Algèbre d'Ensembles sur Représentation CSR d'Intervalles

Implémentation Haute Performance avec Kokkos

Sébastien DUBOIS

Équipe HPC@Maths

Décembre 2025

Plan

I. Contexte : GPU & Kokkos

Architecture GPU — Massively Parallel

Hiérarchie d'exécution

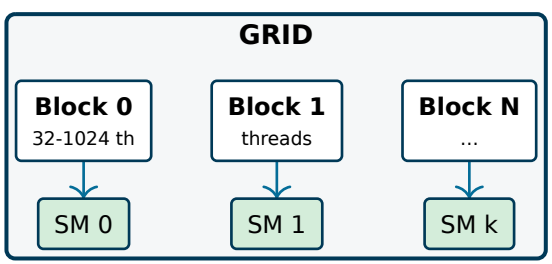
```
GPU
└── SM (Streaming Multiprocessor) ×N
    └── Warps ×64 par SM
        └── Threads ×32 par warp (SIMT)
```

- **Warp** = 32 threads exécutés **en lockstep**
- **SM** = unité de calcul autonome
- Plusieurs warps actifs par SM (latency hiding)

Comparaison B200 vs EPYC 9965

	GPU B200	CPU EPYC 9965
Cœurs	148 SM	192 cores
Mémoire	192 GB HBM3e	jusqu'à 6 TB DDR5
Bandwidth	**8 TB/s**	576 GB/s
FP32	**80 TFlops**	14 TFlops

Modèle d'exécution



Pour notre projet

- **1 thread** = traite 1 ligne Y (ou 1 cellule)
- Des milliers de lignes → **saturent le GPU**

GPU : **14× plus de bandwidth** que CPU
→ idéal pour grands maillages

Kokkos — Portabilité Performance

Le problème

- CUDA = NVIDIA only
- OpenMP = CPU only (GPU limité)
- HIP = AMD only
- Réécrire pour chaque plateforme ?

La solution : Kokkos

```
// 1. COUNT — taille résultat inconnue
parallel_for(num_rows, KOKKOS_LAMBDA(int r) {
  counts[r] = count_intervals(r);
});
// 2. SCAN — calcul des offsets
exclusive_scan(counts, row_ptr);
// 3. FILL — écriture parallèle
parallel_for(num_rows, KOKKOS_LAMBDA(int r) {
  fill_intervals(r, &out[row_ptr[r]]);
});
```

CUDA vs Kokkos

CUDA natif

```
// Allocation
double* d_data;
cudaMalloc(&d_data, n*8);

// Copie Host → Device
cudaMemcpy(d_data, h_data,
  n*8, HostToDevice);

// Kernel
kernel<<<B,T>>>(d_data, n);

// Copie Device → Host
cudaMemcpy(h_data, d_data,
  n*8, DeviceToHost);

// Libération
cudaFree(d_data);
```

Kokkos

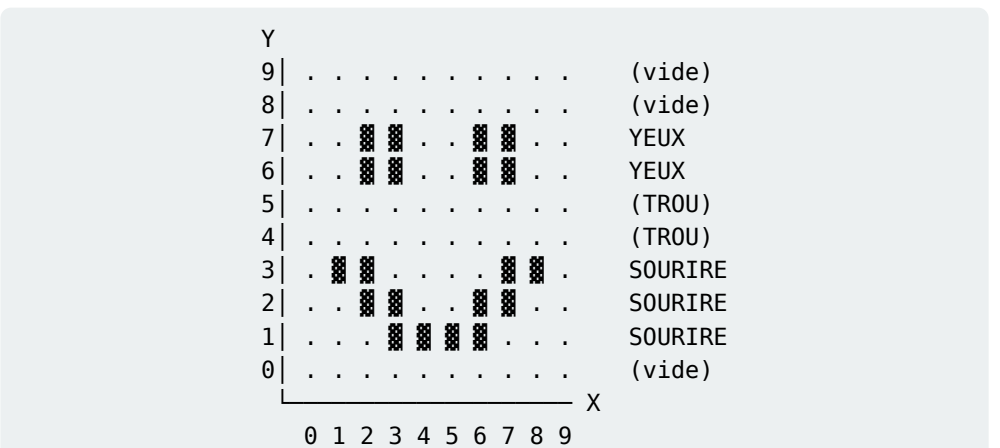
```
// Allocation + miroir auto
View<double*> data("d", n);
auto h_data = create_mirror_view(data);

// Copie Host → Device
deep_copy(data, h_data);

// Parallel (CPU ou GPU)
parallel_for(n, KOKKOS_LAMBDA(int i){
  data(i) = compute(i);
});

// Copie Device → Host
deep_copy(h_data, data);

// Libération automatique (RAII)
```

