Sequential and Parallel BSA Algorithm

Mathieu Brévilliers, Omar Abdelkafi, Lhassane Idoumghar

Université de Haute-Alsace (UHA)
LMIA (E.A. 3993), 4 rue des frères Lumière, 68093 Mulhouse, France
{mathieu.brevilliers, omar.abdelkafi, lhassane.idoumghar}@uha.fr

Mots-clés : *métaheuristique, algorithme évolutionnaire, GPU, CUDA.*

1 Introduction

L'algorithme BSA (Backtracking Search Optimization Algorithm) [1] est un algorithme évolutionnaire conçu pour résoudre des problèmes d'optimisation continue. Une des particularités de BSA réside dans l'utilisation d'une mémoire, pour conserver une population antérieure, qui intervient dans l'opérateur de mutation mis en oeuvre. Il est par ailleurs conçu autour d'un unique paramètre de contrôle qui est utilisé au cours du processus de croisement. La structure de BSA est simple, et les opérateurs de mutation et de croisement utilisés lui permettent de s'adapter à des problèmes variés et de les résoudre efficacement. L'algorithme BSA a été testé sur de nombreux benchmarks mathématiques largement utilisés dans la littérature, et il s'est montré généralement meilleur que de nombreux autres algorithmes de la littérature (SPSO2011, CMAES, ABC, JDE, CLPSO et SADE). Depuis sa publication en 2013, l'algorithme BSA a été utilisé dans divers domaines d'application et a fait l'objet de plusieurs évolutions [2].

2 Modification de BSA

Parmi les améliorations apportées à l'algorithme BSA, on peut citer les suivantes :

- Nous avons introduit un critère de diversification qui consiste à regénérer les deux tiers de la population si au bout de 100 itérations, on n'observe pas d'amélioration.
- Le générateur de nombres aléatoires basé sur la distribution normale est remplacé par le générateur rand53 issu de Mersenne Twister.
- Dans l'opérateur de croisement, le masque s'applique avec une probalité $0,5$.

L'analyse des résultats donnés dans le Tableau 1 montre qu'on améliore la qualité des solutions sur plusieurs benchmarks mathématiques.

3 Implémentation sur GPU

Les algorithmes à base de population sont particulièrement adaptés au parallélisme et peuvent être implémentés efficacement en utilisant les processeurs des cartes graphiques récentes (GPU) [3]. Plus particulièrement, la technolgie CUDA développée par NVIDIA permet d'exécuter des calculs généraux en profitant de l'architecture parallèle de ses cartes graphiques.

Nous avons implémenté l'algorithme BSA avec CUDA et nous l'avons testé sur un ordinateur muni d'un processeur central Intel Core i5-3330 cadencé à 3 GHz, de 4 Go de RAM, et d'une carte graphique NVIDIA GeForce GTX 680. Nous avons choisi d'évaluer les performances de la version GPU de BSA sur des benchmarks mathématiques couramment utilisés, et nous les avons traités en grandes dimensions avec des populations de grandes tailles. Le Tableau 2 donne un aperçu des résultats obtenus avec les fonctions Ackley, Michalewicz et Rastrigin : l'analyse des accélérations obtenues montre que l'algorithme BSA s'exécute entre 3 et 10 fois plus vite sur GPU que sur CPU.

4 Conclusions et perspectives

Ce travail montre que notre version de l'algorithme BSA permet d'améliorer la qualité des solutions obtenues sur quelques benchmarks mathématiques. Nous avons également montré

