A Comparison Exercise on Parallel Evaluation of Rosenbrock Function

Miguel Cárdenas-Montes
CIEMAT
Department of Fundamental Research
Avda. Complutense 40, 28040
Madrid, Spain
miguel.cardenas@ciemat.es

Miguel Angel Vega-Rodríguez
University of Extremadura
ARCO Research Group
Dept. Technologies of Computers and Communications
Cáceres, Spain
mavega@unex.es

Juan José Rodríguez-Vázquez
CIEMAT
Department of Fundamental Research
Avda. Complutense 40, 28040
Madrid, Spain
jj.rodriguez@ciemat.es

Antonio Gómez-Iglesias
Texas Advanced Computing Center
The University of Texas at Austin
Austin, TX, USA
agomez@tacc.utexas.edu

ABSTRACT

GPU computing has spread its capacity over most of the scientific computing areas. Soft computing is aware of the potential of this computing architecture. In order to achieve high performance, practitioners have to deal with the particularities associated with the porting of the problem to the specifications of the GPU card; and specially with an efficient management of the on-board and on-chip memories. In this work, diverse optimized data layouts are analysed when evaluating a large population of high-dimensional individuals for Rosenbrock function. As a consequence of this study, a statement about the most favourable data layout for this kind of evaluation is presented.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures; C.4 [**Performance of Systems**]; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

Keywords

Non-separable function; GPU performance; parallel evaluation; Rosenbrock function

1. INTRODUCTION

GECCO'15 Companion, July 11–15, 2015, Madrid, Spain.

ACM ISBN 978-1-4503-3488-4/15/07.

DOI: http://dx.doi.org/10.1145/2739482.2764641

Due to the excellent ratio between capacity and cost, GPU computing (graphics processing unit) has been become popular for supporting scientific computing in many areas. In soft computing, GPU computing is being used for accelerating algorithms optimizing complex problems. This is the case when evaluating population-based evolutionary algorithms with large population of high-dimensional individuals. Generally in these cases, the evaluation of the fitness is the most time consuming, and therefore, the first target for reducing the processing time.

Not only the dimensionality of individuals and the size of the population increase the processing time, also some characteristics of the function which is being optimized are relevant. Concerning the artificial continuous functions, the non-separable functions are considered more difficult to minimize than the separable ones. For this reason, in this work a population of 5,000 individuals of 5,000 dimensions is used to evaluate a well-known non-separable function: the Rosenbrock function (Eq. 1).

$$f_{Rosenbrock} = \sum_{i=1}^{D-1} 100 \cdot [(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \qquad (1)$$

In order to efficiently evaluate this kind of problems, parallel paradigms can be used to alleviate the large processing time. When using GPU computing for this purpose, an adequate management of the on-board and on-chip memories is mandatory to achieve an efficient usage of the computational resources.

Regarding parallel evolutionary algorithms, three main models can be enunciated [1]. On the one hand, the model used in the current work in which only the evaluation of the population is executed in parallel, whereas the remaining operators of the algorithm are sequentially executed. The

two other models are island a/synchronous cooperation and the distributed evaluation of a single solution.

In this work, an in-depth study about the most appropriate data layout for efficiently evaluating a non-separable function on GPU for a large population of high-dimensional individuals is proposed. Particularly, the study focuses on the best profit of the on-board memory: coalesced access to global memory; and the on-chip memory: registers, shared memory, and L1 cache.

The strategies evaluated in this work are: **S1** allocation of one individual per thread on registers, **AS3** allocation of one individual per thread-block on share memory with coalesced access to global memory and atomic operations, and **S4** allocation of one individual per thread on registers with coalesced access to global memory.

All the numerical experiments have been executed with floating point precision in an NVIDIA C2075 card (Fermi architecture) with CUDA release 5.0 and compute capability 2.0.

2. RESULTS AND ANALYSIS

The analysis of the processing time (Table 1) for the refinements applied to the S1 (vectorization of the reads with float2 and float4) results in a relevant reduction of the original processing time when applying vectorization. As a result of the vectorization with *float4*, the stride pattern access is mitigated with the improvement of the use of the global memory bandwidth.

Table 1: Execution time (ms) for GPU-based evaluation of the Rosenbrock function. Average times and standard deviation after 25 executions per case.

Strategy	Execution Time
S1	21.716 ± 0.048
S1 vectorized float2	21.738 ± 0.069
S1 vectorized float4	8.862 ± 0.024
AS3	57.530 ± 2.244
S4	$\mathbf{6.921 \pm 0.014}$
S4 vectorized float2	11.932 ± 0.015
S4 vectorized float4	22.946 ± 0.021
Sequential	51.698 ± 1.098

On the other hand, the modification on AS3 strategy sets up the possibility to handle individuals larger than the maximum number of threads per block, but unfortunately by introducing an elevated cost in processing time.

Considering the processing time of S4 with and without vectorization, it is observed that the original S4 implementation is the most efficient implementation among all those evaluated in this work. Since S4 has already a coalesced access to global memory, the vectorization of this implementation penalizes its efficiency.

Regarding the L1 cache memory configuration, when reducing the shared memory to 16 KB (Table 2) the processing times are similar to the ones obtained with 48 KB of shared memory. On the other hand, if the strategy has a simple enough memory access pattern, then the explicit caching of global memory in shared memory through L1 turn-off could increment the performance. When turning off the L1 cache memory (Table 3), some strategies improve but not enough to outperform the most suitable strategy until now.

Table 2: Execution time (ms) for GPU-based evaluation of the Rosenbrock function with 16 KB of shared memory configuration. Average times and standard deviation after 25 executions per case.

Strategy	Execution Time
S1	21.665 ± 0.040
S1 vectorized float2	21.698 ± 0.057
S1 vectorized float4	8.861 ± 0.022
AS3	52.689 ± 0.528
S4	6.93 ± 0.017
S4 vectorized float2	11.928 ± 0.015
S4 vectorized float4	22.937 ± 0.010

Table 3: Execution time (ms) for GPU-based evaluation of the Rosenbrock function with L1 cache memory turned-off configuration. Average times and standard deviation after 25 executions per case.

Strategy	Execution Time
S1	14.216 ± 0.029
S1 vectorized float2	14.146 ± 0.021
S1 vectorized float4	12.903 ± 0.024
AS3	64.429 ± 1.174
S4	6.965 ± 0.051
S4 vectorized float2	11.935 ± 0.027
S4 vectorized float4	22.628 ± 0.030

As a conclusion of the Rosenbrock function analysis, it can be stated that the S4 strategy provides the best performance for evaluating it, $\mathbf{6.921 \pm 0.014}$ ms.

3. CONCLUSIONS AND FUTURE WORK

In this work the most efficient data layouts for efficiently evaluating large instances of Rosenbrock function on GPU have been analysed. As example of large instances a population of 5,000 individuals of 5,000 dimensions has been selected.

From the processing time point of view, it has been shown that an incorrect choice of the data layout will penalize the performance of the algorithm during the evaluation of the population. The S4 strategy (allocation of one individual per thread on registers with coalesced access to global memory) outperforms all the other strategies including the refinements checked in this work. The a priory knowledge of the most suitable data layout will permit the design of more efficient evolutionary algorithms.

4. ACKNOWLEDGMENTS

The research leading to these results has received funding by the Spanish Ministry of Economy and Competitiveness (MINECO) for funding support through the grant FPA2013-47804-C2-1-R, together with the European Community's Seventh Framework Programme (FP7/2007-2013) via the project EGI-InSPIRE under the grant agreement number RI-261323.

5. REFERENCES

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 6(5):443–462, 2002.