TP Calcul parallèle

Clément Hue

name	matrix size	average running times (en s)	Gflops	speed up
Blocked (50)	1000	1,164233	1,71786918941483	
Naïve	1000	24,109539	0,082954717632718	
Register-optimised	1000	8,783417	0,227701815819515	
Local sum	1000	8,568809	0,233404665689246	
Naïve	1000	8,693642	0,230053181393943	277 %
Unfolded (4)	1000	8,5952	0,232688011913626	
Unfolded subblock 2x2	1000	4,330089	0,461884270739008	
Interchange	1000	1,123281	1,78049837930135	
Transposed B	1000	1,38042	1,44883441271497	
Blocked (10)	1000	1,267546	1,57785200694886	
Blocked (20)	1000	1,310211	1,52647169043765	
Blocked (50)	1000	1,159131	1,72543051648174	
Blocked (100)	1000	1,524282	1,31209316911175	
Blocked (250)	1000	1,664914	1,20126324843205	
BlockedRectangular (4 x 8)	1000	1,813603	1,1027771789085	
BlockedRectangular (8 x 8)	1000	1,246555	1,60442178644344	
BlockedRectangular (8 x 4)	1000	1,692852	1,18143818833542	
BlockedRectangular (40 x 8)	1000	0,890399	2,24618401413299	
Recursive	1000	11,831318	0,169042874175134	
Recursive blocked (4)	1000	1,201698	1,66431166565976	
Recursive blocked (8)	1000	1,277744	1,56525876857962	
Recursive blocked (16)	1000	1,346742	1,48506543940859	
Recursive blocked (32)	1000	1,347257	1,48449776100625	
Recursive rectangular (4 x 8)	1000	1,360642	1,46989435869244	
Recursive rectangular (8 x 8)	1000	1,369011	1,46090864134766	
Recursive rectangular (8 x 4)	1000	1,359778	1,47082832638857	
Recursive rectangular (40 x 8)	1000	1,169102	1,71071471950266	
Sustained performances (831002166000.000000)	1000	1,470553	1,36003258638077	

3) Au vue des résultats, les performances obtenues par la version naïve avec les optimisations sont proche de celle de la version register-optimised et local-sum, on en déduit que le compilateur optimise le code pour que les registres soient le plus utilisé.
On remarque également au vu des tests, qu'en découpant la matrice par blocks, on obtient de bien meilleurs performance, jusqu'à 10 fois plus rapide. Donc le compilateur ne suffit pas à l'optimisation, le programmeur doit intervenir pour améliorer les performances.. En découpant la matrice par blocks, ceux-ci sont suffisamment petit pour entrer dans les caches du processeur, Or le temps d'accès aux caches du processeur est bien plus rapide que celui en mémoire central.
Les méthodes récursive quand à elles, montrent de moins bonne performance, car le langage C est un langage impérative et non fonctionnel. Le langage C ajoute à la *stack* les fonctions et ses arguments appelées, en faisant de la récursivité, la *stack* peut devenir grande, impactant les performances.
Le parcours par transposé est également plus performant car on parcours les 2 matrices par ligne. Or le parcours par ligne est plus performant car les tableaux sont contigus en mémoire, il n'y a pas de saut d'un

emplacement mémoire vers un autre. Le parcours par colonne engendre des sauts, car on passe d'un tableau à un autre à chaque itération de boucle.

4) Pour obtenir de meilleur résultat on peut combiner la méthode de la transposer et la méthode des BlockRectangular 40x8