Fast Prediction Computation in Learning Classifier Systems Using CUDA

Daniele Loiacono
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano 20133, Italy
loiacono@elet.polimi.it

ABSTRACT

Computing the system prediction is one of the most important and computationally expensive tasks in Learning Classifier Systems. In this paper, we provide a parallel solution to the problem of computing the prediction array in XCS using the NVIDIA's Compute Unified Device Architecture (CUDA). We performed several experiments to test our parallel solution using two different types of GPUs and to study how performances are affected by (i) the problem size, (ii) the number of problem actions, and (iii) the number of classifiers in the population. Our experimental results show a speedup that ranges from slightly less than 2× up to 32×.

Categories and Subject Descriptors

F.1.1 [**Models of Computation**]: Genetics Based Machine Learning, Learning Classifier Systems

General Terms

Algorithms, Performance

Keywords

LCS, CUDA, GPU, Prediction Array

1. INTRODUCTION

Learning classifier systems [5, 3, 8] are rule-based evolutionary systems that can solve classification as well as reinforcement learning problems by evolving a population of classifiers, i.e., condition-action-prediction rules.

Recently, Llorà and Sastry [7] showed that matching classifiers and computing the prediction is the most computational expensive step in learning classifier systems and can take up to 80% of the overall computational time [7]. Accordingly, several methods have been proposed in the literature to speed up matching in learning classifier systems [7, 1, 6, 2].

In an early work Lanzi and Loiacono [6], used the NVIDIA's Compute Unified Device Architecture (CUDA) to implement matching for real inputs using interval-based conditions and for binary inputs using ternary conditions. Later, Franco et al. [2] applied CUDA to speed-up the evaluation of rules in BioHel, an evolutionary learning system specifically devised to deal with large datasets. In this work, following an approach similar to [2], we take the work of Lanzi and Loiacono [6] a step further: while in [6] CUDA has been applied only to build the match set, here GPUs are exploited also to compute the prediction array. In particular, we focus on the XCS classifier system [8] applied to problems involving real inputs.

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

2. COMPUTING THE PREDICTION ARRAY WITH GPUS

Our CUDA implementation builds the prediction array in three main steps: (i) matching the current input and computing accordingly the contribution of *each* classifier to the prediction array; (ii) *reducing* (i.e., summing up) the contributions of all the classifiers in each CUDA's thread block; (iii) computing the final prediction array.

In the first step we exploit the parallel computing capabilities offered by GPUs to perform the classifiers' matching and to compute the contribution of *each* classifier to the prediction array. In the second step we take advantage of the GPUs to sum up the contributions to the prediction array of the classifiers in each block of threads (namely, we applied the *Sequential Addressing* [4] parallel reduction schema). Finally, the contributions of each block of thread are combined together on the CPU and the final result is computed.

3. EXPERIMENTAL RESULTS

We performed a set of experiments to assess the performance of our CUDA implementation with respect to a CPU implementation. We generated a population of N classifiers with interval-based conditions [9] of length n and 1000 random input configurations. We build the prediction array for each random input configuration and measured the average time required. We repeated each experiment 10 times.

Experiments have been carried on a server with 2 quadcore Xeon (2.66 GHz), 8GB of RAM, running Linux Fedora Core 6 and equipped with a *Tesla C1060* and a *GeForce GTX 470*. The performance was measured as the average wall clock time to build the 1000 prediction arrays over a population of N classifiers. In particular, performance takes into account (i) the time to load each one of the 1000 inputs to be matched into the GPU; and (ii) the time to move the result data structures from the GPU to main CPU memory. Table 1 reports the average computation time on the CPU, on `Tesla C1060` and on `GeForce GTX 470` for problems (i) with a number of inputs n, chosen in {10, 50, 100} and (ii) involving a number of action m in {2, 4, 8}; the population

n	m	CPU	Tesla C1060	GeForce GTX 470
10	2	0.261 ± 0.000	0.147 ± 0.002	0.092 ± 0.002
10	4	0.261 ± 0.000	0.164 ± 0.002	0.102 ± 0.003
10	8	0.261 ± 0.000	0.215 ± 0.002	0.139 ± 0.003
50	2	0.797 ± 0.006	0.266 ± 0.002	0.143 ± 0.000
50	4	0.793 ± 0.007	0.283 ± 0.003	0.152 ± 0.002
50	8	0.795 ± 0.004	0.336 ± 0.002	0.190 ± 0.000
100	2	2.242 ± 0.011	0.419 ± 0.001	0.208 ± 0.003
100	4	2.282 ± 0.045	0.435 ± 0.002	0.216 ± 0.000
100	8	2.244 ± 0.009	0.487 ± 0.001	0.254 ± 0.001

