InterFLOP

Vers la nécessité d'une interface SIMD dans InterFLOP

aneo

Réalisé par :
Akli HAMITOUCHE

Missions au sein d'InterFLOP

- Intégration d'une nouvelle bibliothèque d'analyse supportant le SIMD dans PENE
- Intégration de cette bibliothèque dans les outils d'interFLOP
- Définition d'une nouvelle API SIMD
  - Identification des contraintes d'intégration
  - Intégration des bibliothèques dans les outils existants
  - Validation de l'adéquation de la nouvelle API

❑ Intitulé de la mission

➢ Conception d'interfaces parallèles pour des outils d'analyse de stabilité numérique

Objectifs de la présentation

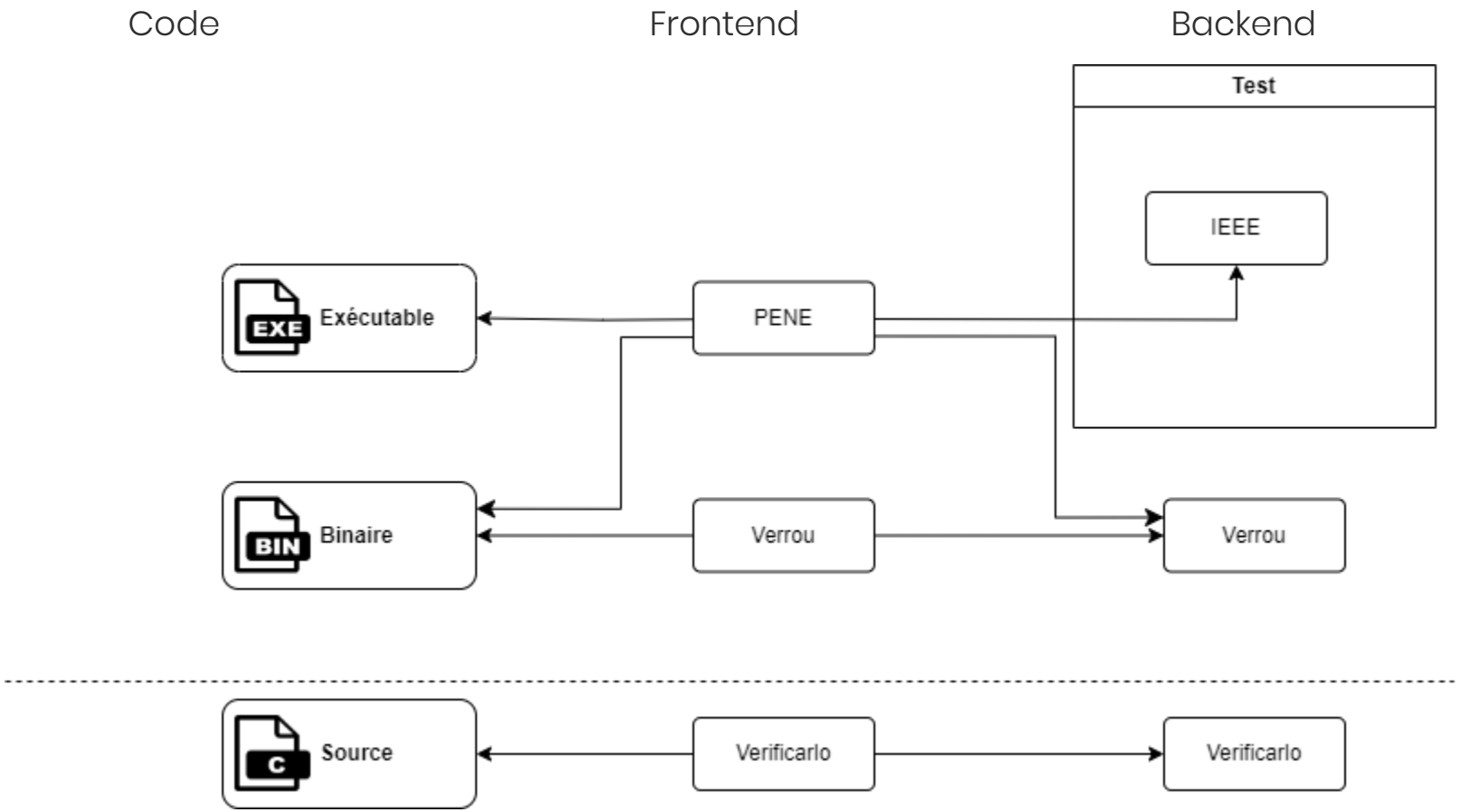
- Présenter le fonctionnement de l'interface Interflop
- Exposer les difficultés d'une interface SIMD simple
- Echanger à propos des interfaces SIMD
  - Niveau d'abstraction
  - La forme de l'interface
  - Rapport au matériel

