JAOUANI Mohamed-Amine
DAKHLIA Cyrille

Lundi 16 Mai 2016

Programmation système avancée

Rapport de projet

I/ Description sommaire et parties réalisées

Tout en suivant scrupuleusement l'interface définie par le fichier *channel.*h, nous avons implémenté les parties suivantes :

- Les canaux de base mentionnés dans le sujet minimal, asynchrones (partie 3)
- Les canaux utilisant les tubes pour comparaison (partie 4.1)
- Les canaux globaux (partie 4.3)
- Les canaux synchrones (partie 4.4)
- La communication par lots (partie 4.5)

1) Canaux de base du sujet minimal, asynchrones (partie 3)

Pour cette partie nous avons utilisé un tableau de void* afin de contenir les données dans le canal, gardant en mémoire le nombre d'éléments dans celui-ci, ainsi que des indices permettant de connaître la position du prochain élément à lire et la position où écrire le prochain élément dans le tableau. Nous avons également utilisé des variables de conditions pour réguler la lecture et l'écriture dans ce type de canal lorsque le canal est vide ou plein.

2) Canaux utilisant les tubes pour comparaison (partie 4.1)

Nous avons simplement utilisé des tubes nommés ainsi que les mêmes variables utilisées pour les canaux de base à l'exception de *indiceRecv* et *indiceSend*.

3) Canaux globaux (partie 4.3)

Pour les canaux globaux nous avons réservé une zone en mémoire partagée, à l'aide de shm_open, que nous avons ajusté de manière précise à la taille de notre structure channel, à l'aide de la fonction ftruncate. Puis nous avons projeté ce bloc de mémoire partagée dans notre espace d'adressage en utilisant mmap, nous permettant ainsi d'effectuer nos send et nos recv dans notre channel.

4) Canaux synchrones (partie 4.4)

Quant aux canaux synchrones, nous utilisons deux variables de conditions permettant d'assurer les présences du lecteur et de l'écrivain simultanément, ainsi que deux sémaphores permettant d'assurer qu'il n'y ait qu'un lecteur et qu'un écrivain à la fois.

Nous utilisons par ailleurs un void* dans notre structure channel qui servira à contenir l'adresse de la data initialement vide du lecteur lorsque celui-ci fera un appel à la fonction recv. Ainsi, lorsque l'écrivain fera appel à la fonction send, il écrira sa donnée dans ce void*, ce qui la copiera directement dans la data du lecteur.

5) Communication par lots (partie 4.5)

Nous avons créé deux fonctions channel_send_lots et channel_recv_lots qui prennent en plus un argument entier qui indique le nombre d'appel aux fonctions channel_send et channel_recv qui vont être effectuées.

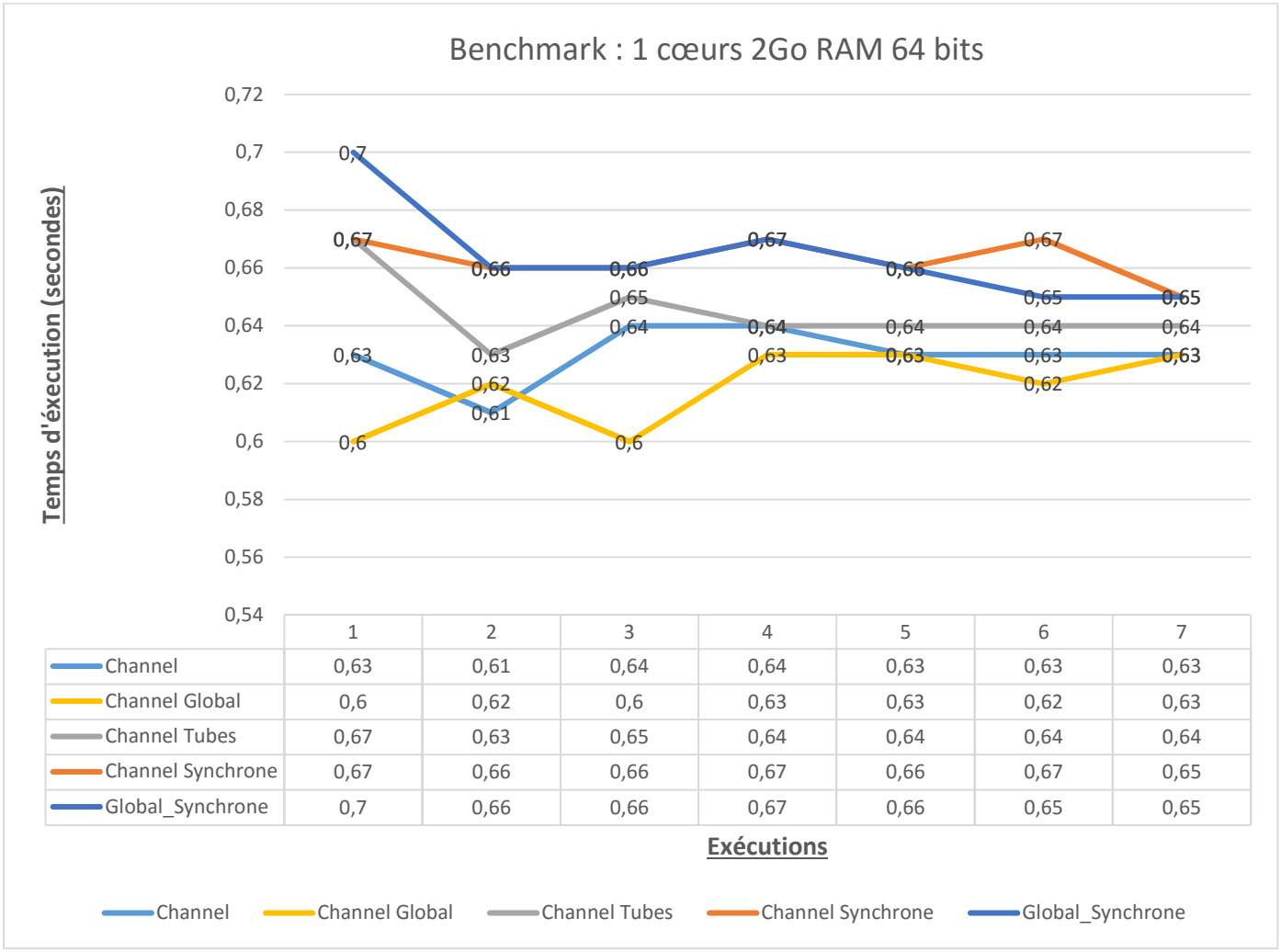
Les fonctions channel_send et channel_recv ayants déjà un mécanisme d'attente si le tampon est plein ou vide, il n'est pas nécessaire d'en implémenter un autre pour les fonctions channel_send_lots et channel_recv_lots.

6) Canaux à une seule copie (partie 4.6)

Dans notre implémentation des canaux synchrones, il n'y a qu'une seule copie de la donnée qui s'effectue. L'envoyeur attend un lecteur et écrit directement la donnée dans le data du lecteur. Ainsi à la fin de cette écriture, le lecteur voit sa data modifiée et n'a pas besoin de copier quoique ce soit en provenance du *channel*.

