Retour vers le futur des bibliothèques de squelettes algorithmiques et DSL

Sylvain Jubertie
sylvain.jubertie@lri.fr

Journée LaMHA - 26/11/2015

Squelettes algorithmiques

Squelettes algorithmiques

Objectifs

- Approche de haut-niveau pour la programmation parallèle.
- Masquer la complexité de la programmation parallèle.
- Synchronisation/communication.
- Composition de squelettes.
- Squelettes + structures de données.

Squelettes algorithmiques

Réduction optimisée en CUDA (M. Harris) :

```
template <unsigned int blockSize>
__device__ void warpReduce(volatile int *sdata, unsigned int tid) {
  if (blockSize >= 64) sdata[tid] += sdata[tid + 32];
  if (blockSize >= 32) sdata[tid] += sdata[tid + 16];
  if (blockSize >= 16) sdata[tid] += sdata[tid + 8];
  if (blockSize >= 8) sdata[tid] += sdata[tid + 4];
  if (blockSize >= 4) sdata[tid] += sdata[tid + 2];
  if (blockSize >= 2) sdata[tid] += sdata[tid + 1];
}
template <unsigned int blockSize>
__global__ void reduce6(int *g_idata, int *g_odata, unsigned int n) {
  extern __shared__ int sdata[];
  unsigned int tid = threadIdx.x;
  unsigned int i = blockIdx.x*(blockSize*2) + tid;
  unsigned int gridSize = blockSize*2*gridDim.x;
  sdata[tid] = 0;

  while (i < n) { sdata[tid] += g_idata[i] + g_idata[i+blockSize]; i += gridSize; }
  __syncthreads();
  if(blockSize >= 512){ if(tid < 256){ sdata[tid] += sdata[tid + 256]; } __syncthreads()
  if(blockSize >= 256){ if(tid < 128){ sdata[tid] += sdata[tid + 128]; } __syncthreads()
  if(blockSize >= 128){ if(tid < 64){ sdata[tid] += sdata[tid + 64]; } __syncthreads();
  if(tid < 32) warpReduce(sdata, tid);
  if(tid == 0) g_odata[blockIdx.x] = sdata[0];
}
```

Squelettes algorithmiques

Implantations

- Muesli
- SkePU
- QUAFF
- SkeTo
- SkelGIS
- OSL
- ...

Squelettes algorithmiques

Limitations

- jeu de squelettes + structures de données fixés.
- limitations à un domaine.
- optimisations logicielles : fusion de squelettes.
- optimisations matérielles : support des architectures.
- conception.
- performance.

Squelettes algorithmiques

Muesli :

```
void msl::InitSkeletons(int argc,
                        char** argv,
                        bool serialization) {
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD,
                &Muesli::MSL_numOfTotalProcs);
  MPI_Comm_rank(MPI_COMM_WORLD,
                &Muesli::MSL_myId);
  ...
}
```

Squelettes algorithmiques

SkeTo :

```
void make_edges() const {

  ...

  MPI_Barrier( MPI_COMM_WORLD );
}
```

Squelettes algorithmiques

SkePU :

```
// unary transpose operator
inline Matrix<T>& operator~() {
  ...
#if defined(SKEPU_CUDA)
  transpose_CU(Environment<int>::getInstance()->...
#elif  defined(SKEPU_OPENCL)
  transpose_CL(0);
#elif defined(SKEPU_OPENMP)
  transpose_OMP();
#else
  transpose_CPU();
#endif
  ...
}
```

Squelettes algorithmiques

Remarques

- Difficiles à maintenir/étendre.
- Séparer les aspects :
  - Squelettes fournis
  - Optimisations (fusion)
  - Support des architectures.
- Couche d'abstraction supplémentaire : DSL.

DSLs : Domain Specific Languages

Domain Specific Languages

Objectifs

- Fournir aux utilisateurs des langages adaptés à leurs domaines.
- Opérateurs/fonctions + structures de données.

```
Vector v1(...), v2(...);

auto v3 = map( plus, v1, v2 ); // squelettes.

auto v3 = v1 + v2; // DSL
```

Domain Specific Languages

Implantations

- LateX
- SQL
- HTML
- VHDL
- Matlab
- ...

Domain Specific Languages

DSL pour le HPC

- NT2
- Delite : OptiML, OptiQL, OptiGraph, OptiMesh
- . . .

Domain Specific Languages

NT2

- syntaxe Matlab.
- C++, méta-programmation.
- squelettes algorithmiques : transform, fold, scan.
- mémoire partagée : multithreading, vectorisation.
- accélérateurs.