Un code source unique → compile pour OpenMP, CUDA, HIP, SYCL, Serial — spécialisable si nécessaire

II. Représentation Sparse

Exemple : Maillage 2D Sparse avec Intervalles

Géométrie “Smiley” :-)



Complexité mémoire

O(R + I) — R = lignes Y, I = intervalles

Dense O(W×H) vs CSR O(R+I) ≪

Représentation CSR

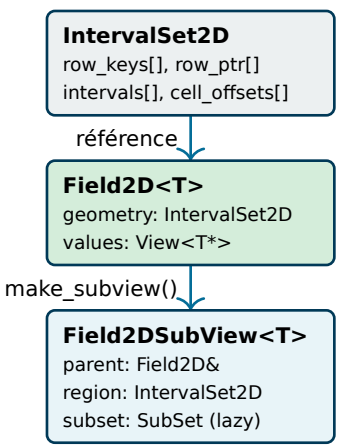
```
// 5 lignes, TROU Y=4,5
row_keys = [1, 2, 3, 6, 7]  // saute 4,5!
num_rows = 5

// Lignes avec 1 ou 2 intervalles
row_ptr = [0, 1, 3, 5, 7, 9]

intervals = [
  {3, 7},         // Y=1: sourire bas
  {2, 4}, {6, 8},// Y=2: sourire épais
  {1, 3}, {7, 9},// Y=3: sourire coins
  {2, 4}, {6, 8},// Y=6: YEUX bas
  {2, 4}, {6, 8},// Y=7: YEUX haut
]
num_intervals = 9

cell_offsets = [0,4,6,8,10,12,14,16,18,20]
total_cells = 20
```

Trou Y=4,5 : row_keys saute de 3 à 6

III. Structures de Données

Vue d'Ensemble — Structures Device

Architecture des types GPU



IntervalSubSet2D (intersection)

```
// Décrit l'intersection parent ∩ region
struct IntervalSubSet2D {
  interval_indices[];  // → parent
  x_begin[], x_end[];  // sous-plages
  row_indices[];       // index lignes
};
```

Workflow GPU typique

```
// Champs sur même géométrie
Field2D<Real> rho(fluid_geom);
Field2D<Real> rhou(fluid_geom);

// Zone d'intérêt (overlap, guard, etc.)
auto overlap = set_intersection_device(
    field.geometry, other_level, ctx);

// SubView = référence légère
auto sub_rho = make_subview(rho, overlap);
auto sub_rhou = make_subview(rhou, overlap);

// Opérations sur la zone uniquement
fill_subview_device(sub_rho, 0.0, &ctx);
copy_subview_device(dst, src, &ctx);

// Stencil, AMR restrict/prolong...
restrict_field_subview_device(coarse, fine);
prolong_field_subview_device(fine, coarse);
```

Pourquoi SubView ?

Avantage	Description
Non-owning	Pas de copie, juste des références
Lazy subset	Intersection calculée à la demande
Cache ctx	SubSet réutilisé entre opérations
API unifiée	fill, copy, stencil, AMR...

IntervalSet2D — Structure CSR Complète

Définition C++

```
template<class MemorySpace>
struct IntervalSet2D {
  // Coordonnées Y des lignes non-vides
  View<RowKey2D*> row_keys;  // [num_rows]

  // Index dans intervals[] pour chaque ligne
  View<size_t*> row_ptr;     // [num_rows + 1]

  // Tous les intervalles (contigus)
  View<Interval*> intervals; // [num_intervals]

  // Offset linéaire des cellules
  View<size_t*> cell_offsets;// [num_intervals]

  size_t total_cells;
  int num_rows;
  int num_intervals;
};
```