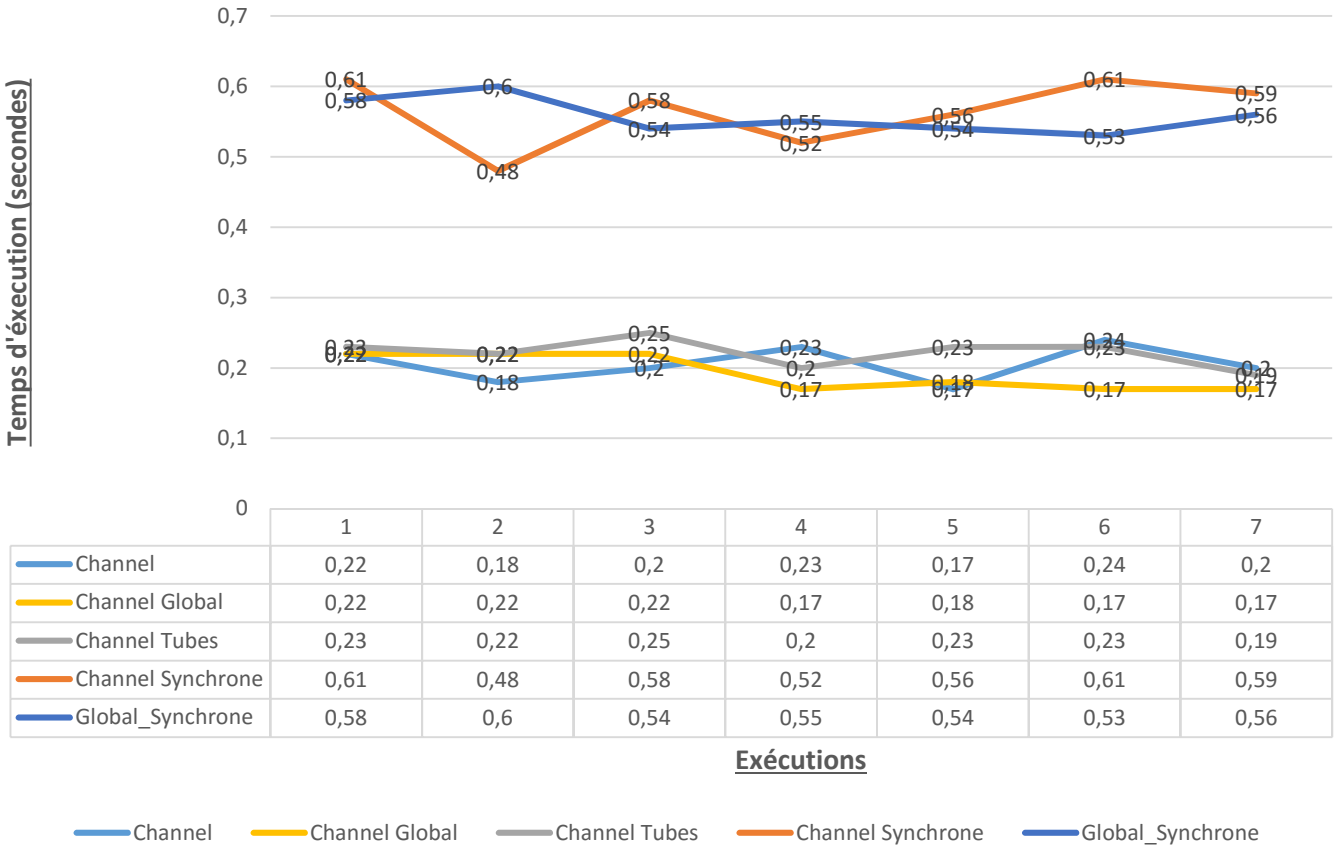
II/ Benchmarks

Voici quelques graphiques représentatifs des benchmarks obtenus pour nos différentes implémentations :



	1	2	3	4	5	6	7
Channel	0,63	0,61	0,64	0,64	0,63	0,63	0,63
Channel Global	0,6	0,62	0,6	0,63	0,63	0,62	0,63
Channel Tubes	0,67	0,63	0,65	0,64	0,64	0,64	0,64
Channel Synchrone	0,67	0,66	0,66	0,67	0,66	0,67	0,65
Global_Synchrone	0,7	0,66	0,66	0,67	0,66	0,65	0,65

Benchmark : 4 cœurs 2Go RAM 64 bits

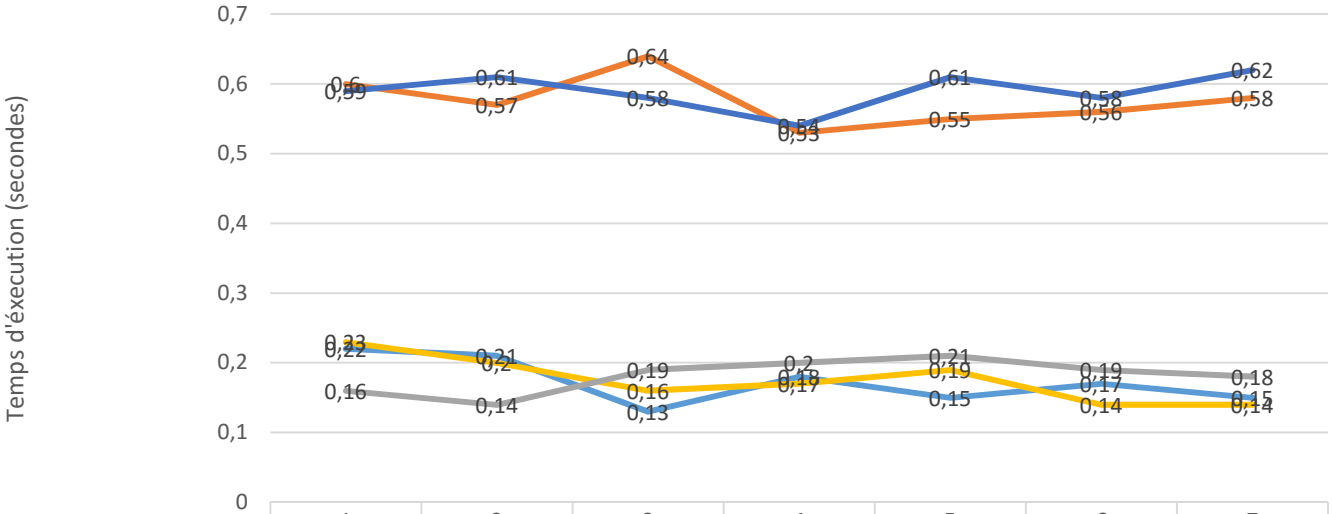


	1	2	3	4	5	6	7
Channel	0,22	0,18	0,2	0,23	0,17	0,24	0,2
Channel Global	0,22	0,22	0,22	0,17	0,18	0,17	0,17
Channel Tubes	0,23	0,22	0,25	0,2	0,23	0,23	0,19
Channel Synchrone	0,61	0,48	0,58	0,52	0,56	0,61	0,59
Global_Synchrone	0,58	0,6	0,54	0,55	0,54	0,53	0,56

