Optimisation du Game of Life sur GPU

De l'implémentation naïve au Bit-Packing

A5 TP Conway

NVIDIA RTX 3080 — Grille 32k $\times$ 32k

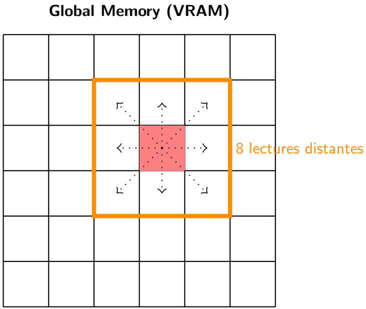
48 FPS $\rightarrow$ 416 FPS

Version 1: noif-char (Naïf)

Taille Grille: 32768×32768 octets
≈ 1 GB de RAM

FPS: 48.69
Type des données: `char` (1 octet = 1 cellule)
Pas de Shared Memory

Taille bloc: 32×32 threads (calculent 32×32 cellules)



Logique “Branchless” (noif)

```
// On compte les voisins vivants
int neighbors = 0;
for (int dy = -1; dy < 2; dy++) {
    for (int dx = -1; dx < 2; dx++) {
        if (x + dx >= 0 && x + dx < width && y + dy >= 0 && y + dy < height && (dy != 0 || dx !=
            0)) {
        neighbors += grid[idx + dx + dy * width];
        }
    }
}

// Calcul conditionnel sans 'if'
int alive = (neighbors == 3) | (grid[y * width + x] == 1 && neighbors == 2);
new_grid[idx] = alive;
```

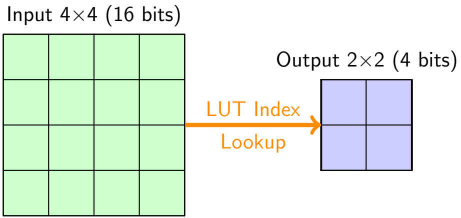
Lecture 9 cells → Nombre de voisins → Calcul état cellule → Ecriture

Version 2: Lookup Table 4×4

Performance:

- **FPS:** 112.76 (×2.3)
- **Stratégie:** Pré-calcul (LUT)
- **Mémoire:** Utilisation de la **Shared Memory**

Concept: Réduire les calculs (comptage voisins) par une simple lecture tableau.



`unsigned char host_lut[65536]`

Construction de l'index LUT

```c
// Construction de l'index 16-bits
uint16_t state_idx = 0;
#pragma unroll
for (int r = 0; r < 4; r++) {
  #pragma unroll
  for (int c = 0; c < 4; c++) {
    if (tile[tile_r + r][tile_c + c]) {
       state_idx |= (1 << (r * 4 + c));
    }
  }
}
unsigned char res = lut[state_idx];
```

b30	b31	b32	b33
b20	b21	b22	b23
b10	b11	b12	b13
b00	b01	b02	b03

$\longrightarrow$ `uint16_t state_idx`

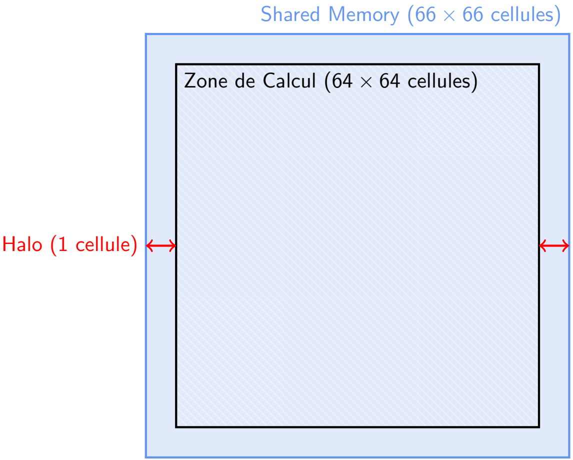
Block 4×4 (voisinage)

Gestion Mémoire: Stratégie par Bloc

Un Thread calcule **2×2 cellules**

Bloc 32×32 threads
Surface calculée: **64 × 64 cellules**

Shared Memory: **66 × 66 cellules**



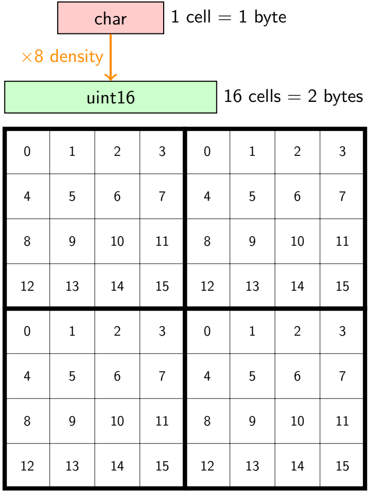
Version 3: uint16 (Bit Packing)

