Abstraction des structures de données

Impact sur le développement et les performances (vectorisation)

Sylvain Jubertie
sylvain.jubertie@{lri — univ-orleans}.fr

10 juin 2016

LIFO Plateforme MIReV3x

- 4 stations DELL
- 2 processeurs Intel E5-2650v2 2,6-3,4Ghz
- 8 coeurs / processeur
- unités vectorielles AVX (8 floats SP)
- mémoire 1866Mhz, 4 canaux
- GPUs Nvidia GTX780 2304 coeurs, 863Mhz

Puissance **théorique**

Calcul en simple précision :

- Processeurs : 4noeuds x 2proc. x 8coeurs x 8floats x 2units x 3Ghz = 3.072 GFLOPS
- GPUs : 4noeuds x 2304coeurs x 2FP/cycle (FMA) x 863Mhz = 15.906 GFLOPS

Bande passante mémoire :

- Processeur : 4canaux x 1866Mhz x 64bits = 59,7GB/s
- GPU : 288 GB/s

Programmation

Langages :

- C/C++, Fortran (éviter Python, Java, Matlab, . . .)

Bibliothèques :

- unités vectorielles AVX : intrinsics, compilateurs
- threads : pthreads, OpenMP, Cilk, . . .
- NUMA : libnuma
- MPI : OpenMPI, mpich, . . .
- Accélérateurs : Nvidia CUDA, OpenCL

Performance > combinaison

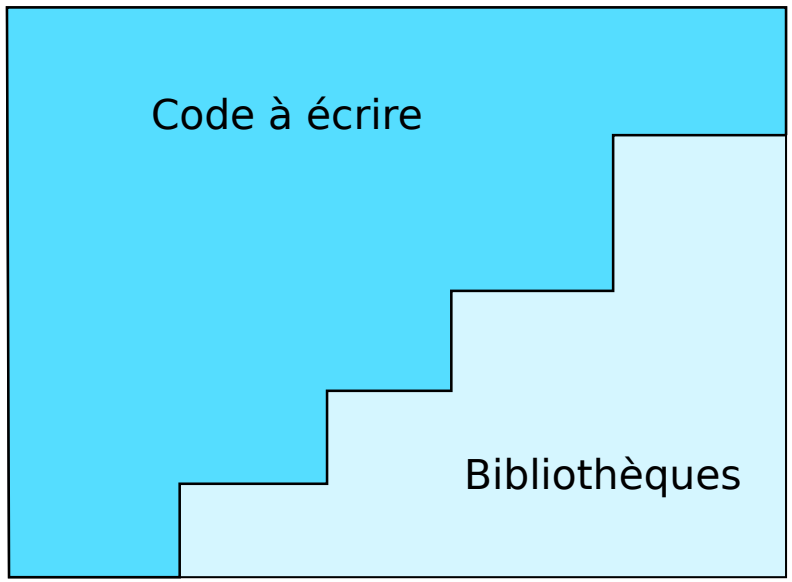
- SIMD + threads + NUMA + MPI
- CUDA + CPU + MPI

Limitations :

- Formation
- Complexité, temps de développement
- Lisibilité
- Portabilité
- Evolutivité

Schéma de développement

- Code utilisateur
- Appels bibliothèques : BLAS, FFT, ...
- Parallélisme explicite ou dans bibliothèque : cuFFT, PETSc, ...



Objectifs

- Séparer l'expression de l'algorithme/structures de données de son optimisation.
- Changer le schéma de développement

2 approches

- DSL : Domain Specific Language
- Transformations : Méta-programmation

DSL

- Exemples : Matlab, LateX, Make, ...
- Définir le langage du domaine : structures de données, fonctions
- Focalisation sur l'algorithme
- Pas de parallélisme apparent à ce niveau

```
1 image2DRGB img( ... );                    // lecture
2 image2DG img2 = grayscale( img );         // gris
3 sobel( img2 );                            // contours
4 stencil( ... );                           // convolution
5 ...
```