Exécutions

Channel | Channel Global | Channel Tubes | Channel Synchrone | Global_Synchrone

Benchmark : 4 cœurs 4Go RAM 64 bits

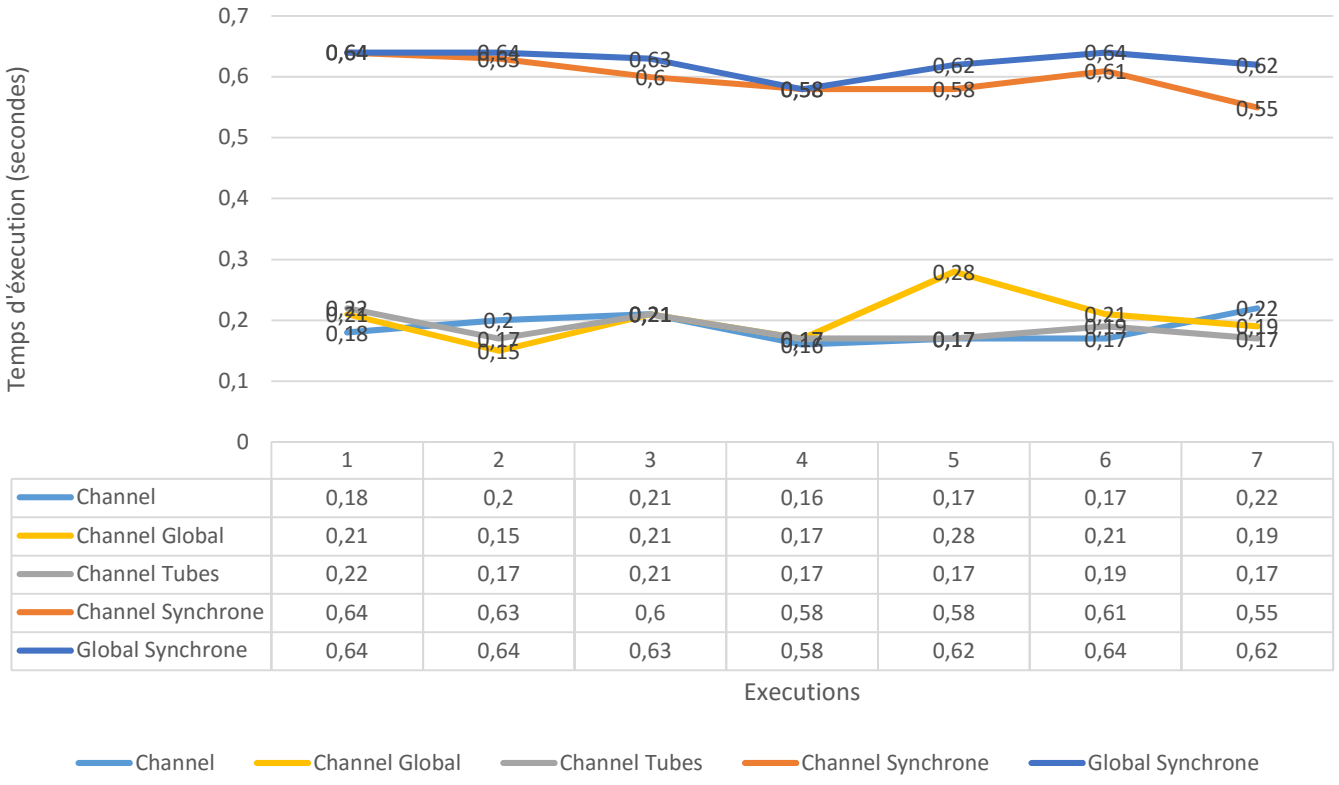


	1	2	3	4	5	6	7
Channel	0,22	0,21	0,13	0,18	0,15	0,17	0,15
Channel Global	0,23	0,2	0,16	0,17	0,19	0,14	0,14
Channel Tubes	0,16	0,14	0,19	0,2	0,21	0,19	0,18
Channel Synchrone	0,6	0,57	0,64	0,53	0,55	0,56	0,58
Global_Synchrone	0,59	0,61	0,58	0,54	0,61	0,58	0,62

Executions

Channel | Channel Global | Channel Tubes | Channel Synchrone | Global_Synchrone

Benchmark : 6 cœurs 6Go RAM 64 bits

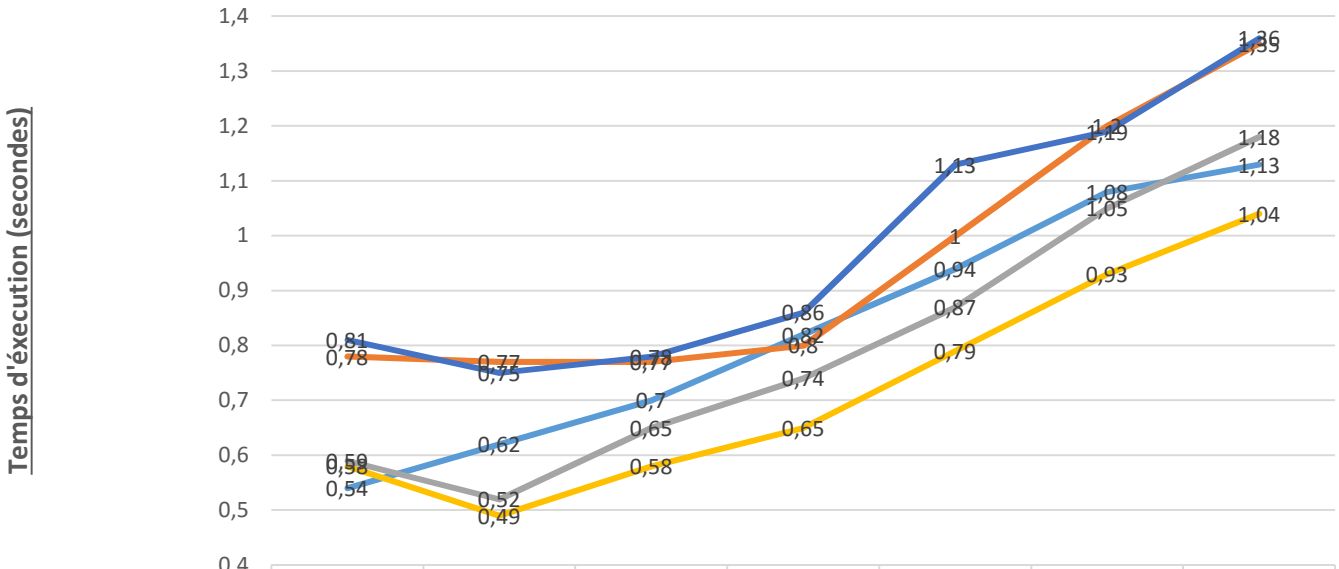


	1	2	3	4	5	6	7
Channel	0,18	0,2	0,21	0,16	0,17	0,17	0,22
Channel Global	0,21	0,15	0,21	0,17	0,28	0,21	0,19
Channel Tubes	0,22	0,17	0,21	0,17	0,17	0,19	0,17
Channel Synchrone	0,64	0,63	0,6	0,58	0,58	0,61	0,55
Global Synchrone	0,64	0,64	0,63	0,58	0,62	0,64	0,62

Executions

Channel Channel Global Channel Tubes Channel Synchrone Global Synchrone

Benchmark : 2 cœurs Intel Celeron CPU 4Go RAM 1,60 GH



	1	2	3	4	5	6	7
Channel	0,54	0,62	0,7	0,82	0,94	1,08	1,13
Channel Global	0,58	0,49	0,58	0,65	0,79	0,93	1,04
Channel Tubes	0,59	0,52	0,65	0,74	0,87	1,05	1,18
Channel Synchrone	0,78	0,77	0,77	0,8	1	1,2	1,35
Global_Synchrone	0,81	0,75	0,78	0,86	1,13	1,19	1,36