Performance:

- **FPS: 416.51** ($\times$8.5 vs Naïf)
- **Stratégie:** Compression Mémoire

Idée Clé:

- 1 cellule $=$ 1 bit (pas 1 octet)
- Grille 32768×32768 cells
- **Empreinte:** 128 MB
- $\frac{32768 \times 32768}{8 \text{ bits/octet}} = 128$ MB



Architecture du Kernel uint16

1 uint16 = 1 thread

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Architecture du Kernel uint16

1 uint16 $=$ 1 thread

Utilisation de la LUT 4×4

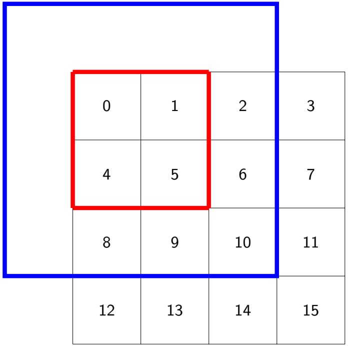
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Architecture du Kernel uint16

Utilisation de la LUT 4×4

Construction de l'indice 16-bits
à partir des uint16 voisins

1 uint16 = 1 thread



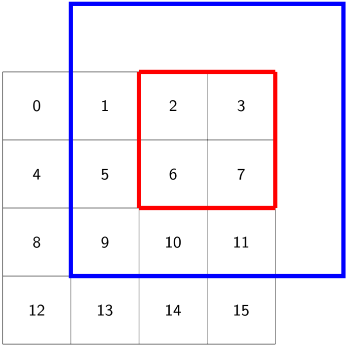
Architecture du Kernel uint16

Utilisation de la LUT 4×4

Construction de l'indice 16-bits
à partir des uint16 voisins

Patch suivant

1 uint16 $=$ 1 thread



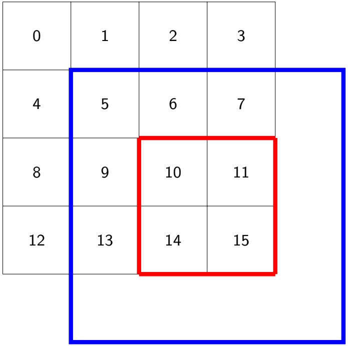
Architecture du Kernel uint16

1 uint16 = 1 thread

Utilisation de la LUT 4×4

Construction de l'indice 16-bits
à partir des uint16 voisins

Patch suivant



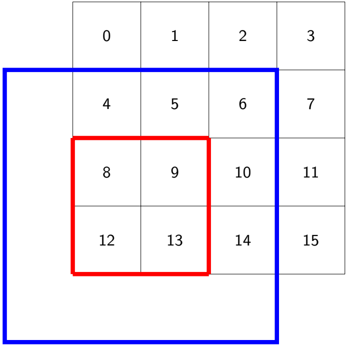
Architecture du Kernel uint16

1 uint16 = 1 thread

Utilisation de la LUT 4×4

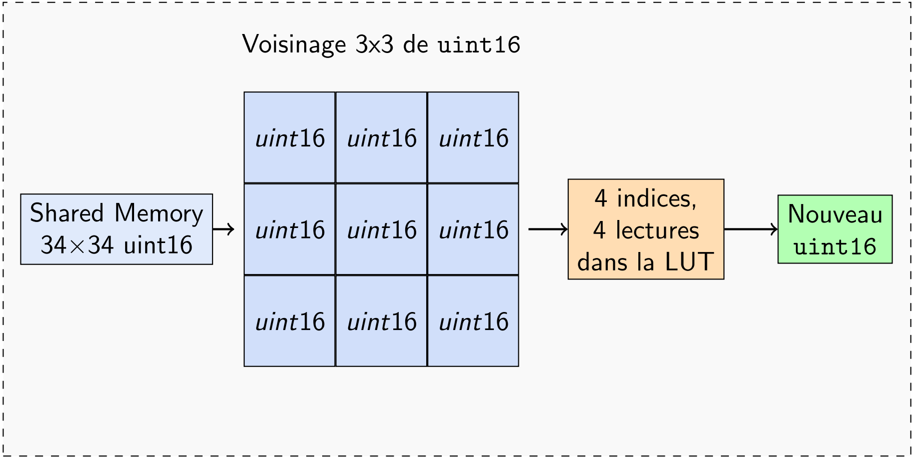
Construction de l'indice 16-bits
à partir des uint16 voisins

Patch suivant



Architecture du Kernel uint16

1 Thread = `uint16`
Bloc 32×32 threads = 128×128 cellules



Résumé des Gains de Performance

Version du Kernel	**Performance (FPS)**
`noif-char` (Naïf)	48.81
`lookup4x4` (Shared Memory)	152.04
`uint16` (Bit-packing)	**416.51**