Invariants

- `row_keys` trié par Y croissant
- Intervalles triés par X dans chaque ligne
- Pas de chevauchement entre intervalles
- `row_ptr[r+1] - row_ptr[r]` = nb intervalles ligne r

Accès aux intervalles d'une ligne

```
// Intervalles de la ligne r
int begin = row_ptr[r];
int end   = row_ptr[r + 1];
for (int i = begin; i < end; i++) {
  Interval iv = intervals[i];
  // Traiter [iv.begin, iv.end)
}
```

Template MemorySpace : Device ou Host

Field2D — Champ sur Géométrie Creuse

Définition

Associe une **valeur** à chaque cellule sparse

```
template<class T, class MemorySpace>
struct Field2D {
  IntervalSet2D geometry;  // Réf géométrie
  View<T*> values;         // [total_cells]

  // Accès à une valeur
  T& at(interval_idx, x) {
    offset = cell_offsets(interval_idx);
    x0 = intervals(interval_idx).begin;
    return values(offset + x - x0);
  }
};
```

Stockage mémoire

```
Géométrie: ███ ░ ███ ░ ████
values[]:  [v0 v1 | v2 v3 | v4 v5 v6]
            ↑       ↑       ↑
offsets:    0       2       4
```

Valeurs **contiguës** → cache-friendly

Opérations sur Fields

```
// Algèbre élément par élément
field_add(a, b, result);       // a + b
field_sub(a, b, result);       // a - b
field_mul(a, b, result);       // a * b
field_axpby(α, a, β, b, r);    // αa + βb

// Réductions globales
T sum  = field_reduce_sum(f);
T dot  = field_dot(a, b);      // Σ aᵢbᵢ
T norm = field_norm_l2(f);     // √(Σ fᵢ²)
T min  = field_min(f);
T max  = field_max(f);
```

Implémentation (Kokkos std-like)

```
// Utilise transform, reduce, etc.
Kokkos::Experimental::transform(
  exec, a.values, b.values, result.values,
  KOKKOS_LAMBDA(T x, T y) { return x + y; }
);
```

Pour opérer sur une **zone spécifique** → SubSet

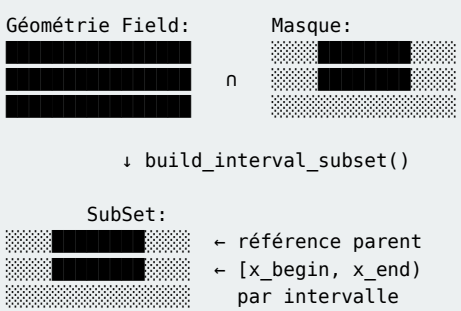
SubSet — Opérations sur Zones Ciblées

Problème

Comment appliquer une opération **uniquement** sur une zone ?

- Conditions aux limites (bords)
- Zone source (injection d'énergie)
- Mise à jour partielle

Solution : SubSet = Intersection



Pas de copie — référence la géométrie parente

Structure

```
struct IntervalSubSet2D {
  IntervalSet2D parent;  // Géométrie ref
  interval_indices[];    // Index intervalles
  x_begin[], x_end[];    // Sous-plages X
  row_indices[];         // Index lignes
  num_entries;
  total_cells;
};
```

Utilisation

```
// Construire le subset (intersection)
build_interval_subset(
  field.geometry, mask, subset);

// Opérations sur la zone uniquement
fill_on_subset(field, subset, 0.0);
scale_on_subset(field, subset, 2.0);

// Functor personnalisé
apply_on_subset(field, subset,
  KOKKOS_LAMBDA(x, y, val, idx) {
    val = source(x, y);
  });
```

GPU-friendly : parallélisme sur les entries

Workspace & Support AMR

UnifiedCsrWorkspace

Pool de buffers réutilisables

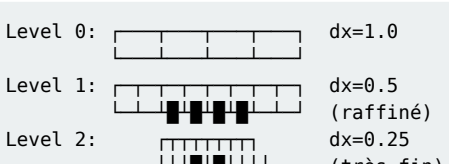
```
struct UnifiedCsrWorkspace {
  View<int*> int_bufs_[5];
  View<size_t*> size_t_bufs_[2];
  View<RowKey2D*> row_key_bufs_[2];
  View<Interval*> interval_buf_0;

  auto get_int_buf(int id, size_t n) {
    if (n > int_bufs_[id].extent(0))
      Kokkos::resize(int_bufs_[id], n*1.5);
    return subview(int_bufs_[id], {0,n});
  }
};
```