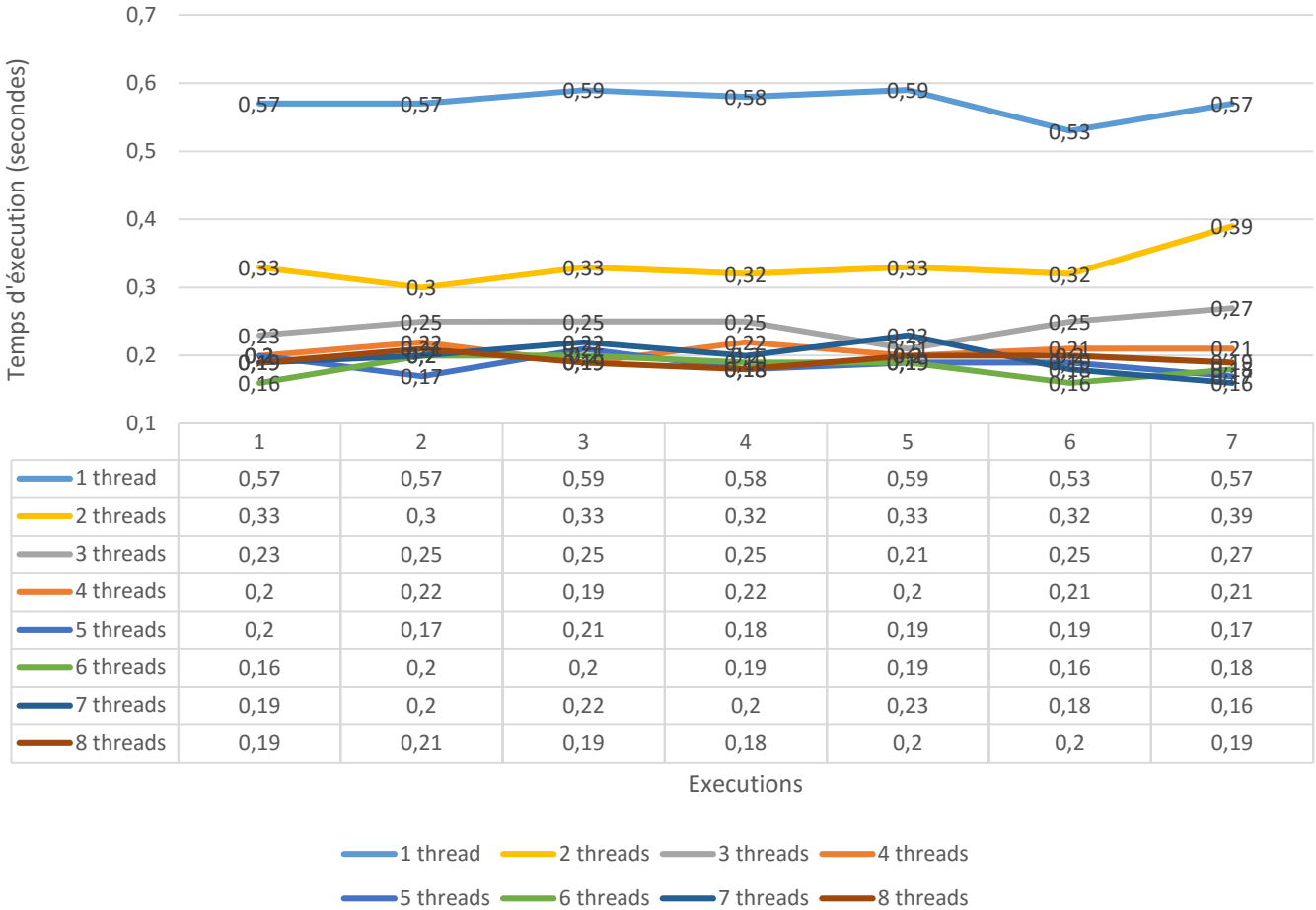
Zoom

Channel Channel Global Channel Tubes Channel Synchrone Global_Synchrone

Benchmark : CHANNEL 6Go RAM 64 bits 8 CPUs

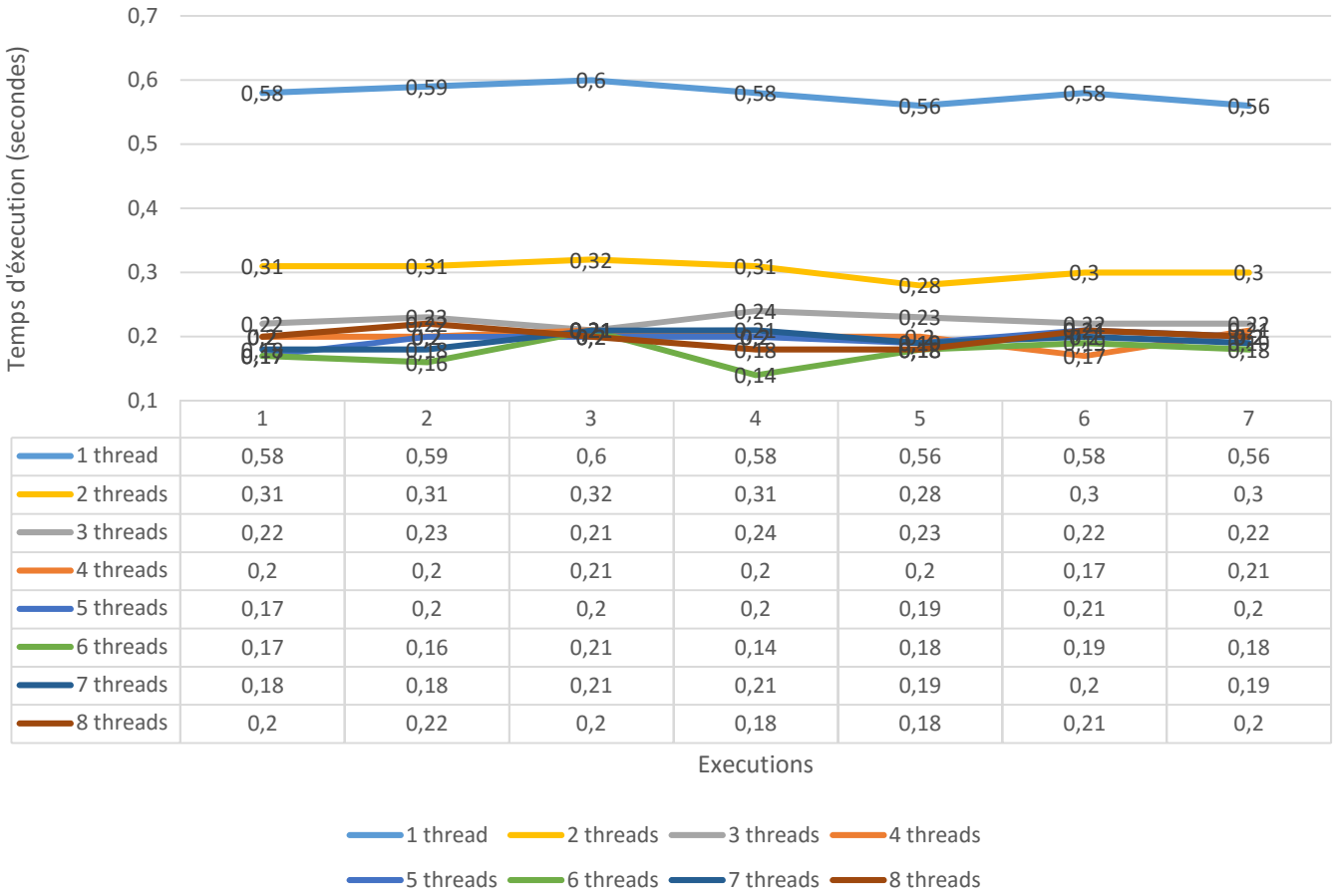
	1	2	3	4	5	6	7
1 thread	0,63	0,61	0,64	0,64	0,63	0,63	0,63
2 threads	0,32	0,29	0,28	0,37	0,31	0,32	0,27
3 threads	0,22	0,24	0,21	0,2	0,23	0,24	0,23
4 threads	0,22	0,21	0,13	0,18	0,15	0,17	0,15
5 threads	0,18	0,2	0,15	0,2	0,15	0,2	0,19
6 threads	0,18	0,2	0,21	0,16	0,17	0,17	0,22
7 threads	0,18	0,18	0,18	0,2	0,19	0,16	0,23
8 threads	0,2	0,22	0,15	0,17	0,18	0,15	0,19

