Challenges

Pointage

Le pointage de chaque challenge et bonus est évalué avec le tableau suivant.

Fonctionne pour toutes les entrées	Ressources utilisées	Nombre de cycles	Total
bool	%	%	
80	10	10	100

Ce pourcentage est ensuite multiplié par la valeur de la question pour obtenir un nombre de point. Le pointage final est la somme de tous ces points.

Les challenges vont être testés avec différentes entrées que celles fournies en exemple dans ce document.

Les objectifs peuvent être de différents types:

- bool: Soit vous avez tous les points ou aucun
- %: Le meilleur compétiteur pour ce challenge fait tous les points de cette catégorie, le dernier, aucun. Le pointage de tous les autres participants est donné par une interpolation linéaire entre ces deux extrêmes. Vous devez d'abord réussir tous les objectifs booléen du challenge avant de pouvoir en faire dans cette catégorie.

Définitions:

- Nombre de cycles: Nombre de cycles pour avoir une sortie pour toutes les entrées testées.
- Ressources utilisées: `(nombre de banques de mémoire) x (taille des banques de mémoire) + (nombre de coeurs)`

Sauf si indication contraire, vos programmes doivent rouler dans une boucle. C'est-à-dore qu'ils doivent attendre pour une prochaine entrée après en avoir traité une.

1. Passage de valeurs (10 pts)

Restrictions

```
- Deux entrées (a, b)
- Deux sorties (A, B)
```

Description

Déplacez les valeurs des deux entrées vers les deux sortie (`A=a, B=b`).

Exemple d'entrées

Entrées:

```
235 225
87  128
162 156
242 220
81  46
```

```
15  233
192 126
62  79
100 2
140 65
```

Sorties:

```
235 225
87  128
162 156
242 220
81  46
15  233
192 126
62  79
100 2
140 65
```

Bonus (5 pts)

Déplacer les valeurs des deux entrées vers les deux sorties, mais en les inversant de sorte que `A=b, B=a`.

2. Somme simple (10 pts)

Restrictions

```
- Deux entrées (a, b)
- Une sortie (A)
```

Description

Additionner les deux entrées et retourner le résultat sur une seule sortie (`A=a+b`). La sortie devrait dépasser si elle excède le maximum.

Exemple d'entrées

Entrées:

```
0   0
0   1
153 108
250 224
16  176
109 18
203 78
143 19
141 71
228 145
```

Sorties:

```
0
1
5
218
192
127
25
162
212
117
```

Bonus (10 pts)

Faites la même chose, mais avec des nombre 16 bits en little endian. Vous devrez ainsi avoir quatre entrées (a, b, c, d) and deux sorties tel que:

```
resultat = b << 8 | a + d << 8 | c
A = resultat & 0xff
B = resultat >> 8
```

3. Négation conditionnelle (20 pts)

Restrictions

```
- Trois entrées (a, b, c)
- Une sortie (A)
```

Description

La sortie `A` doit être `c` quand `a < b`, `0` quand `a = b` and `-c` quand `a > b`.

Exemple d'entrées

Entrées:

```
232 4 35
222 75 35
48 48 35
145 123 35
71 248 35
28 64 35
86 251 35
43 0 35
252 196 35
64 192 0
```

Sorties:

```
221
221
0
221
```

```
35
35
35
221
221
0
```

Note: Le simulateur ne sait pas s'il a affaire à des nombre signés ou non-signé et en conséquant, les affichent toujours comme des non-signés. Par exemple, 221 est `0b11011101` qui est aussi -35 quand interprété comme un nombre signé de 8 bits.

Bonus (5 pts)

Retourner plutôt `max(a, b)` quand `a = b`.

4. Logarithme en base 2 parallèle (20 pts)

Restrictions

```
- Trois entrées (a, b, c)
- Trois sorties (A, B, C)
```

Description

Calcule le logarithme en base 2 de tous ses entrées de sorte que:

```
A = floor(log2(a))
B = floor(log2(b))
C = floor(log2(c))
```

Exemple d'entrées

Entrées:

```
113 167 204
21  57  26
201 121 204
```

Sorties:

```
6 7 7
4 5 4
7 6 7
```

Bonus (5 pts)

Calculer le logarithme en base 16, plutôt que le logarithme en base 2.

5. Multiplication (30 pts)

Restrictions

```
- Deux entrées (a, b)
- Une sortie (A)
```

Description

Calcule `A = a * b`. Les dépassement doivent «wrap around».

Exemple d'entrées

Entrées:

```
0 3
0 0
6 8
3 32
79 39
66 56
```

Sorties:

```
0
0
48
96
21
126
```

6. Flou gaussien

Restrictions

```
- 900 entrées (a)
- 900 sorties (A)
```

Description

En partant d'une image d'entrée de 30x30, produisez une image floué de 30x30 en utilisant le noyau suivant:

```
1/16 1/8 1/16
1/8  1/4 1/8
1/16 1/8 1/16
```

Tel que

$$\begin{aligned} A_{i,j} = 1/16a_{i-1,j-1} + 1/8a_{i-1,j} + 1/16a_{i-1,j+1} + \\ 1/8a_{i,j-1} + 1/4a_{i,j} + 1/8a_{i,j+1} + \\ 1/16a_{i+1,j-1} + 1/8a_{i+1,j} + 1/16a_{i+1,j+1} + \end{aligned}$$

Ce challenge est spécial pour deux raisons:

1. C'est le premier challenge pour lequel vous devriez absolument utiliser un CPU 3D. Je suggère d'utiliser un CPU de Zx30x30 avec tous les coeurs de la première couche en Z utilisés comme entrées et tous les coeurs de la dernière couche en Z utilisés comme sorties. Z est à votre guise. Vous pouvez utiliser plusieurs couche intermédiaire en Z pour exécuter vos calculs.
2. La configuration du CPU est difficile Malheureusement, à ce jour, le simulator ne supporte pas les plages de paramètres. En effet, pour utiliser 900 entrées, vous devrez toutes les lister. La même limitation s'applique aux sorties et à `core_to_mem`. S'il vous plait, n'essayer pas de tout lister à la main et utiliser plutôt un outil comme Python pour générer ces listes pour vous. Voici quelques exemples pour baser votre travail:

```
# Affiche une liste de 900 éléments dans le format accepté par le simulateur LAVAL.
print(", ".join([str(i) for i in range(900)]))

# Affiche tous les identifiants de coeurs membres de la deuxième colomne de la première couche.
print(", ".join([str(i) for i in range(1, 30*30, 30)]))
```

Exemple d'entrées

Trop gros pour être inclus ici.