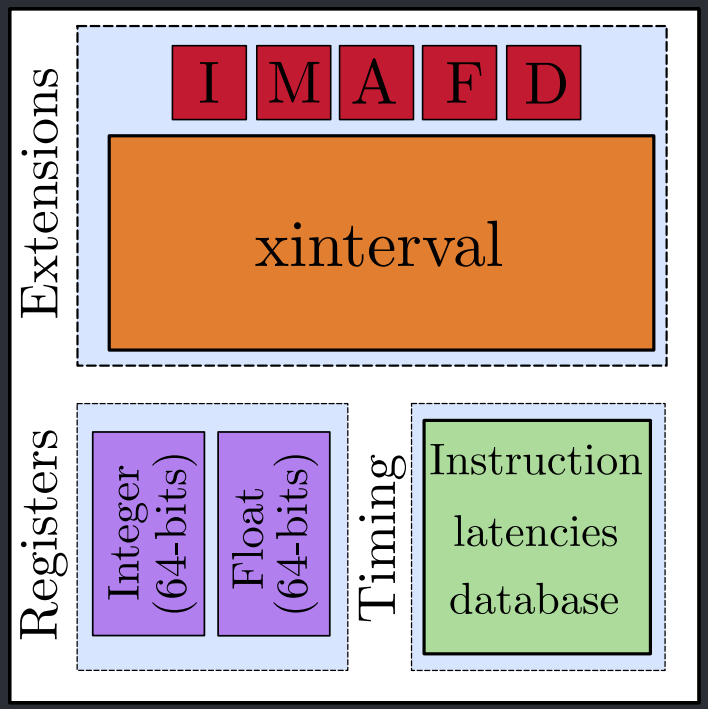
Utilisation du prototypage virtuel

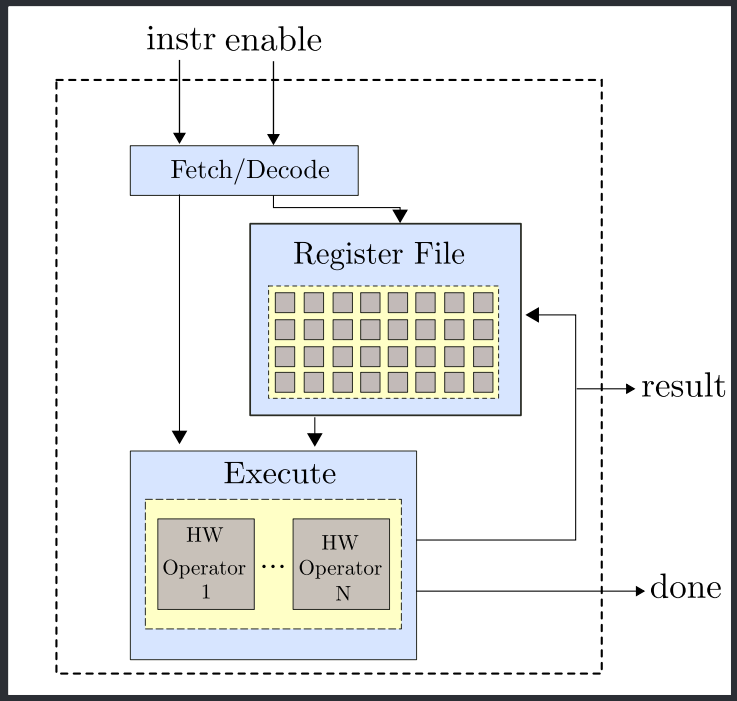
- Pas d'intégration dans un RISC-V synthétisé:
  - Approche peu flexible et techniquement complexe.
  - A utiliser dans un second temps !

- A la place, on a recours à un stratégie *Hardware-in-the-loop*.
  - Le *SoC* RISC-V est un hybride software/hardware.



Architecture du coeur RISC-V hybride





- Le modèle est constitué de deux parties:
  - Un simulateur d'ISA RISC-V pour les extensions standards.
  - Un coeur matériel *xinterval* minimaliste.