Benchmark : TUBES 6Go RAM 64 bits 8 CPUs



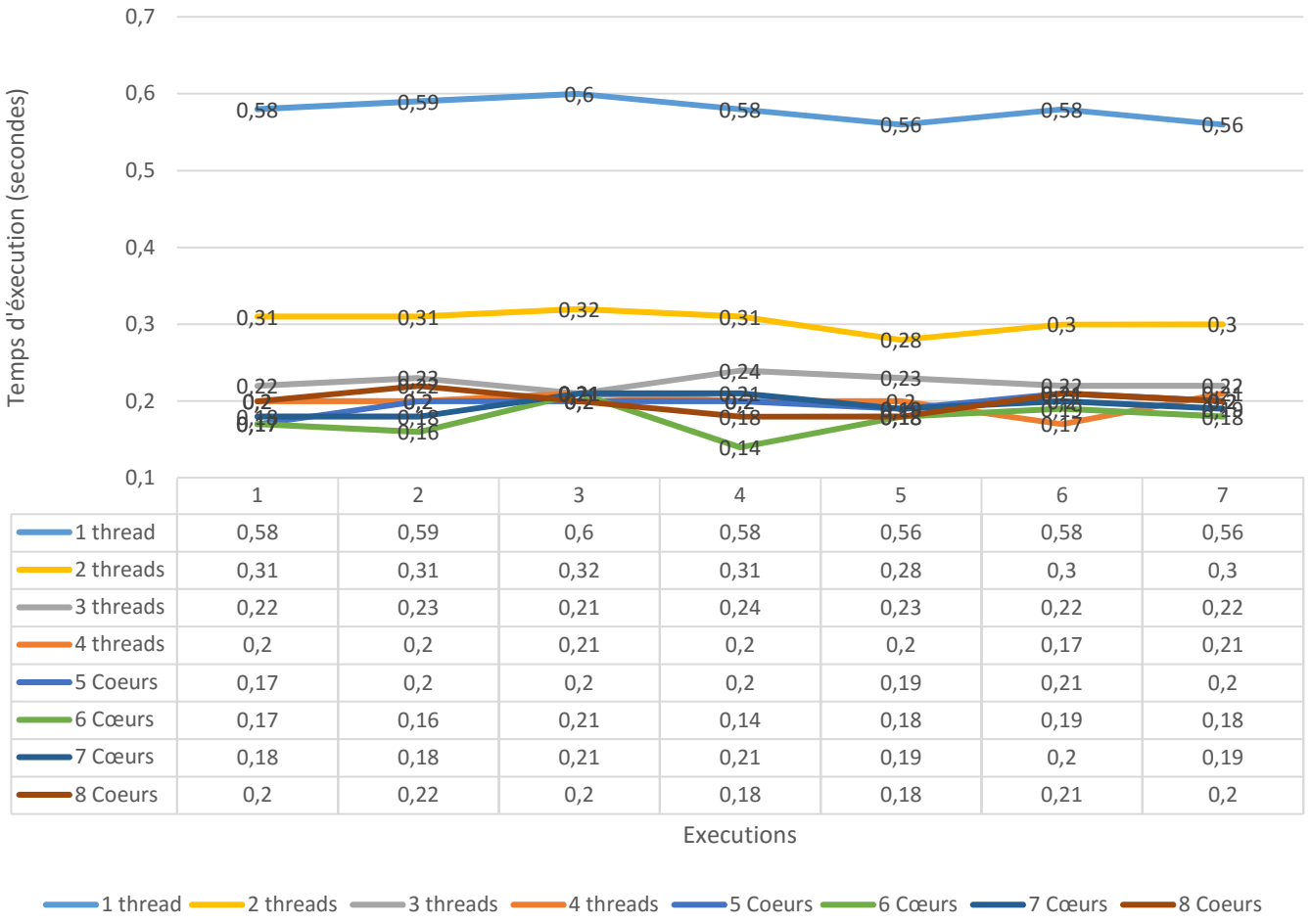
	1	2	3	4	5	6	7
1 thread	0,57	0,57	0,59	0,58	0,59	0,53	0,57
2 threads	0,33	0,3	0,33	0,32	0,33	0,32	0,39
3 threads	0,23	0,25	0,25	0,25	0,21	0,25	0,27
4 threads	0,2	0,22	0,19	0,22	0,2	0,21	0,21
5 threads	0,2	0,17	0,21	0,18	0,19	0,19	0,17
6 threads	0,16	0,2	0,2	0,19	0,19	0,16	0,18
7 threads	0,19	0,2	0,22	0,2	0,23	0,18	0,16
8 threads	0,19	0,21	0,19	0,18	0,2	0,2	0,19