❑ Enjeu/Intérêt:

- Gain de performance

Notion de frontend et de backend

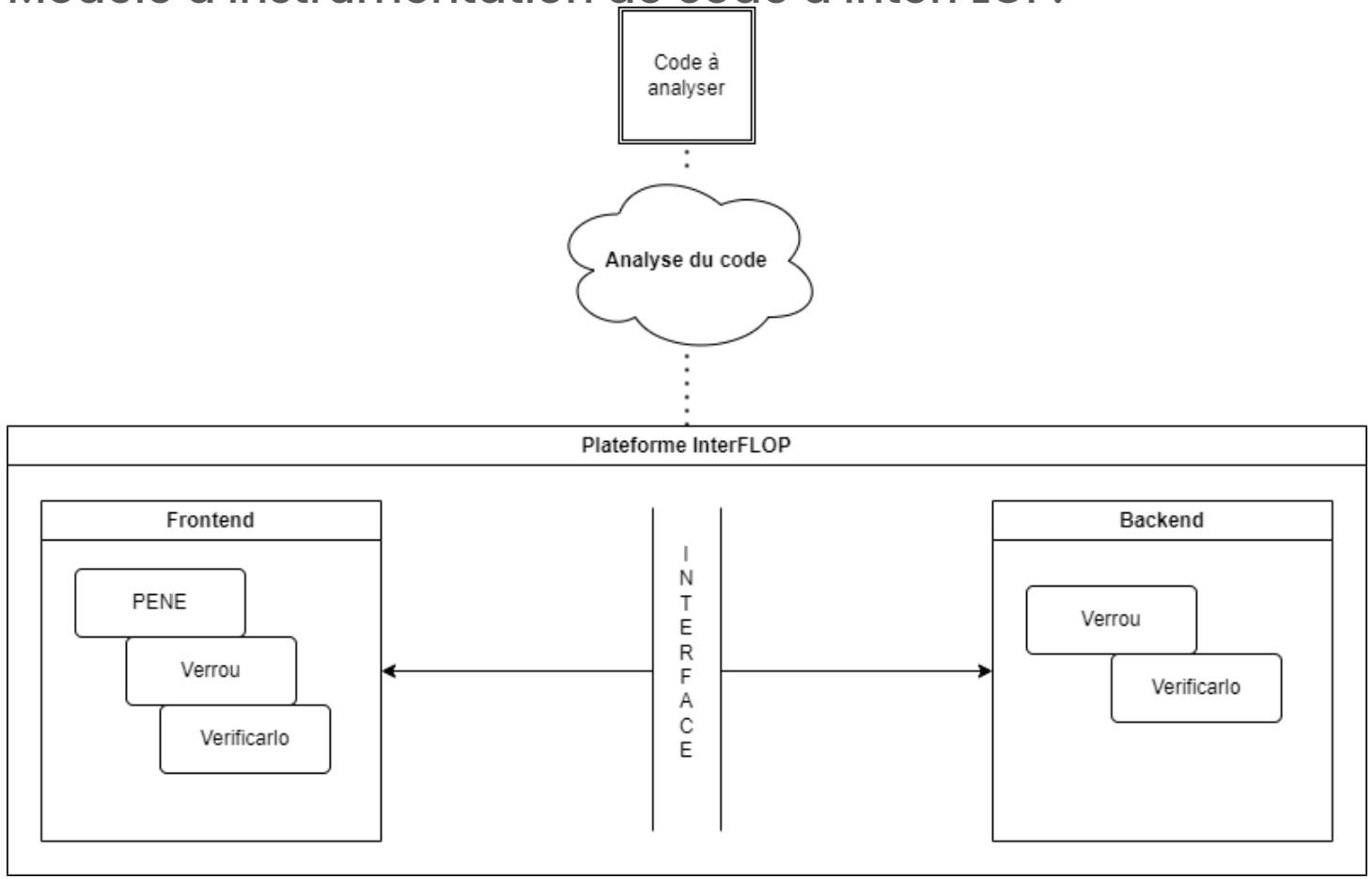
Fonctionnement des outils d'InterFLOP:



- Niveaux d'action:
  - Source
  - Binaire/Exécutable
- Modes d'action
  - Compilation
  - Exécution

Analyse dans InterFLOP

Modèle d'instrumentation de code d'InterFLOP:



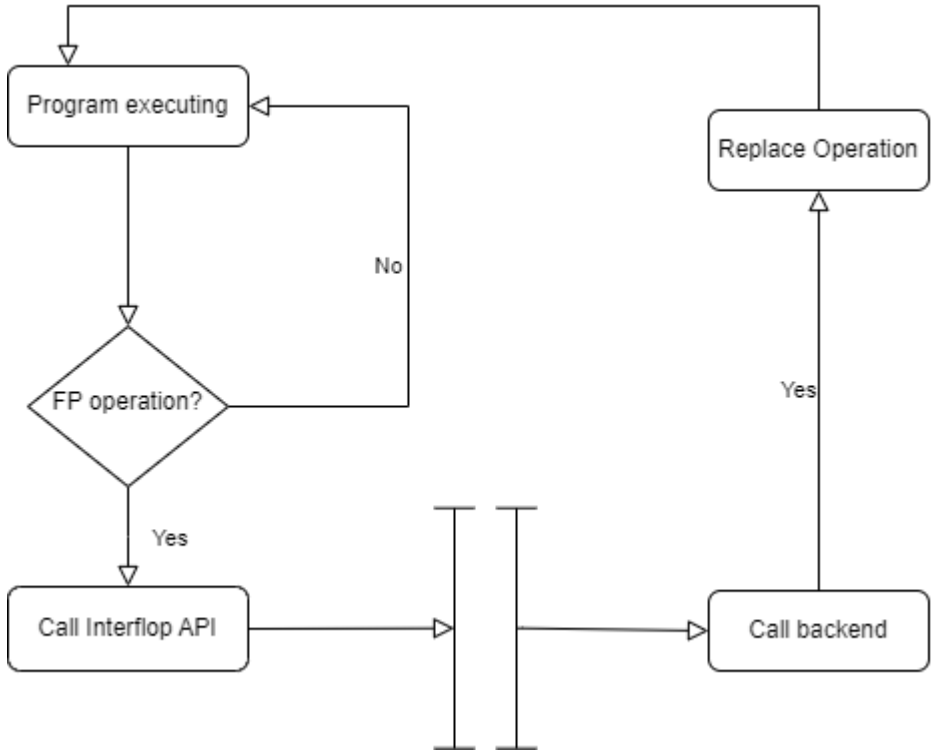
Fournit :

- Des frontends
- Des backends
- Une interface commune

Le but :

- Fournir une plateforme d'analyse de stabilité numérique

Fonctionnement d'un backend



L'instrumentation se fait à l'aide d'un **Frontend** et ce dernier map des fonctionnalités d'un **Backend** au travers d'une **Interface** définissant les opérations cibles

Structure de l'interface InterFLOP

L'interface définit :

- Le type d'opération
- Le type de donnée
- Les arguments à fournir au backend

```
/* interflop backend interface   */

struct interflop_backend_interface_t {
  void (*interflop_add_float)(float, float, float*, void*);
  ...
  void (*interflop_add_double)(double, double, double*, void*);
  ...
};
```

Signature de l'appel d'API pour les additions 32 et 64 bits

Implémentation par le backend

Le backend IEEE implémente les opérations définies par la norme IEEE 754

```
static void add_float(float a, float b, float* cptr, void*) noexcept
{
    *cptr = a + b;
}
```