Limitations

- Copies et parcours inutiles
- Redéfinition pour chaque domaine

Exemple transformations

OSL : Orléans Skeleton Library

- Bibliothèque de squelettes algorithmiques parallèles C++, MPI
- map, reduce, ...
- Mécanisme de fusion (expression templates)

```
1  DArray a0( ... );
2  auto tmp = map( g, a0 );
3  tmp = map( f, tmp );
4  float x = reduce( std::plus, 0.0f, tmp );
```

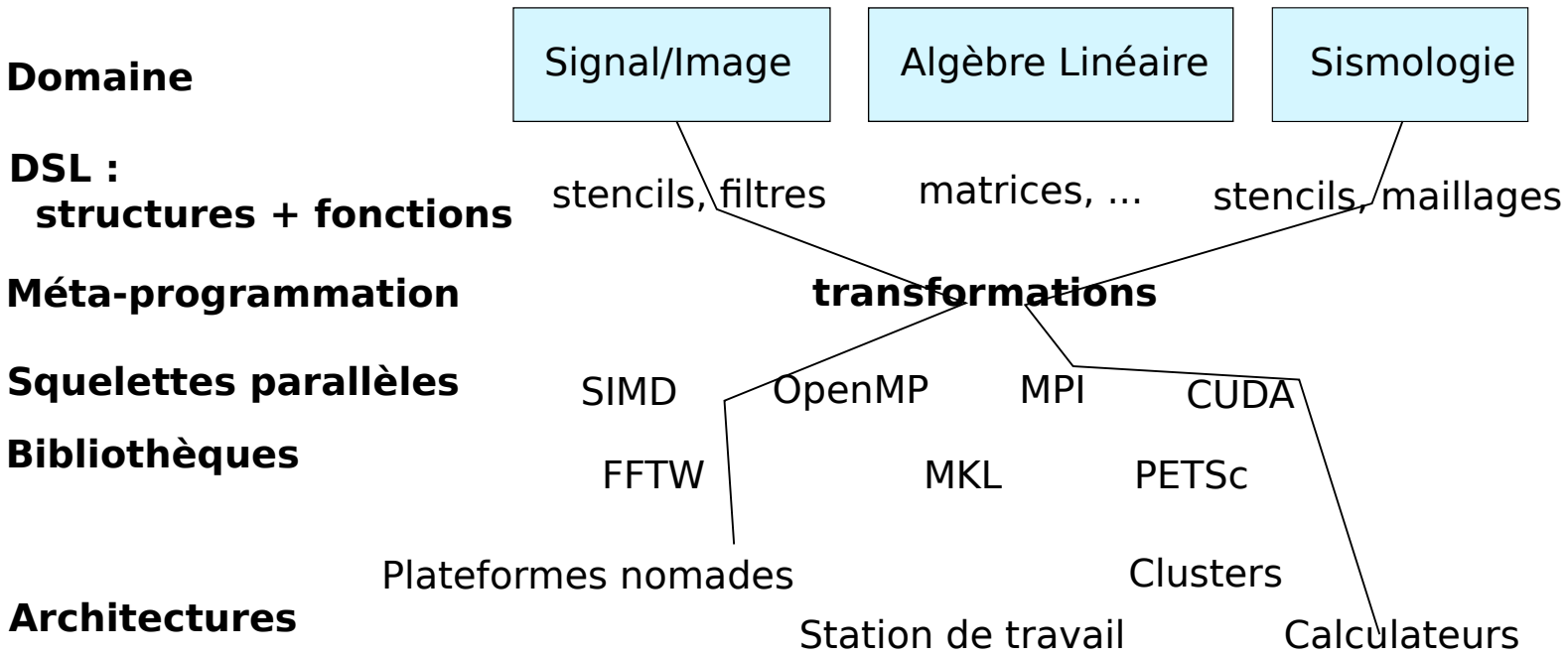
Code transformé

- Suppression du tableau tmp
- 1 seul parcours

```
1  float x = reduce( std::plus, 0.0f, (fog)(a0) );
2  // version sequentielle
3  // float x = 0;
4  // for( auto v: a0 )
5  //     x += f( g( x ) );
```

Schéma proposé

1. Utilisateur : DSL
2. Parallélisation : vectorisation, découpage
3. Optimisations bas-niveau



Problématique

- OOP : Représentation standard des structures de données AoS (Array of Structures).
- Architectures : Représentation adaptée SIMD SoA (Structure of Arrays) ou hybride.

```
1  struct PixelAoS { byte r, g, b; };
2  std::vector< PixelAoS > image;
3  struct ParticleAoS { float x, y, z, m, ...; };
4  std::vector< ParticleAoS > particles;
5
6  struct image { std::vector< byte > rs, ...; };
7  struct particles { std::vector< float > xs, ...; };
```

Structures

1. Buffers : tableau d'octets, localité, allocateurs
2. Conteneurs : dimensions, base index, stockage (C/Fortran), **layout**
3. Structures utilisateur : image2D, particules, ...