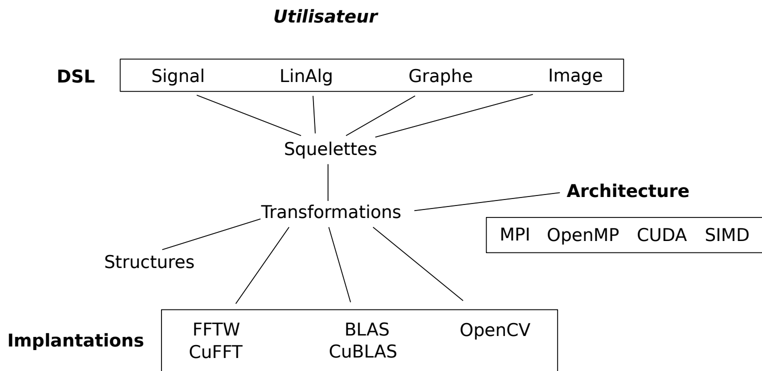
Domain Specific Languages

Delite

- Scala
- Squelettes algorithmiques.
- Mémoire partagée, accélérateurs.
- Mémoire distribuée : Apache Mesos, Google Protocol Buffers.
- Forge : méta-DSL.

Ce que l'on voudrait. . .

Architecture générale



Problèmes à résoudre

Données

- Génération de structures de données optimisées.
- Transformation des données AoS - SoA.
- Choix localité des données (NUMA, accélérateurs, distribué).
- Solution : tags, modèle de coût ?

```
Image image(...);                  // conversion AoS -> SoA ?
auto gray = grayscale(image); // RGBRGB. -> RR. GG. BB.

Matrix m1(...), m2(...);
auto m3 = m1 * m2; // sur CPU ou accélérateur ou ...  ?
```

Problèmes à résoudre

Aspects distribués

- Distribution des données.
- Bordures.
- Génération des synchronisations/communications.
- Recouvrement calcul/communication.
- Solution : allocation dynamique des bordures.

```
Image image(...);  // info sur les bordures ici ?
image.apply(/*stencil3x3*/); // allocation retardée ?
image.apply(/*stencil5x5*/); // reallocation ?
```

Problèmes à résoudre

Autres

- Interfaces des bibliothèques: allocation interne, type des paramètres, ...
- Portabilité : libnuma.
- Chaînes de compilation (nvcc), ...

```
cv::Mat m = cv::imread("image.png",
                       cv::IMREAD_UNCHANGED );
// allocation interne
auto * pointer = m.data; // pointeur aligné ?
```

Travaux en cours

Génération de structures de données optimisées

J. Falcou, I. Masliah, S. Jubertie - LRI

DSL pour la géologie, . . .

Thèse de G. Sornet - BRGM-LIFO - 2015-2018.

Génération de structures de données optimisées

Idée

1. Représentation utilisateur : AoS Image 2D pixels RGB
2. Représentation intermédiaire : SoA 3 tableaux 2D
3. Représentation mémoire : segment mémoire 1D

Allocateurs - Buffers - Conteneurs

- Allocateurs : alignement, NUMA, huge pages, accélérateurs.
- Conteneurs : localité, dimensions, . . .
- Optimisation vectorisation : SoA.
- Transferts mémoire centrale - accélérateurs.
- Distribution/communication.

Génération de structures de données optimisées

Allocators

- malloc, aligned, NUMA, CUDA, ...
- adaptateur allocateurs C++.
- retourne un Block.

Block

Type + unique_ptr + taille;

Génération de structures de données optimisées

Buffer

- capacité.
- iterateurs.
- redimensionnement.
- copie.

Génération de structures de données optimisées

AoS - SoA

Fournir une vision AoS à une structure de données SoA.

```
vectors< int, float, int > vecs;
vecs.add( 3, 4.0f, 5 );
vecs.add( 6, 7.3f, 8 );
vecs.add( 9, 10.2f, 11 );
...
using imageRGB = vectors< unsigned char,
                          unsigned char,
                          unsigned char >;
```

Conclusion

Existant

- Beaucoup d'approches différentes/similaires.
- Pas de factorisation.
- Limitations liées à la conception (squelettes).
- Beaucoup de DSL ad hoc.
- Delite, Forge : unification ?

Conclusion

Travaux en cours

- Reconstruction par le bas :
  1. bloc mémoire,
  2. structure de données interne,
  3. distribution,
  4. structure de données exposée à l'utilisateur.
- Ce n'est que le début !

Questions, suggestions, idées, . . . ?