Backend IEEE : *Addition 32 bits*

Le backend Verrou implémente des modes d'arrondis software sur des types scalaires

```
void verrou::add_float(float a, float b, float* res,void* context) {
  typedef OpWithSelectedRoundingMode<AddOp <float>  > Op;
  Op::apply(Op::PackArgs(a,b),res,context);
}
```

Backend VERROU : *Addition 32 bits*

Intérêt des interfaces

- Offrir l'abstraction nécessaire à la manipulation de type
- Minimiser l'effort de développement pour l'intégration d'un backend
- Permettre de conserver la cohérence des analyses

Intérêt d'une interface SIMD

- Gain de performance

Interface avec l'API Intel x86

- Utilisation d'intrinsics
  - Spécifique à une architecture cible
  - Offre des garanties sur la taille des registres utilisés et non sur l'extension utilisée

- Pour le backend IEEE, l'implémentation des opérations SIMD avec les intrinsics Intel a le même comportement qu'en scalaire

```
void addf32_4 (void * reg_a, void * reg_b, void * reg_c, void * ctx)
{
  __m128* xmm_a = (__m128 *) reg_a;
  __m128* xmm_b = (__m128 *) reg_b;
  __m128* xmm_c = (__m128 *) reg_c;
  *xmm_c = _mm_add_ps (*xmm_a, *xmm_b);
}
```

```
void addf32_8 (void * reg_a, void * reg_b, void * reg_c, void * ctx)
{
    __m256* ymm_a = (__m128 *) reg_a;
    __m256* ymm_b = (__m128 *) reg_b;
    __m256* ymm_c = (__m128 *) reg_c;
    *(ymm_c) = _mm256_add_ps (*ymm_a, *ymm_b);
  }
}
```

- Une version :
  - Pour chaque type primitif (double, float)
  - Pour chaque type de vecteur
  - Pour chaque architecture

Impact sur les performances du mix SSE-AVX

Une pénalité de 'false-dependency' impacte les performances lorsqu'on utilise à la fois les instructions AVX et SSE sur les processeurs à exécution désordonnée (out of order).

```
avx512_addf32_16(float*, float*, float*, void*):
        vmovups zmm0, ZMMWORD PTR [rdi]         .
        vaddps  zmm0, zmm0, ZMMWORD PTR [rsi]   .
        vmovaps ZMMWORD PTR [rdx], zmm0         .
        ret                                     .
```

Mix d'instructions legacy-SSE et AVX

- Perte de performance induite par un « context switch »

```
avx512_addf32_16(float*, float*, float*, void*):
        vmovups zmm0, ZMMWORD PTR [rdi]         .
        vaddps  zmm0, zmm0, ZMMWORD PTR [rsi]   .
        vmovaps ZMMWORD PTR [rdx], zmm0         .
        vzeroupper                              .
        ret                                     .
```

- Perte de performance pour palier aux fausses dépendances : vzeroupper

- Différence de compilation entre un frontend et un backend pouvant bloquer l'inlining des routines d'instrumentation

Abstraction + généricité

Implémentation de l'addition vectorielle basée sur l'extension SSE

- Addition de 4 flottant 32 bits en SSE

```
void addf32_4_sse (void * reg_a, void * reg_b, void * reg_c, void * ctx)
{
  __m128* xmm_a = (__m128 *) reg_a;
  __m128* xmm_b = (__m128 *) reg_b;
  __m128* xmm_c = (__m128 *) reg_c;
  *xmm_c = _mm_add_ps (*xmm_a, *xmm_b);
}
```

- Addition de 4 flottant 32 bits en SSE

```
void addf32_8_sse (void * reg_a, void * reg_b, void * reg_c, void * ctx)
{
  for (int i = 0; i < 2; i++)
    {
      __m128* xmm_a = (__m128 *) reg_a+i;
      __m128* xmm_b = (__m128 *) reg_b+i;
      __m128* xmm_c = (__m128 *) reg_c+i;
      *(xmm_c) = _mm_add_ps (*xmm_a, *xmm_b);
    }
}
```

Une interface SIMD

- Pour
- De la généricité
- Un gain de performance

- A rappeler que les suppositions faites sur les types primitifs ne peuvent être les mêmes qu'avec les registres vectoriels

- Un point particulier doit être fait pour le respect de l'ABI tout en garantissant des performances