Fonction	P	D		BSA modifié	BSA	CMAES	SPSO2011
Ackley	30	30	Moy	$2,034334 \times 10^{-14}$	$1,050000 \times 10^{-14}$	$1,170400 \times 10^{+01}$	$1,521432 \times 10^{+00}$
			E	$5,727308 \times 10^{-15}$	$3,400000 \times 10^{-15}$	$9,720196 \times 10^{+00}$	$6,617570 \times 10^{-01}$
			Min	$1,406690 \times 10^{-14}$	$8,000000 \times 10^{-15}$	$8,000000 \times 10^{-15}$	$8,000000 \times 10^{-15}$
Michalewicz	30	2	Moy	$\mathbf{-1,801300 \times 10^{+00}}$	$-1,821044 \times 10^{+00}$	$-1,782927 \times 10^{+00}$	$-1,821044 \times 10^{+00}$
			E	$\mathbf{6,775215 \times 10^{-16}}$	$9,000000 \times 10^{-16}$	$1,450584 \times 10^{-01}$	$9,000000 \times 10^{-16}$
			Min	$\mathbf{-1,801300 \times 10^{+00}}$	$-1,821044 \times 10^{+00}$	$-1,821044 \times 10^{+00}$	$-1,821044 \times 10^{+00}$
Michalewicz	30	5	Moy	$\mathbf{-4,687658 \times 10^{+00}}$	$-4,693468 \times 10^{+00}$	$-4,100895 \times 10^{+00}$	$-4,656565 \times 10^{+00}$
			E	$\mathbf{0,000000 \times 10^{+00}}$	$8,000000 \times 10^{-16}$	$4,951250 \times 10^{-01}$	$5,570215 \times 10^{-02}$
			Min	$\mathbf{-4,687658 \times 10^{+00}}$	$-4,693468 \times 10^{+00}$	$4,693468 \times 10^{+00}$	$-4,693468 \times 10^{+00}$
Michalewicz	30	10	Moy	$\mathbf{-9,660152 \times 10^{+00}}$	$-9,660152 \times 10^{+00}$	$-7,619351 \times 10^{+00}$	$-8,971733 \times 10^{+00}$
			E	$1,806724 \times 10^{-15}$	$7,000000 \times 10^{-16}$	$7,904830 \times 10^{-01}$	$4,927013 \times 10^{-01}$
			Min	$\mathbf{-9,660152 \times 10^{+00}}$	$-9,660152 \times 10^{+00}$	$-9,138398 \times 10^{+00}$	$-9,577782 \times 10^{+00}$
Rastrigin	30	30	Moy	$\mathbf{0,000000 \times 10^{+00}}$	$0,000000 \times 10^{+00}$	$9,597999 \times 10^{+01}$	$2,563676 \times 10^{+01}$
			E	$\mathbf{0,000000 \times 10^{+00}}$	$0,000000 \times 10^{+00}$	$5,669192 \times 10^{+01}$	$8,294351 \times 10^{+00}$
			Min	$\mathbf{0,000000 \times 10^{+00}}$	$0,000000 \times 10^{+00}$	$2,984876 \times 10^{+01}$	$1,293447 \times 10^{+01}$
Schwefel_2_22	30	30	Moy	$\mathbf{0,000000 \times 10^{+00}}$	$0,000000 \times 10^{+00}$	$0,000000 \times 10^{+00}$	$0,000000 \times 10^{+00}$
			E	$\mathbf{0,000000 \times 10^{+00}}$	$0,000000 \times 10^{+00}$	$0,000000 \times 10^{+00}$	$0,000000 \times 10^{+00}$
			Min	$\mathbf{0,000000 \times 10^{+00}}$	$0,000000 \times 10^{+00}$	$0,000000 \times 10^{+00}$	$0,000000 \times 10^{+00}$

TAB. 1 – Comparaison de notre algorithme BSA modifié avec BSA, CMAES, et SPSO2011 (résultats issus de [1]). Pour chaque fonction et pour chaque couple (P,D), notre algorithme a été exécuté 30 fois : l'algorithme s'arrête après 2×10^6 appels à la fonction d'évaluation. P : taille de la population, D : dimension du problème, Moy : moyenne des solutions, E : écart-type, Min : meilleure solution.