Évite allocations répétées GPU
Crucial pour chaînage d'opérations

MultilevelGeo (AMR)

Grilles multi-résolution

```
template<class MemorySpace>
struct MultilevelGeo {
  double origin_x, origin_y;
  double root_dx, root_dy;
  int num_active_levels;
  Array<GeoView, 16> levels;

  double dx_at(int level) {
    return root_dx / (1 << level);
  }
};
```



IV. Algorithmes

Constructeurs de Géométrie

Formes primitives

```
// Rectangle
auto box = make_box_device(
  Box2D{0, 100, 0, 50});

// Disque
auto disk = make_disk_device(
  Disk2D{cx, cy, radius});

// Damier
auto checker = make_checkerboard_device(
  domain, cell_size);
```

À partir de données

```
// Depuis bitmap (masque binaire)
auto geo = make_bitmap_device(
  bitmap_view, corner_x, corner_y);

// Aléatoire (test/benchmark)
auto rand = make_random_device(
  domain, fill_prob, seed);
```

Pattern : build_interval_set_from_rows

```
template<class Compute, class Fill>
IntervalSet2D build_interval_set_from_rows(
  int num_rows, Compute compute, Fill fill)
{
  // 1. COUNT (parallel)
  parallel_for(num_rows, [&](int r) {
    counts[r] = compute(r).num_intervals;
  });

  // 2. SCAN → row_ptr
  exclusive_scan(counts, row_ptr);

  // 3. FILL (parallel)
  parallel_for(num_rows, [&](int r) {
    fill(r, &intervals[row_ptr[r]]);
  });

  // 4. SCAN → cell_offsets
  compute_cell_offsets(intervals, offsets);
}
```

Pattern **Count-Scan-Fill** : allocation exacte

Algèbre Ensembliste — Opérations Binaires

Opérations supportées

```
set_union_device(A, B, out, ctx);
set_intersection_device(A, B, out, ctx);
set_difference_device(A, B, out, ctx);
set_symmetric_difference_device(...);
```

Algorithme Two-Pointer

- **O(n + m)** par ligne
- Parallélisme sur les **lignes**
- Template `CountOnly` : même code count/fill

Architecture unifiée

```
template<class RowOp>
void apply_binary_csr_operation(
  A, B, mapping, output, row_op)
{
  int num_out_rows = mapping.num_rows;

  // Phase 1: COUNT
  parallel_for(num_out_rows, [&](int r) {
    counts[r] = row_op.count(r, A, B);
  });

  // Phase 2: SCAN
  exclusive_scan(counts, row_ptr);

  // Phase 3: FILL
  parallel_for(num_out_rows, [&](int r) {
    row_op.fill(r, A, B, out_intervals);
  });
}
```

`RowOp` : UnionRowOp, IntersectionRowOp…

Opérations sur Champs

Algèbre élément par élément

```
// Binaires
field_add_device(a, b, result);
field_sub_device(a, b, result);
field_mul_device(a, b, result);

// Combinaison linéaire: α·a + β·b
field_axpby_device(alpha, a, beta, b, result);

// Réductions
T dot  = field_dot_device(a, b);
T norm = field_norm_l2_device(a);
```

Implémentation (Kokkos std algorithms)

```
void field_add_device(a, b, result) {
  Kokkos::Experimental::transform(
    exec_space,
    a.values, b.values, result.values,
    KOKKOS_LAMBDA(T x, T y) {
      return x + y;
    });
}
```

Opérations sur sous-ensembles

```
// Remplir avec une valeur
fill_on_subset_device(field, subset, val);

// Multiplier par un scalaire
scale_on_subset_device(field, subset, k);

// Copier
copy_on_subset_device(src, dst, subset);
```