Evaluation des performances

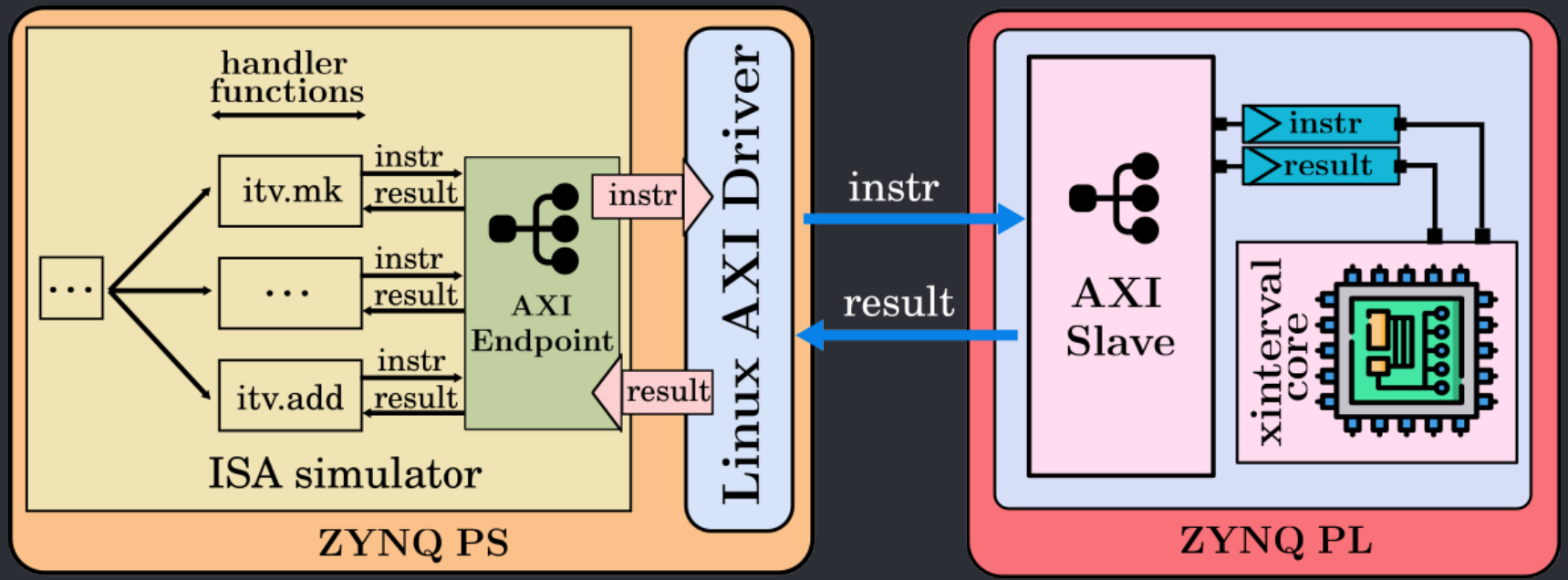
- Le simulateur d'ISA évalue la durée d'exécution du programme:
  - Comptage d'occurences par instruction.
  - Utilisation d'une table de latences.

Extension	**Methode de sélection**
I	Paramètres par défaut du RISC-V (Gem5)
M	Paramètres par défaut du RISC-V (Gem5)
A	Paramètres par défaut du RISC-V (Gem5)
F	Opérateur FloPoCo (obtenu en synthèse)
D	Opérateur FloPoCo (obtenu en synthèse)
Xinterval	Voir précédent

- Méthode imparfaite, mais suffisante en première approximation.

