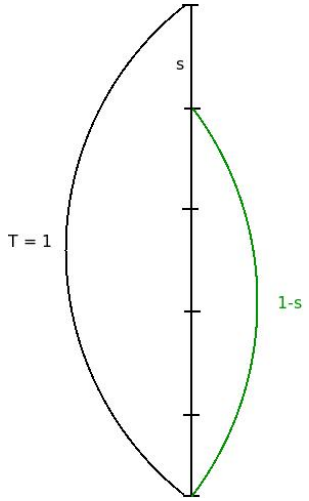
TD n°5

*Digression palpitante et indispensable sur la loi d'Amdahl (**« Wikipédia, c'est des branques, ne pas reprendre leurs notations ! »**, traduction libre du prof d'APP, le 03/12/2014) :*



$$T_{acc} = T \times S + \frac{(1-S) \times T}{p}$$
$$speedup = \frac{T}{T_{acc}} = \frac{1}{S + \frac{1-S}{p}}$$
$$speedup_{max} = \frac{1}{S}$$

Soit :

- T le temps d'exécution d'un programme sur le système d'origine.
- T_{acc} le temps d'exécution du même programme sur le système amélioré.
- S est la fraction du temps T non concernée par l'amélioration.
- Speedup est l'accélération obtenue par l'amélioration.
- P est le nombre de cœurs/processeurs.

Exercice 1

S = 6%

10 = 1/((6/100) + (0,94/P)) → P = 24

Exercice 2

Speedup = 9

P = 10

9 = 1/(S + (1 - S)/10) → S = 1,24%

Exercice 3

Erreur à la ligne « la ligne 0 de ce tableau est facile à calculer : S[0][j] = 1 » : remplacer 1 par U[0].
Erreur à la ligne « En considérant N = 512 et C = 16 » : remplacer par N = 256 et C + 1 = 32.

Question 2 :

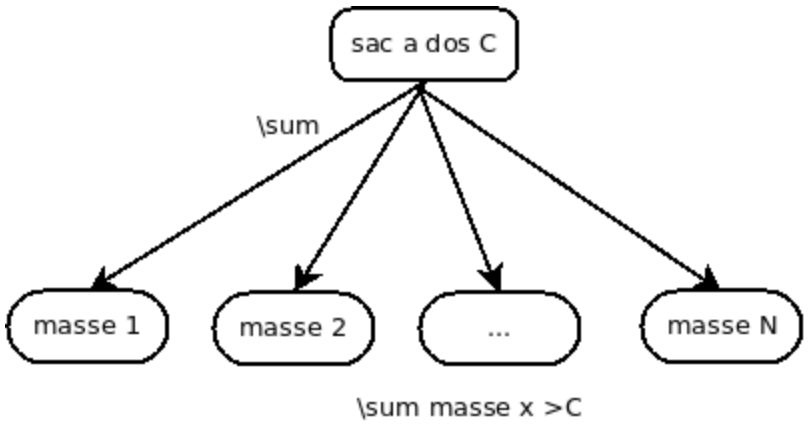
N = 256 C + 1 = 32

Problème du sac à dos… (Décathlon d'après le Monsieur)

Sac à dos C

N objets M_i $\sum M_i > C$

U_i



Maximiser U

	0	1	2	3	...	6	...	...	C
0	U0	U0	U0	0	0	0	...	...	0
1									
2									
...									
i									

…									
N - 1									

Question 1 :

```
// chaque ligne va calculer tous les éléments de la ligne
// 4 ESPACES LES INDENTATIONS :)
# pragma omp parallel private (i)
{
        int nt = omp.get_num_threads();
        # pragma omp for private(j) shared (S, M, U) schedule(static, (C+1)/nt)
        for (j = 0; j <= C; j++) {
            if (M[0] <= j) {
                S[0][j] = U[0];
            } else {
                S[0][j] = 0;
            }
        }
        for (i = 0; i < N; i++) {
        # pragma omp for private(j) shared (S, M, U) schedule (static, (c+1)/nt)
            for(j = 0; j <= C; j++) {
                if ((j < M[i]) || (S[i-1][j] > (S[i-1][j - M[i]] + U[i])) {
                    S[i][j] = S[i-1][j];
                }
                else {
                    S[i][j] = S[i - 1][j - M[i]] + U[i];
                }
            }
        int cap = C;
        for (i = N-1; i > 0; i++) {
            if (s[i][cap] == s[i-1][cap]) {
                E[i] = 0;
            } else {
                E[i] = 1;
                cap = cap - M[i];
            }
        }
        if (cap >= M[0]) {
            E[0] = 1;
        }
        else {
            E[0] = 0;
        }
}
```

Question n°2 :

N = 256 C+1 = 32 nt = 2,4,8,16,32

partie séquentielle :

init M et U = 2048

calcul de E = N*2 = 512

partie parallèle :

((C + 1) / nt) * 1

N * ((C + 1) / nt) * 4

Volume de données total sur le bus :

1 thread : 2048 + 512 + 32 + 256*32*4 $\backsimeq 2^{15}$

2^{11} 2^{9} 2^{5} 2^{15}

2 thread : 2048 + 512 + 16 + 256*16*4 $\backsimeq 2^{14}$ Acc = 2

2^{11} 2^{9} 2^{4} 2^{14}

4 : 2^{13} Acc = 4

8 : 2^{12} Acc = 8

16 : $2^{11}+2^{11}$ Acc = 8

32 : $2^{11}+2^{10}$ Acc = 16

Acc max = $(2^{11}+2^{9}+2^{5}+2^{15})/(2^{11}+2^{9})$

Acc : Tseq/(Tseq + ((1 - Tseq)/nt))

bloc = 64 octets = 16 éléments

- 1° ligne de S
  - ➔ bloc 0 de M present dans tout les cadres dans l'état S ⟶ nt BusRd
  - ➔ idem pour U[0] ⟶ nt BusRd
  - ➔ ecriture de S[0][ ] : (C+1) / (16 * nt) BusRdX (bloc dans l'état M)
- interlignes
  - ➔ si c'est multiple de 16 alors 2* nt BusRd (M et U)
  - ➔ (C+1) / (16 * nt) BusRdX + x BusRd (S[i-1][j-M[i]])
- E
  - ➔ N/64 échecs ⟶ N/64 BusRdx
  - ➔ 1 BusRd par bloc de S lu accès à M[i] => 0 transaction ⟶ 1 BusRd par bloc de S lu et calculé par un autre thread (on a déjà M dans l'état S) 2*N BusRdw"