Implantation

- C++11/14, templates variadiques
- std :: tuple + std :: vector
- Surchage opérateur ()
- Layouts : aos, soa, hyb, combine

```
1  // RGBRGBRGBRGB...
2  using imageRGB = aos< unsigned char
3                      , unsigned char
4                      , unsigned char>;
5  // RRR..., GGG..., BBB..., AAA...
6  using imageRGBA = soa< unsigned char
7                       , unsigned char
8                       , unsigned char
9                       , unsigned char >;
10 // X..., Y..., Z..., M..., VX..., VY..., VZ...
11 using particles = soa< float, float, float
12                       , float
13                       , float, float, float >;
```

```
1  // Rx32Gx32Bx32Rx32Gx32Bx32...
2  using imageRGB = hyb< unsigned char, 3, 32 >;
3  // Rx32Gx32Bx32..., A...
4  using imageRGBA = combine<
5                          hyb< unsigned char, 3, 32 >
6                          , unsigned char >;
7  // Xx8Yx8Zx8..., M..., VXx8VYx8VZx8...
8  using particles = combine< hyb< float, 3, 8 >
9                             , float
10                            , hyb< float, 3, 8 > >;
```

```
 1  void grayscaleRGB( image const & v_in
 2                   , img_gray & v_out
 3                   , std::size_t const rows
 4                   , std::size_t const cols )
 5  {
 6    for( std::size_t i = 0 ; i < rows * cols ; ++i )
 7    {
 8      unsigned int gray = 307 * v_in( R, i )
 9                        + 604 * v_in( G, i )
10                        + 113 * v_in( B, i );
11      v_out[ i ] = gray >> 10;
12    }
13  }
```

```
1  using image = aos< byte, byte, byte >;
2  using image = soa< byte, byte, byte >;
3  using image = hyb< byte, 3, 32 >;
4
5  std::integral_constant< std::size_t, 2 > R;
6  std::integral_constant< std::size_t, 1 > G;
7  std::integral_constant< std::size_t, 0 > B;
```

```
1  using part = combine<
2                  hyb< float, 3, 8 >, // x, y, z
3                  float, // m
4                  hyb< float, 3, 8 > > // vx, vy, vz
```

Résultats

Modification de la structure, vectorisation à la charge du compilateur.

	GCC	GCC fastmath	ICC
normalize-raw-aos	1488	662	1545
normalize-abs-aos	1366	666	1374
normalize-raw-soa	1321	343	386
normalize-abs-soa	1348	368	1373
nbody-raw-aos	518	362	401
nbody-abs-aos	517	283	414
nbody-raw-soa	570	414	78
nbody-abs-soa	568	414	?

Résultats

Comparaison avec code écrit manuellement, sans/avec vectorisation explicite.

test	layout	ms (raw ms)
grayscale	aos	126 (150)
	soa	162 (idem)
	hyb	205 (140)
grayscale AVX	soa	45 (idem)
	hyb	45 (idem)
normalize	aos	425 (idem)
	soa	433 (idem)
normalize AVX	soa	219 (idem)
	hyb	246 (215)

Remarques

- Dépendance compilateur
- Dépendance architecture
- Dépendance ratio calcul/volume données

Conclusion

- Abstraction de l'organisation des données
- Solution pour portabilité SIMD : Boost.SIMD
- NT2 Numerical Template Toolkit : DSL Matlab-like
- Travaux en cours sur conteneurs
- Architectures distribuées en cours
- Automatisation choix transformation

Travaux connexes

- Relation Optimisation — Parallélisation — Consommation énergétique
- Métriques de complexité de codes — Effort de développement