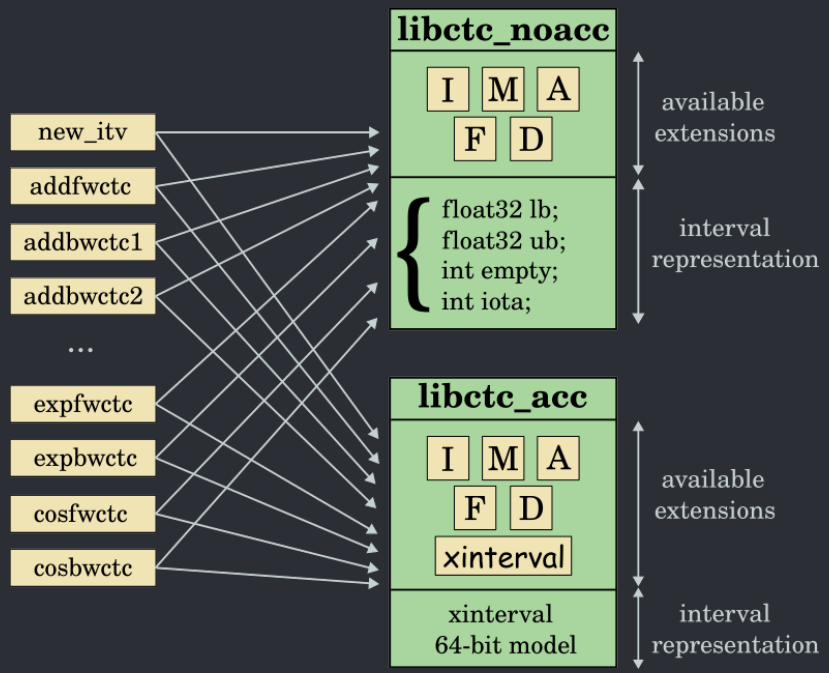
Déploiement sur une plateforme Zynq

- Utilisation d'une architecture Zynq:
  - Coeur ARM: ⇒ simulateur d'ISA (linux).
  - Puce FPGA ⇒ synthèse du coeur *xinterval*.
  - Echange de données via le bus AXI.



Un mot sur les benchmarks

- SIVIA sur des contracteurs axés robotiques.
  - Compilé pour nos deux backends de contracteurs.
- Benchmarks évalués dans les deux configurations:
  - RISC-V non accéléré: modèle d'intervalle classique.
  - RISC-V accéléré: modèle 64-bit de *xinterval*.



Benchmark 1: **ring**

$$\mathcal{H}: \begin{cases} X: (x_1, x_2) \\ D: ([-10, 10], [-10, 10]) \\ C: \mathbf{f}(X) \in [16, 25] \end{cases} \quad \text{where } \mathbf{f}(X) = (x_1 - 1)^2 + (x_2 - 2)^2$$

Features: Addition, Subtraction, Sqr

Benchmark 2: **waves**

$$\mathcal{H}: \begin{cases} X: (x_1, x_2) \\ D: ([-3.5, 2], [-3, 3]) \\ C: \mathbf{f}(X) \in [0.325, +\infty] \end{cases} \quad \text{where } \mathbf{f}(X) = \frac{\sin({x_1}^2 + {x_2}^2)}{e^{{x_1}^2 + {x_2}^2}}$$

Features: Addition, Sqr, Exponential, Sine

Benchmark 3: **donut**

$$\mathcal{H}: \begin{cases} X: (x_1, x_2) \\ D: ([-3, 3], [-3, 3]) \\ C: \mathbf{f}(X) \in [\text{-}0.2, 0.2] \end{cases} \quad \text{where } \mathbf{f}(X) = e^{x_1 \times x_2} - \sin(x_1 - x_2)$$

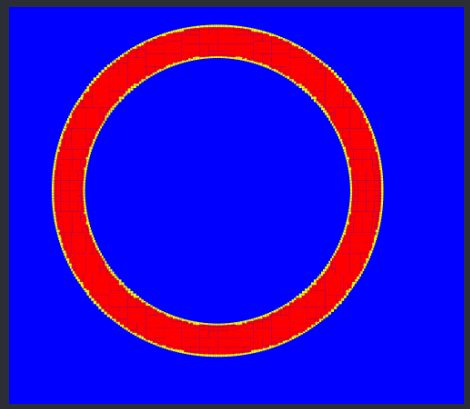
Features: Subtraction, Multiplication, Exponential, Sine