Benchmark : GLOBAL 6Go RAM 64 bits 8 CPUs

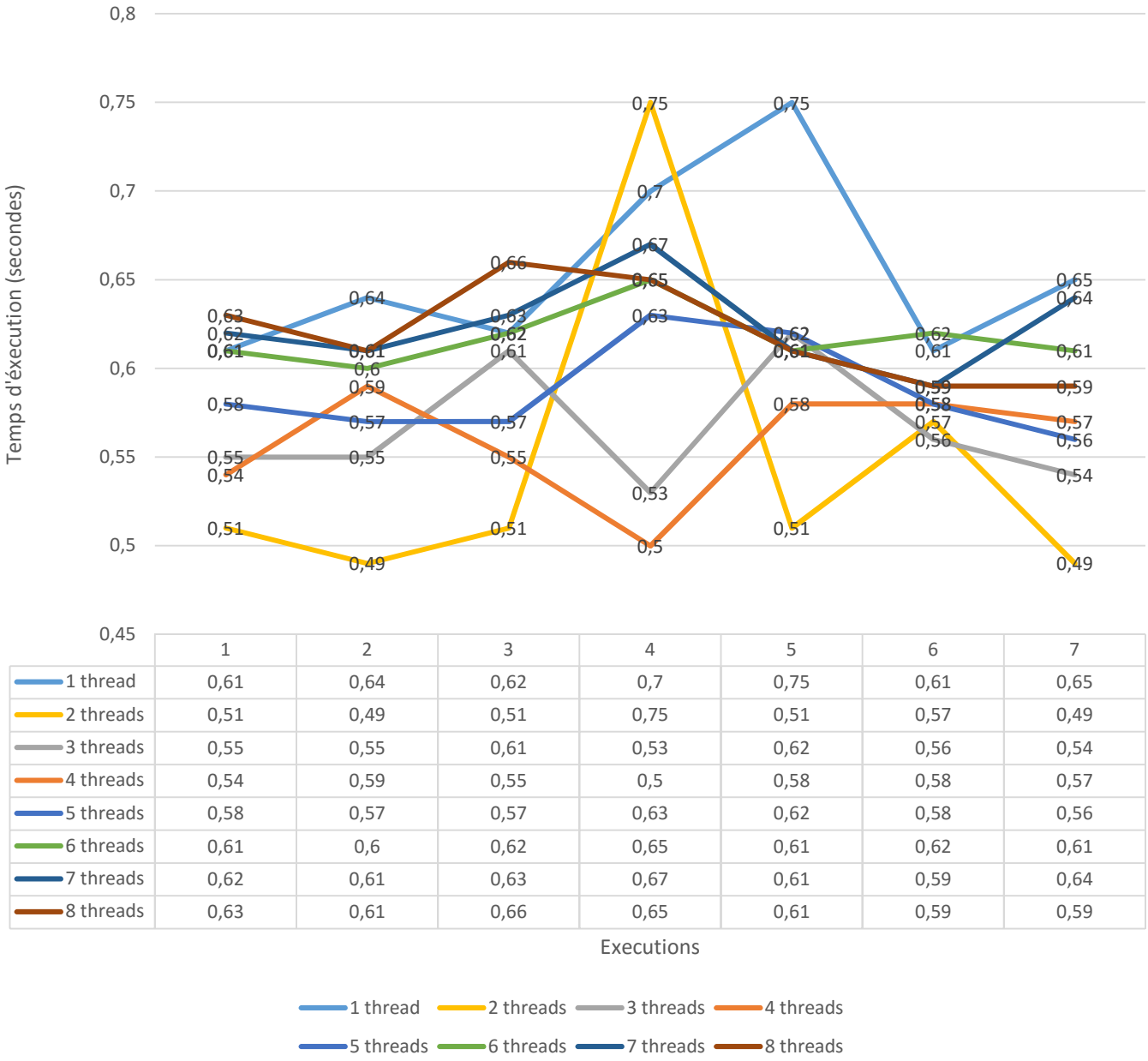


	1	2	3	4	5	6	7
1 thread	0,58	0,59	0,6	0,58	0,56	0,58	0,56
2 threads	0,31	0,31	0,32	0,31	0,28	0,3	0,3
3 threads	0,22	0,23	0,21	0,24	0,23	0,22	0,22
4 threads	0,2	0,2	0,21	0,2	0,2	0,17	0,21
5 threads	0,17	0,2	0,2	0,2	0,19	0,21	0,2
6 threads	0,17	0,16	0,21	0,14	0,18	0,19	0,18
7 threads	0,18	0,18	0,21	0,21	0,19	0,2	0,19
8 threads	0,2	0,22	0,2	0,18	0,18	0,21	0,2

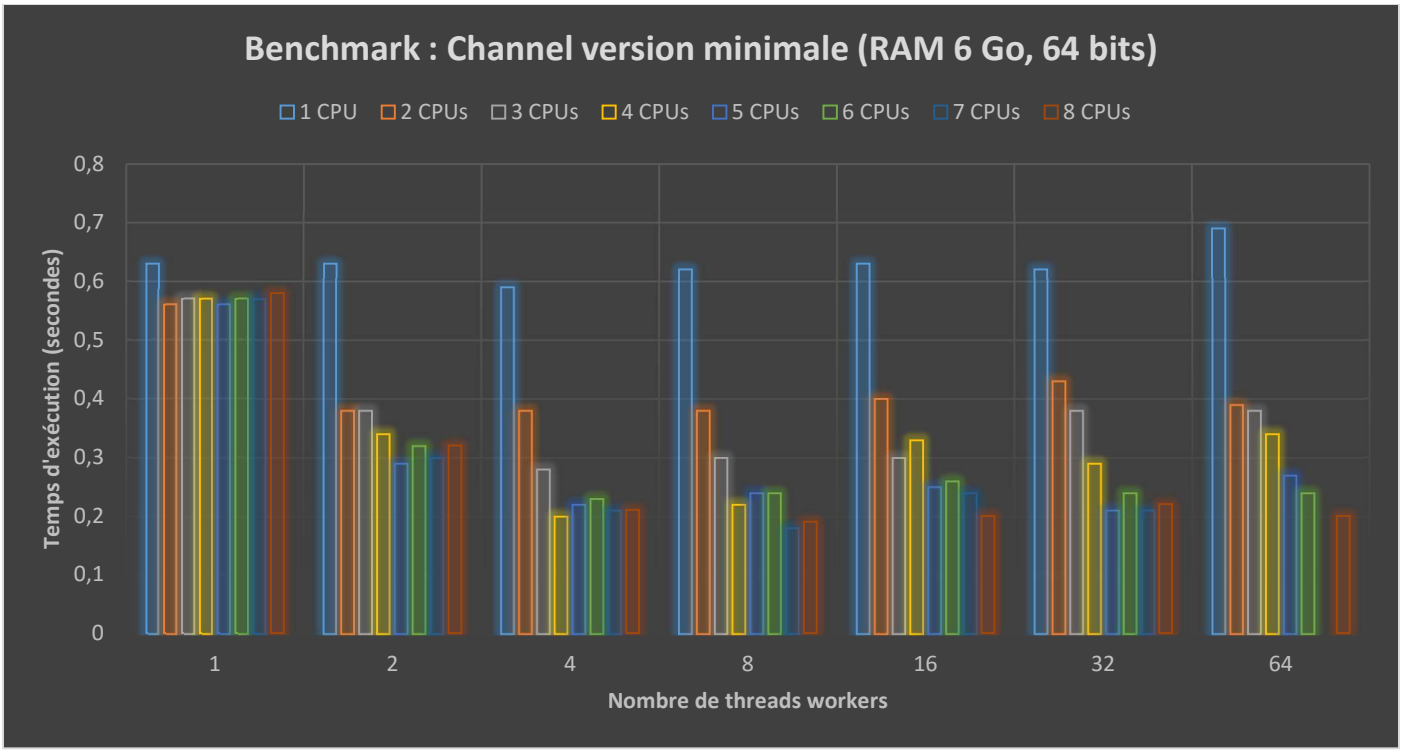
Benchmark : SYNCRHONE 6Go RAM 64 bits 8 CPUs