			Ackley		Michalewicz		Rastrigin	
P	D		CPU	GPU	CPU	GPU	CPU	GPU
256	256	Moy	$6,176 \times 10^{-04}$	$4,217 \times 10^{-03}$	$-2,000 \times 10^{+02}$	$-1,911 \times 10^{+02}$	$1,316 \times 10^{+02}$	$5,078 \times 10^{+02}$
		E	$5,062 \times 10^{-04}$	$2,665 \times 10^{-03}$	$1,173 \times 10^{+00}$	$2,307 \times 10^{+00}$	$1,635 \times 10^{+01}$	$3,442 \times 10^{+01}$
		Min	$1,267 \times 10^{-04}$	$7,581 \times 10^{-04}$	$-2,020 \times 10^{+02}$	$-1,961 \times 10^{+02}$	$9,400 \times 10^{+01}$	$4,326 \times 10^{+02}$
		T	23,88	**14,45**	90,34	**14,94**	28,41	**13,68**
512	512	Moy	$2,607 \times 10^{+00}$	$4,624 \times 10^{+00}$	$-2,326 \times 10^{+02}$	$-2,207 \times 10^{+02}$	$2,048 \times 10^{+03}$	$2,157 \times 10^{+03}$
		E	$6,908 \times 10^{-01}$	$9,213 \times 10^{-01}$	$3,180 \times 10^{+00}$	$3,514 \times 10^{+00}$	$1,722 \times 10^{+02}$	$\mathbf{1,106 \times 10^{+02}}$
		Min	$1,337 \times 10^{+00}$	$2,800 \times 10^{+00}$	$-2,443 \times 10^{+02}$	$-2,277 \times 10^{+02}$	$1,704 \times 10^{+03}$	$1,968 \times 10^{+03}$
		T	62,26	**16,82**	184,93	**18,55**	61,67	**16,09**
1024	1024	Moy	$1,041 \times 10^{+01}$	$\mathbf{1,027 \times 10^{+01}}$	$-2,597 \times 10^{+02}$	$-2,494 \times 10^{+02}$	$9,234 \times 10^{+03}$	$\mathbf{8,472 \times 10^{+03}}$
		E	$2,064 \times 10^{+00}$	$\mathbf{1,840 \times 10^{+00}}$	$3,979 \times 10^{+00}$	$\mathbf{3,881 \times 10^{+00}}$	$7,613 \times 10^{+02}$	$\mathbf{4,260 \times 10^{+02}}$
		Min	$6,370 \times 10^{+00}$	$6,385 \times 10^{+00}$	$-2,693 \times 10^{+02}$	$-2,598 \times 10^{+02}$	$7,961 \times 10^{+03}$	$\mathbf{7,354 \times 10^{+03}}$
		T	129,36	**33,40**	376,66	**36,85**	129,31	**31,29**

TAB. 2 – Comparaison entre les implémentations CPU et GPU de l'algorithme BSA. Pour chaque fonction et pour chaque couple (P,D), l'algorithme a été exécuté 30 fois. L'algorithme s'arrête après 2×10^6 appels à la fonction d'évaluation. P : taille de la population, D : dimension du problème, Moy : moyenne des solutions, E : écart-type, Min : meilleure solution, T : temps moyen en secondes.

que l'implémentation sur GPU de cet algorithme permet d'accélérer la résolution de problèmes de grande taille. Nous réalisons actuellement une étude plus approfondie pour nous assurer que l'implémentation GPU trouve des solutions de meilleure qualité que ce qui est observé dans le cas de l'algorithme séquentiel. Par ailleurs, l'implémentation actuelle sur GPU n'exploite pas encore toutes les possibilités offertes par CUDA, ce qui laisse à penser que les calculs pourraient encore être accélérés.

Références

[1] P. Civicioglu. Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation*, 219(15) :8121 – 8144, 2013.

[2] S. Das, D. Mandal, R. Kar, and S. Prasad Ghoshal. A new hybridized backtracking search optimization algorithm with differential evolution for sidelobe suppression of uniformly excited concentric circular antenna arrays. *International Journal of RF and Microwave Computer-Aided Engineering*, 2014.

[3] G. Luo, S. Huang, Y. Chang, and S. Yuan. A parallel bees algorithm implementation on gpu. *Journal of Systems Architecture*, 60(3) :271 – 279, 2014. Real-Time Embedded Software for Multi-Core Platforms.