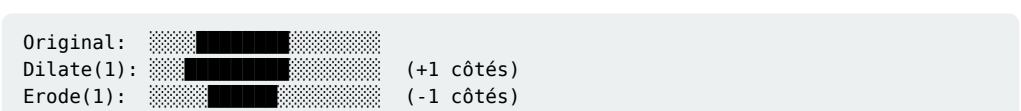
Implémentation TeamPolicy

```
TeamPolicy policy(num_entries, AUTO);
parallel_for(policy, KOKKOS_LAMBDA(team) {
  int e = team.league_rank();
  Coord x0 = subset.x_begin[e];
  Coord x1 = subset.x_end[e];

  TeamThreadRange(team, x0, x1,
    [&](Coord x) {
      field.at(e, x) = value;
    });
});
```

Morphologie Mathématique & AMR

Dilatation / Érosion

```
// Union N-way avec décalage ±radius
row_n_way_union_impl(rows[], radius, out)

// Intersection N-way avec shrink
row_n_way_intersection_impl(rows[], r, out)
```

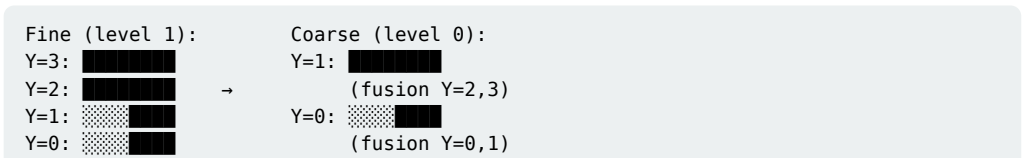


Extension 2D

- Considérer lignes y-r à y+r
- Fusionner avec opération N-way
- Structuring element implicite (carré)

Opérations AMR

```
// Coarsening: fin → grossier
build_row_coarsen_mapping(fine, ws)
// y_coarse = y_fine / 2, fusionner X

// Refinement: grossier → fin
refine_level_up_device(coarse, ws)
// [a,b) → [2a, 2b), doubler Y
```



Transfert de champs

```
// Projection fine → coarse (moyenne)
// Prolongation coarse → fine (interp)
build_amr_interval_mapping(coarse, fine)
```

V. Démonstration

Mach2 Cylinder — Simulation AMR Multi-Niveaux

Description

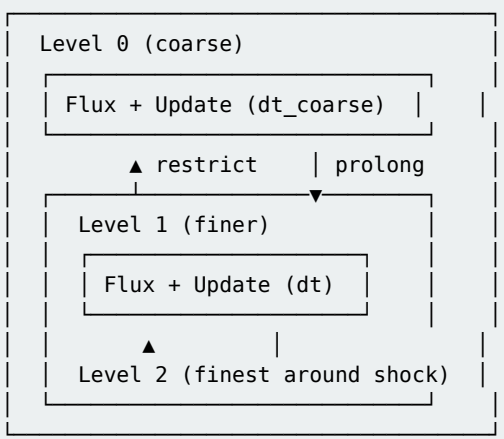
Simulation d'écoulement compressible 2D :

- **Mach 2** supersonique autour d'un cylindre
- Schéma Godunov 1er ordre + flux Rusanov
- **AMR dynamique** : jusqu'à 6 niveaux

Utilisation de Subsetix

```
// Géométrie fluide = domaine - obstacle
auto fluid = set_difference_device(
  make_box_device(domain),
  make_disk_device(cylinder),
  ctx);

// Champs conservés (ρ, ρu, ρv, E)
Field2DDevice<Real> rho(fluid);
Field2DDevice<Real> rhou(fluid);
// ...
```

Architecture AMR



Raffinement dynamique

- Indicateur : gradient de densité
- `expand_device()` pour zones de garde
- Remaillage tous les N pas de temps

Mach2 Cylinder — Résultats & Visualisation

Outputs générés

```
output/
├── fluid_geometry.vtk
├── obstacle_geometry.vtk
├── level_0_density_0000.vtk
├── level_0_density_0050.vtk
├── level_1_density_0050.vtk
├── level_2_density_0050.vtk
└── ...
```

Commande d'exécution

```
./mach2_cylinder \
  --nx 400 --ny 160 \
  --radius 20 \
  --mach-inlet 2.0 \
  --max-steps 1000 \
  --output-stride 50 \
  --amr
```

Phénomènes observés

- **Choc d'étrave** (bow shock) devant le cylindre
- Zone subsonique dans le sillage
- Allée de **Von Kármán** (vortex)
- Raffinement automatique près du choc

Points techniques clés

- Stencil CSR : `apply_csr_stencil_on_set_device()`
- Struct-of-Arrays pour cache efficiency
- `prolong_guard_from_coarse()` : interpolation
- `restrict_fine_to_coarse()` : conservation
- Export VTK multi-niveau pour ParaView

Sparse : calcul uniquement sur cellules fluides !