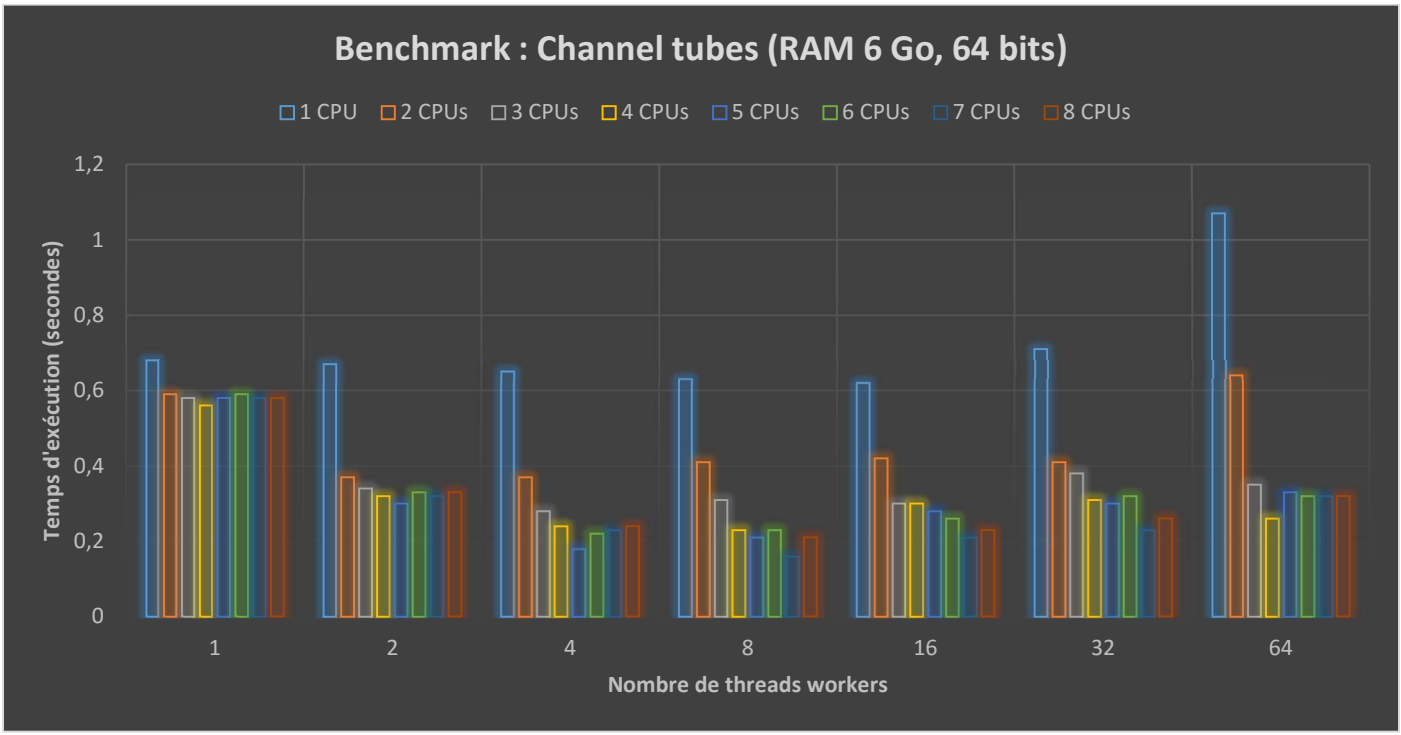


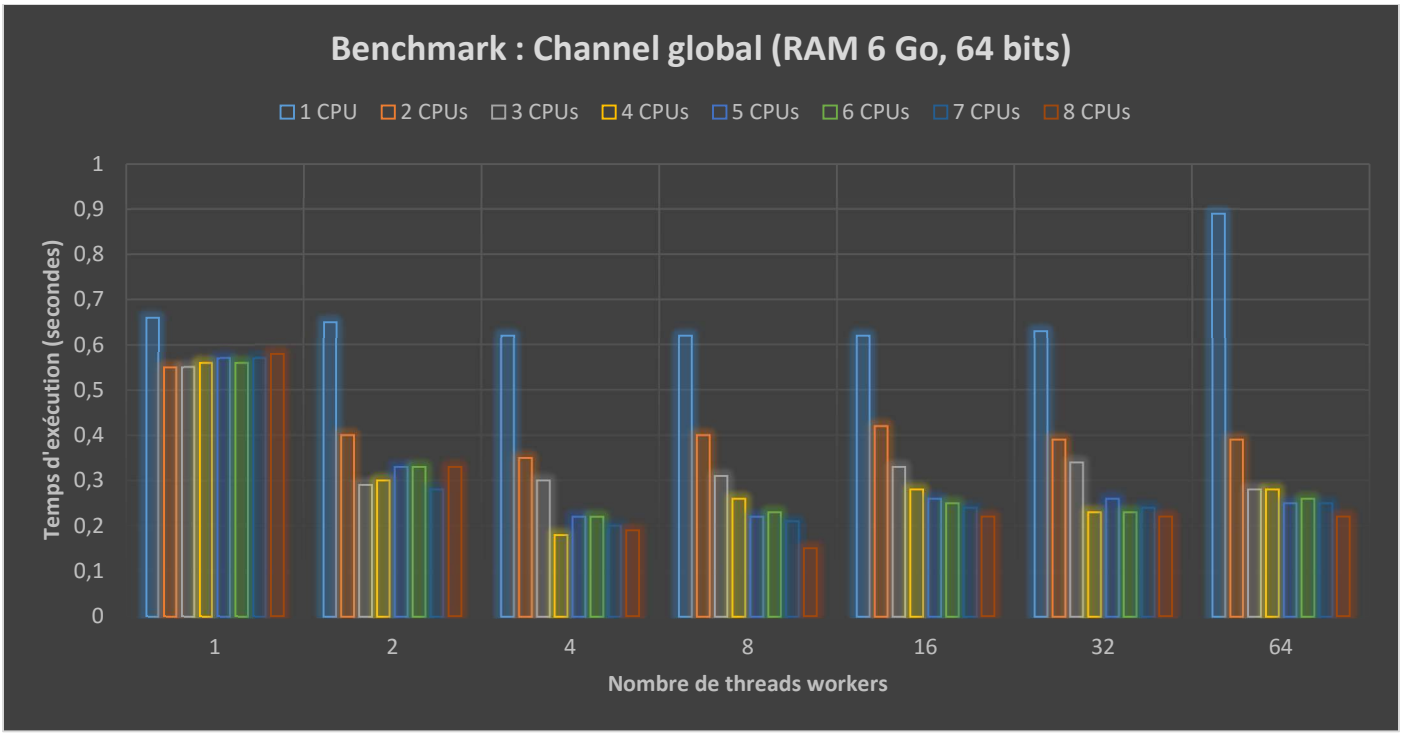
	1	2	3	4	5	6	7
1 thread	0,58	0,59	0,6	0,58	0,56	0,58	0,56
2 threads	0,31	0,31	0,32	0,31	0,28	0,3	0,3
3 threads	0,22	0,23	0,21	0,24	0,23	0,22	0,22
4 threads	0,2	0,2	0,21	0,2	0,2	0,17	0,21
5 Coeurs	0,17	0,2	0,2	0,2	0,19	0,21	0,2
6 Cœurs	0,17	0,16	0,21	0,14	0,18	0,19	0,18
7 Cœurs	0,18	0,18	0,21	0,21	0,19	0,2	0,19
8 Coeurs	0,2	0,22	0,2	0,18	0,18	0,21	0,2

