

# Journée Interflop

Task 2 : avancement

A. Hamitouche

ANEO



Juin 2024



# Table of Contents

## 1 Proposition d'une interface vectorielle

## 2 Ajout de backends dans les outils InterFLOP

- Branchement sous forme de bibliothèques dynamiques dans PENE
- Ajout de backends par génération de code
- Mise en place de tests unifiés

## 3 Next steps



# Proposition d'une interface vectorielle

Principaux travaux réalisés en 2023 :

- proposition d'une interface vectorielle
- POC avec le backend Verrou en SSE et AVX pour les modes d'arrondis Nearest, Upward et Downward
- utilisation des backends sous forme de bibliothèques dynamiques dans PENE (Linux)



# Ajout de backends dans les outils InterFLOP



# Branchement sous forme de bibliothèques dynamiques dans PENE

## Linux :

- divergence entre PinCRT et la glibc
- non-prise en charge de la libquadmath par Pin
- non-prise en charge de la libargp par Pin

## Windows:

- divergence entre PinCRT et la glibc
- utilisation de helpers tel que ARPG\_PARSE spécifiques à Linux
- non-prise en charge de la quad précision par MSVC

# Ajout de backends par génération de code

# Structure de l'adaptateur

interflop\_backend\_adapter

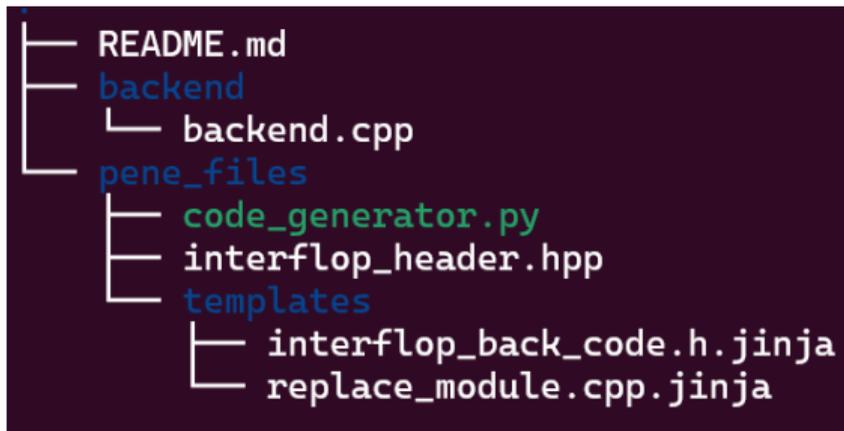


Figure: Arborescence du générateur de PENE

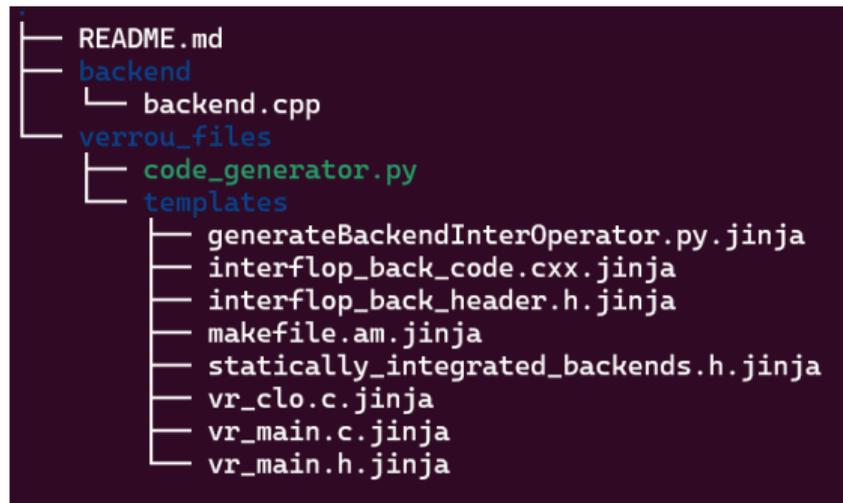


Figure: Arborescence du générateur de Verrou

# Implémentation d'un backend simple

```
add_float:  
    *cptr = a+b;  
end_add_float  
  
sub_float:  
    *cptr = a-b;  
end_madd_float  
...
```

Figure: Template d'un backend

- description/implémentation d'un backend
- utilisation de génération de code pour ajouter le backend au frontend cible

# Structure d'un backend : PENE

```
namespace pene
{
  namespace replace {
    namespace backend {
      struct {{ backend_name }}
      {
        static void add_float(float a, float b, float* res, void*) noexcept {
          {{ add_double_code }}
        }
        static void sub_float(float a, float b, float* res, void*) noexcept {
          {{ sub_float_code }}
        }
        static void mul_float(float a, float b, float* res, void*) noexcept {
          {{ mul_float_code }}
        }
        static void div_float(float a, float b, float* res, void*) noexcept {
          {{ div_float_code }}
        }
      }
    }
  }
};
```

Figure: Générateur de code avec la structure d'un backend dans PENE



# Mise en place de tests unifiés

- Tests réalisés : test des backends est réalisé par des test unitaires embarqués par chaque projet.
  
- Test manquant : s'assurer que le frontend a effectivement instrumenté le code cible

# Méthodologie de test mise en place : cas pratique

```
float apply_bitmask(float x, uint32_t mask, uint32_t bits) {
    uint32_t i;
    std::memcpy(&i, &x, sizeof(i));

    i &= ~mask;
    i |= bits;

    std::memcpy(&x, &i, sizeof(i));

    return x;
}
```

Figure: Modification des bits de poids fort du résultat

```
void add_float(float a, float b, float* cptr, void*) noexcept {
    uint64_t add_bits = 0x00000000;
    uint64_t mask = 0x0000000F;
    *cptr = apply_bitmask(a+b, mask, add_bits);
}

void sub_float(float a, float b, float* cptr, void*) noexcept {
    uint64_t sub_bits = 0x00000001;
    uint64_t mask = 0x0000000F;
    *cptr = apply_bitmask(a-b, mask, sub_bits);
}
}
```

Figure: Exemple sur l'addition et la soustraction IEEE



## Next steps

- un backend qui s'intègre des 3 côtés et qui permet de valider la complétude de l'instrumentation réalisée par le front-end
- une intégration partielle d'un backend vectorisé
- une intégration du delta debug avec PENE