Benchmark 4: **vortex**

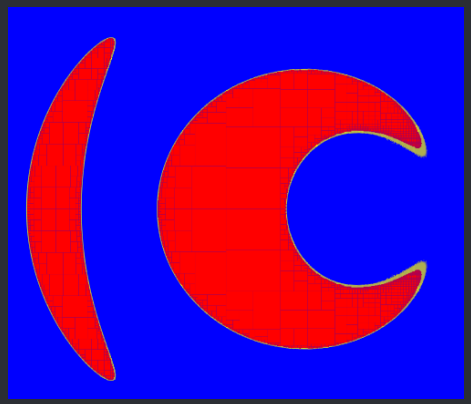
$$\mathcal{H}: \begin{cases} X: (x_1, x_2) \\ D: ([-3.5, 2], [-3, 3]) \\ C: \mathbf{f}(X) \in [0.325, +\infty] \end{cases} \quad \text{where } \mathbf{f}(X) = (x_2 - 5)\cos(4\sqrt{(x_1 - 4)^2) + {x_2}^2} - x_1 \sin(2\sqrt{{x_1}^2 + {x_2}^2})$$

Features: Addition, Subtraction, Multiplication, Sqr, Sqrt, Sine, Cosine

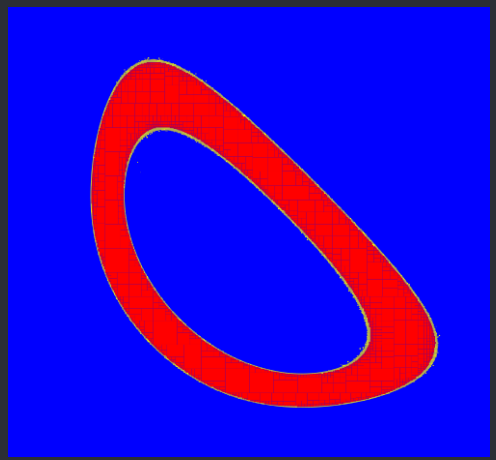
Résultats obtenus via prototypage virtuel



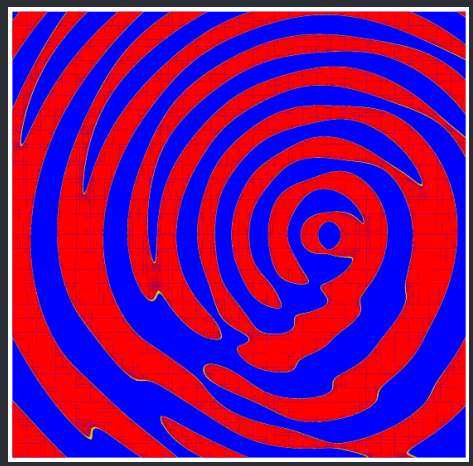
ring



waves



donut



vortex

Facteurs d'accélération obtenus

Benchmark	Speedup
ring	×3.2
waves	×7.3
donut	×8.2
vortex	×7.8

- Estimation pessimiste:
  - Dûe à la méthode d'évaluation.
  - Difficile d'évaluer par rapport à une baseline x86.
- Gains plus importants sur des contracteurs complexes.

Sommaire

Synthèse des travaux

- Viabilité d'une approche matérielle des intervalles.
  - Stratégie basée sur les contracteurs élémentaires.
  - Simulation d'une chaine complète.

- Une solution transparente pour le développeur:
  - Abstraction de la bibliothèque de contracteurs.
  - Permet la réutilisation de code existant.

- Evaluation des performances à consolider:
  - Méthode largement améliorable.
  - Intégration dans un RISC-V physique.
  - Comparatif plus poussé avec l'architecture x86.

Pistes pour des travaux futurs

- Contracteurs élémentaires validés en matériel.
  - Briques de base.
  - Idée: s'éloigner du compromis RISC-V.
- Deux approches sont envisageables:
  - Approche matérielle monolithique via HC4-R.
  - Architecture parallèle: array de processeurs *xinterval*.

Sommaire

Questions