Mini-projet MPI: Modèle Numérique de Terrain

Objectifs du mini-projet

Ce mini-projet MPI a pour but de paralléliser l'algorithme de Darboux, utilisé pour calculer le remplissage des cuvettes de la carte. (Il pourra aussi être optimisé via OpenMPI si jamais ça prendrait beaucoup de temps à s'effectuer, notamment avec 2 cœurs – surtout si un non hyperthreadé est utilisé pour la parallélisation entre autres, le parallélisme n'est alors pas vraiment garanti.)

Présentation de la solution parallèle

Attention: dans darboux.c, **ne pas confondre darboux(mnt) avec darboux_ref(mnt).** Le second ne sert que de référence pour la comparaison du temps d'exécution de l'algorithme, alors que le premier est parallélisé avec MPI et OpenMP.

Cette solution parallèle s'exécute comme suit : tous les processus ouvrent le même fichier .mnt, mais seul le maître (de rang 0) s'occupe d'initialiser la variable m, qui a été allouée à tous. Après initialisation de m dans le processus racine, deux broadcasts sont effectuées afin de transmettre aux aures processus la taille de la matrice. Il en sera de même pour le terrain (initialisation par le processus 0, distribution aux autres processus – par un Scatter, cette fois).

Une fois leur matrice initialisée, les processeurs effectueront l'algorithme de Darboux sur leur propre matrice de données, qui leur sera au préalable donnée par le processus 0 par un Scatterv. Ceci consistera à envoyer les première/denière ligne(s) aux processus qui se suivent/précèdent de façon synchrone, d'en calculer leur valeur et de notifier s'il y avait modification ou non. Et ça, tant qu'il y en a eu. Une fois toutes les modifications terminées, les données sont rassemblées via un Gather sur le processus 0, qui en gèrera le résultat.

Problème : *les valeurs* ***ne sont pas les mêmes*** *en seconde partie de W, même si elles s'y approximent de peu pour la plupart. On n'a pas pu réussir à en trouver la source et la corriger à temps, malgré de nombreuses tentatives effectuées (donc ratées, car la plupart donnaient le même résultat).*

Mesures & analyses de performance

Pour mesurer les performances des implémentations séquentielle (de référence) et MPI de l'algorithme, on calcule le temps d'exécution de la fonction de référence (darboux_ref) spécialement réinsérée pour placer une référence de temps, et celui de l'implémentation MPI (kernel) : de là, on pourra calculer son speedup.

Mesures des performances

			Performances testées et temps obtenus sur des machines, à titre d'exemples :			
			Non hyperthreadés		Hyperthreadés	
			2 coeurs	3 machines de Turing, 1 slot chacun	4 coeurs, 8 processus	? machines de Turing, ? slots chacun
Calcul des performances du MNT de dimensions :	Mini	Référence	**0.00084s**	**0.0031s**	**0.00013s**	**?**
		Parallélisation	**0.02304s**	**0.00070s**	**0.0916s**	**?**
		Speedup	**0.03649**	**0.44275**	**0.00146** / 0.00141921397	**?**
		Efficacité			**0.00018**	**?**
	Small	Référence	**1.34789s**	**0.32262s**	**0.15781s**	**?**
		Parallélisation	**0.74923s**	**0.10697s**	**0.18442s**	**?**
		Speedup	**1.79904**	**3.01586**	**0.85572** / 0.85570979286	**?**
		Efficacité			**0.10696**	**?**
	Medium	Référence	**3m 41.66364s**	**1m 8.14439s**	**32.55601s**	**?**
		Parallélisation	**1m 29.2352s**	**22.63932s**	**10.37022s**	**?**
		Speedup	**2.48404**	**3.01**	**3.13938** / 10.7358593803	**?**
		Efficacité			**0.39242**	**?**
	Large	Référence	**44m 9.32928s**	**18m 46.50592s**	**13m 26s**	**?**
		Parallélisation	**15m 24.73629s**	**6m 26.97673s**	**4m 19s**	**?**
		Speedup	**2.86496**	**2.91104**	**~3.1** / 8,13	**?**
		Efficacité			**~0.3**	**?**

Annexe : Comparaison des performances testées et des temps obtenus selon les machines utilisées et de leur nombre de coeurs threadés *En gris se trouvent d'autres valeurs trouvées sur une même machine.*

Problème : *Lancer le programme sur Turing ne fonctionnait pas comme il faut lors de nos tests finaux – surtout avec le même nombre de coeurs que nos PCs locaux ! Entre déconnexions et combo « Operation in progress » - « Resource temporarily unavailable », on était servis.*

Analyse des performances

Comme ça peut se remarquer sur le tableau ci-dessus (enfin, de ce qu'on a pu tester), ce n'est pas toujours fameux en terme de temps d'exécution et de speedup. Cela est très probablement dû à des échanges qui soit ne sont pas passés correctement (d'où l'affichage final raté vers la fin), ou à certains échanges inutiles, même si on doit admettre que la première colonne de résultats est testée à partir une machine qui n'est pas une bête de

guerre. Il y a plus optimal que ce qu'on a pu faire, on doit admettre. Surtout qu'en terme de speedup, un meilleur avertissement général de la modification d'un Wprec aurait pu être implémenté avec un moins grand nombre d'envois par tour de boucle while. Par ce moyen, l'efficacité aurait probablement été meilleure. On aurait pu améliorer la distribution des cases entre les processus, tant bien que leur rassemblement dans le processus 0.