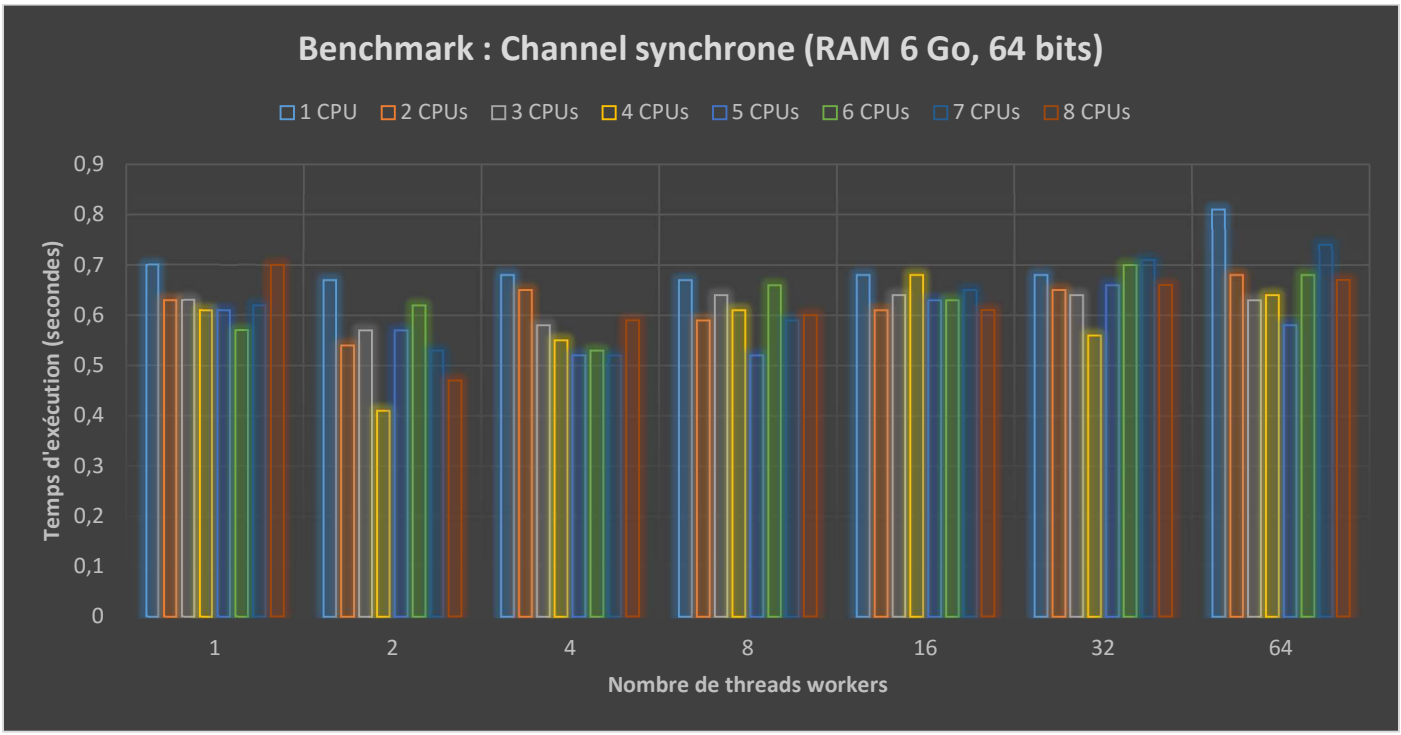


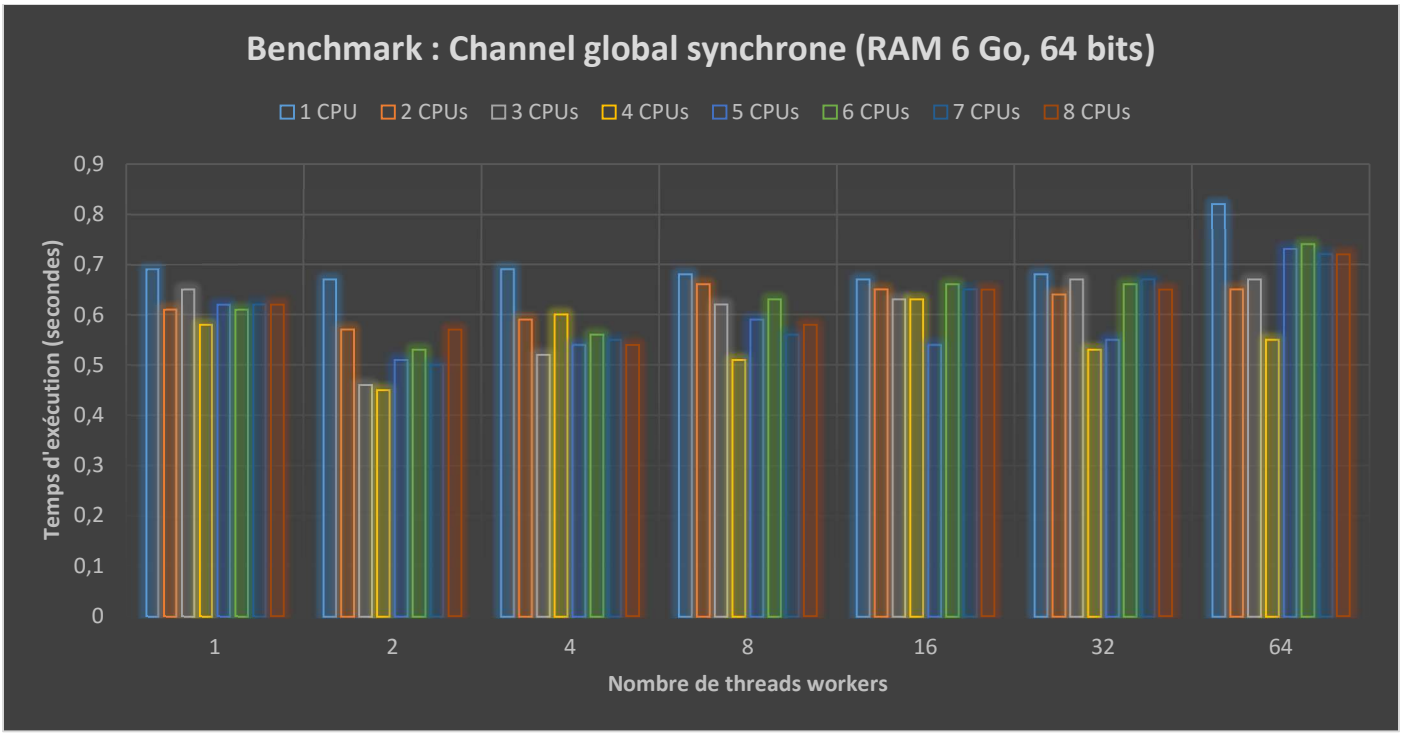
	1	2	3	4	5	6	7
1 thread	0,61	0,64	0,62	0,7	0,75	0,61	0,65
2 threads	0,51	0,49	0,51	0,75	0,51	0,57	0,49
3 threads	0,55	0,55	0,61	0,53	0,62	0,56	0,54
4 threads	0,54	0,59	0,55	0,5	0,58	0,58	0,57
5 threads	0,58	0,57	0,57	0,63	0,62	0,58	0,56
6 threads	0,61	0,6	0,62	0,65	0,61	0,62	0,61
7 threads	0,62	0,61	0,63	0,67	0,61	0,59	0,64
8 threads	0,63	0,61	0,66	0,65	0,61	0,59	0,59


Benchmark : Channel version minimale (RAM 6 Go, 64 bits)
1 CPU
2 CPUs
3 CPUs
4 CPUs
5 CPUs
6 CPUs
7 CPUs
8 CPUs
Temps d'exécution (secondes)
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
2
4
8
16
32
64
Nombre de threads workers


Benchmark : Channel tubes (RAM 6 Go, 64 bits)
1 CPU
2 CPUs
3 CPUs
4 CPUs
5 CPUs
6 CPUs
7 CPUs
8 CPUs
Temps d'exécution (secondes)
1,2
1
0,8
0,6
0,4
0,2
0
1
2
4
8
16
32
64
Nombre de threads workers


Benchmark : Channel global (RAM 6 Go, 64 bits)
1 CPU
2 CPUs
3 CPUs
4 CPUs
5 CPUs
6 CPUs
7 CPUs
8 CPUs
Temps d'exécution (secondes)
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
2
4
8
16
32
64
Nombre de threads workers


Benchmark : Channel synchrone (RAM 6 Go, 64 bits)
1 CPU
2 CPUs
3 CPUs
4 CPUs
5 CPUs
6 CPUs
7 CPUs
8 CPUs
Temps d'exécution (secondes)
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
2
4
8
16
32
64
Nombre de threads workers


Benchmark : Channel global synchrone (RAM 6 Go, 64 bits)
1 CPU
2 CPUs
3 CPUs
4 CPUs
5 CPUs
6 CPUs
7 CPUs
8 CPUs
Temps d'exécution (secondes)
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
2
4
8
16
32
64
Nombre de threads workers