(a)

n	m	CPU	Tesla C1060	GeForce GTX 470
10	2	3.029 ± 0.019	0.608 ± 0.002	0.268 ± 0.002
10	4	3.017 ± 0.016	0.757 ± 0.002	0.321 ± 0.001
10	8	3.024 ± 0.009	1.161 ± 0.003	0.529 ± 0.001
50	2	15.003 ± 0.077	1.821 ± 0.001	0.587 ± 0.003
50	4	15.013 ± 0.074	1.969 ± 0.001	0.640 ± 0.003
50	8	15.039 ± 0.128	2.376 ± 0.001	0.849 ± 0.003
100	2	29.511 ± 0.095	3.316 ± 0.001	0.980 ± 0.003
100	4	29.515 ± 0.075	3.465 ± 0.001	1.032 ± 0.001
100	8	29.510 ± 0.090	3.864 ± 0.001	1.239 ± 0.001

(b)

n	m	CPU	Tesla C1060	GeForce GTX 470
10	2	34.025 ± 0.061	5.217 ± 0.002	2.067 ± 0.001
10	4	34.014 ± 0.062	6.676 ± 0.002	2.576 ± 0.003
10	8	34.057 ± 0.054	11.048 ± 0.002	4.395 ± 0.010
50	2	151.962 ± 0.234	17.622 ± 0.005	5.144 ± 0.002
50	4	151.866 ± 0.281	19.082 ± 0.003	5.654 ± 0.002
50	8	151.864 ± 0.403	23.469 ± 0.012	7.470 ± 0.004
100	2	296.013 ± 0.712	30.228 ± 0.026	8.977 ± 0.005
100	4	296.154 ± 0.623	31.663 ± 0.020	9.487 ± 0.008
100	8	295.622 ± 0.455	35.957 ± 0.011	11.303 ± 0.005

(c)

Table 1: Time (in seconds) required to build the prediction array for 1000 instances when the problem involves 10, 50 or 100 real inputs and 2, 4, or 8 actions; the population size N is (a) 10000, (b) 100000, and (c) 1000000; statistics are averages over 10 runs.

size N is either 10000 (Table 1a), 100000 (Table 1b), or 1000000 (Table 1c).

The results show that the CUDA implementation always outperforms the CPU implementation and that the `GeForce GTX 470` always outperforms the `Tesla C1060`. In addition, the speedup changes with respect to the number of actions (m) and to the number of classifiers in the population (N). Notice that only on larger problems (i.e., when $n = 50$ or $n = 100$) the computation on the GPUs results in a very significant speedup, that is up to 32× on the `GeForce GTX 470` and up to 10× on the `Tesla C1060`. Finally, while the number of actions m does not affect the performance on the CPU, on the GPUs the higher is m the worse is the performance.

4. CONCLUSIONS

In this paper, we took the the work of Lanzi and Loiacono [6] a step further by exploiting the massive parallelism available on GPUs to compute the prediction array in XCS. Although our approach can be applied to any encoding of the classifier condition, here we focused on the interval-based encoding, introduced in [9] to deal with real inputs. We tested our parallel implementation on two different GPUs and compared it to a CPU implementation with several configurations. The results of the experimental analysis were very promising: (i) the computation on GPUs resulted, in our experimental setup, always faster with respect to the CPU and (ii) the speedup obtained against the CPU-based implementation is up to 32×; in particular, the larger is the population of classifiers the larger is the speedup; on the other hand, the larger is the number of problem actions, the worse is the speedup achieved on the GPUs.

5. REFERENCES

[1] Martin V. Butz, Pier Luca Lanzi, Xavier Llorà, and Daniele Loiacono. An analysis of matching in learning classifier systems. In Conor Ryan and Maarten Keijzer, editors, *Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings, Atlanta, GA, USA, July 12-16, 2008*. ACM Press, 12-16 July 2008.

[2] María A. Franco, Natalio Krasnogor, and Jaume Bacardit. Speeding up the evaluation of evolutionary learning systems using gpgpus. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 1039–1046, New York, NY, USA, 2010. ACM.

[3] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.

[4] Mark Harris. Optimizing parallel reduction in cuda, 2007. `http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf`.

[5] John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. 1978. Reprinted in: Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN: 0-7803-3481-7.

[6] Pier Luca Lanzi and Daniele Loiacono. Speeding up matching in learning classifier systems using cuda. In Jaume Bacardit, Will N. Browne, Jan Drugowitsch, Ester Bernadó-Mansilla, and Martin V. Butz, editors, *IWLCS*, volume 6471 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2009.

[7] Xavier Llorà and Kumara Sastry. Fast rule matching for learning classifier systems via vector instructions. In Mike Cattolico, editor, *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1513–1520, New York, NY, USA, 2006. ACM Press.

[8] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[9] Stewart W. Wilson. Mining oblique data with xcs. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *IWLCS*, volume 1996 of *Lecture Notes in Computer Science*, pages 158–176. Springer, 2000.