Démonstration Live

Live Demo

Construction
- Box, Disk, Bitmap
- Différence (obstacle)
- Affichage CSR

Opérations
- Union / Intersection
- Field algebra
- Stencil

Mach2
- Lancement simulation
- Visualisation ParaView
- AMR en action

Démonstration en direct...

Merci !

Questions ?

Points clés

- Représentation CSR d’intervalles
- Pattern Count-Scan-Fill
- Parallélisme Kokkos (CPU/GPU)
- Workspace pour réutilisation mémoire
- AMR multi-niveaux (Mach2)

Contact

Sébastien DUBOIS

Équipe HPC@Maths

Code : `include/subsetix/`

Demo : `examples/mach2_cylinder/`

Annexes

Historique des implémentations

Version	Description	Performance	Statut
v1	CPU only, Sparse CSR + Workspaces Première implémentation séquentielle	Faster than baseline	✓ Stable
v3	CUDA only Algèbre d'ensembles GPU Proof of concept	**Plus rapide**	✓ PoC validé

Leçons apprises

- Le **tiling** améliore la localité mais complexifie énormément
- CUDA natif plus rapide mais moins portable
- Kokkos = meilleur compromis **fiabilité/portabilité**

Choix final : Kokkos

- Code **unique** pour CPU et GPU
- Maintenance simplifiée
- Facilité de test et vérification
- Écosystème actif (Sandia, Trilinos)

Comparaison avec CUDA natif

Aspect	CUDA	Kokkos

Backends supportés

- **OpenMP** : CPU multi-thread
- **CUDA** : NVIDIA GPU
- **HIP** : AMD GPU
- **SYCL** : Intel GPU
- **Serial** : debug et tests

Avantages pour ce projet

1. Développement plus rapide
Debug sur CPU (Serial/OpenMP), deploy sur GPU

2. Tests fiables
Même code testé sur CPU et GPU
Pas de bugs “GPU-only” cachés

3. Std Algorithms
`transform`, `reduce`, `scan`, `copy`...
API familière, optimisée par plateforme

4. Écosystème
Trilinos, ArborX, Cabana...
Support Sandia National Labs

Utilisation Intensive de LLMs

Modèles utilisés

- **Claude Opus 4** (Anthropic)
- **Claude Sonnet 4** (Anthropic)

Pattern de travail

```
1. PLAN
   Architecture et interfaces
   Discussion des alternatives

2. QUESTION
   Détails d'implémentation
   Edge cases

3. IMPLEMENTATION
   Génération du code
   Revue et itération
```

Avantages observés

- **Exploration rapide** des designs
- Documentation inline générée
- Tests suggérés automatiquement
- Refactoring assisté

Points d'attention

- Vérification systématique du code
- LLMs peuvent halluciner des APIs
- Toujours compiler et tester
- Garder le **contrôle architectural**

LLM = **accélérateur**, pas remplacement
L'expertise humaine reste essentielle

Kokkos

- Site : kokkos.org
- GitHub : github.com/kokkos/kokkos
- Wiki : kokkos.org/kokkos-core-wiki

CUDA

- CUDA Toolkit Documentation
- CUDA C++ Programming Guide

Visualisation

- VTK : vtk.org
- ParaView : paraview.org

Morphologie Mathématique

- Serra, J. “Image Analysis and Mathematical Morphology” (1982)
- Soille, P. “Morphological Image Analysis” (2003)

Code source

```
include/subsetix/
├── geometry/      # IntervalSet2D
├── field/         # Field2D
├── csr_ops/       # Algorithmes
├── multilevel/    # AMR
└── detail/        # Utilitaires

examples/mach2_cylinder/
└── mach2_cylinder.cpp  # Demo AMR
```