GPU-based Parallel Hybrid Genetic Algorithms

Thé Van Luong, Nouredine Melab, El-Ghazali Talbi
INRIA Lille Nord Europe / CNRS LIFL FRANCE
The-Van.Luong@inria.fr, Nouredine.Melab@lifl.fr, El-Ghazali.Talbi@lifl.fr

***Abstract*—Over the last years, interest in hybrid metaheuristics has risen considerably in the field of optimization. Combinations of algorithms such as genetic algorithms (GAs) and local search (LS) methods have provided very powerful search algorithms. However, due to their complexity, the computational time of the solution search exploration remains exorbitant when large problem instances are to be solved. Therefore, the use of GPU-based parallel computing is required as a complementary way to speed up the search. This paper presents a new methodology to design and implement efficiently and effectively hybrid genetic algorithms on GPU accelerators.**

I. Scheme of parallelization

The adaptation of hybrid GAs on GPU requires to take into account at the same time the characteristics and underlined issues of the GPU architecture and the metaheuristics parallel models. Since the evaluation of the neighborhood is generally the time-consuming part of hybrid GAs, we focus on the re-design of LS algorithms on GPU (see Fig. 1). We propose a three-level decomposition of the GPU adapted to the popular parallel iteration-level model [1] (generation and evaluation of the neighborhood in parallel) allowing a clear separation of the GPU memory hierarchical management concepts (see Fig. 2).

In the high-level layer, the CPU sends the number of expected running threads to the GPU, then candidate neighbors are generated and evaluated on GPU (at intermediate-level and low-level), and finally newly evaluated solutions are returned back to the host. This model can be seen as a cooperative model between the CPU and the GPU. Indeed, the GPU is used as a coprocessor in a synchronous manner. The resource-consuming part i.e. the incremental evaluation kernel is calculated by the GPU and the rest is handled by the CPU.

The intermediate-level layer focuses on the generation of the LS neighborhood on GPU. This generation is performed in a dynamic manner which implies that no explicit structure needs to be allocated or copied (unlike traditional GAs on GPU [2]). Thereby, only the representation of this candidate solution must be copied from the CPU to the GPU. Therefore, the main difficulty of the intermediate-level layer is to find an efficient mapping between a GPU thread and a LS neighbor candidate solution. In other words, the issue is to say which solution must be handled by which thread. The answer is dependent of the solution representation.

Afterwards, GPU memory management of the evaluation function computation is done at low-level. The use of texture memory is a solution for reducing memory transactions due to non-coalesced accesses (matrices, solution which generates the neighborhood). Indeed, texture memory can be seen as a relaxed mechanism for the thread processors to access global memory because the coalescing requirements do not apply to texture memory accesses.

II. Experimentation

A. Configuration

For this problem, a hybrid GA with an iterative local search (ILS) has been implemented on GPU with CUDA for the quadratic assignment problem. The embedded method is a tabu search (TS) and the perturbation applied in the ILS process uses a random number of pair-wise exchanges. The number of ILS iterations has been fixed to 3 and the number of TS iterations to 10000. The chosen neighborhood is based on a Hamming distance of 3 where generating a neighbor is obtained by performing two swaps from the initial solution. The tabu list size has been set to $2 \times \lfloor \sqrt{\frac{n \times (n-1) \times (n-2)}{6}} \rfloor$.

Regarding the GA, first, random initializations are performed with a population of 10 individuals. Second, the selection operator is a deterministic tournament with a size of 2. Third, the crossover selects the common attributes in both parents and the remaining entries are chosen at random (crossover rate fixed to 100%). Fourth comes the hybridization where the mutation operator is replaced by the ILS-TS (mutation rate fixed to 100%). Finally, the replacement strategy is a generational replacement and the number of generations has been fixed to 10.

The used configuration for the experiments is a Core 2 Duo 2.67Ghz with a NVIDIA GTX 280 card (30 multiprocessors).

B. Measures in Terms of Efficiency and Effectiveness

For each instance, a standalone mono-core CPU implementation, a CPU-GPU, and a CPU-GPU version using texture memory are considered. The average time has been measured for 30 runs. Average values of the evaluation function have been collected and the number of successful tries (hits) is also represented. The associated standard deviation for each average measurement is shown in sub-index. Table I reports the obtained results.

Generate and evaluate the neighborhood in parallel on GPU provides an efficient way to speed-up the search process in comparison with a single CPU. Indeed, from the instance tai30a, the GPU version is already faster than the CPU one (acceleration factor of $\times 5.2$). As long as the problem size increases, the speed-up grows significantly (up to $\times 7.2$ for the tai100a instance).

Due to high misaligned accesses to global memories (flows and distances in QAP), non-coalescing memory reduces the performance of the GPU implementation. Binding texture on global memory allows to overcome the problem. Indeed, from the instance tai30a, using texture memory starts providing significant acceleration factor of $\times 8.5$. GPU keeps accelerating the hybrid genetic process as long as the size grows (up to $\times 14.6$).

Regarding the quality of solutions, in comparison with the literature [3], the obtained results by the proposed hybrid GA is

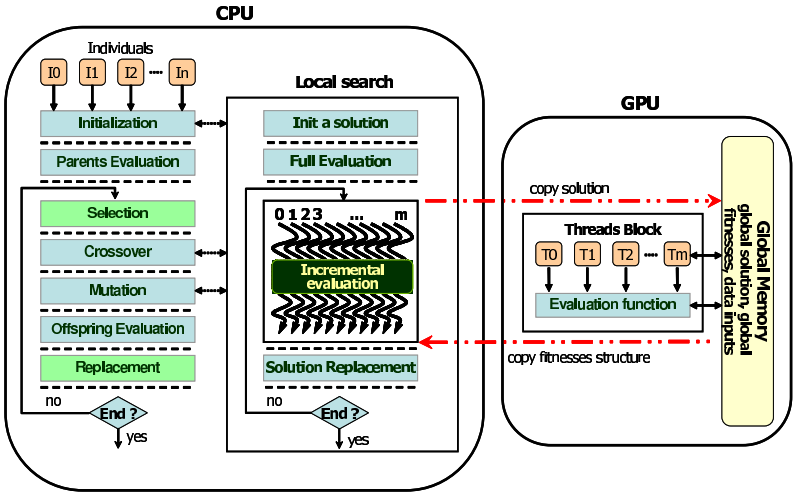

Figure 1. In the high-level layer, the CPU manages the whole hybrid genetic process. The generation and the evaluation of the LS neighborhood are performed in parallel on GPU.

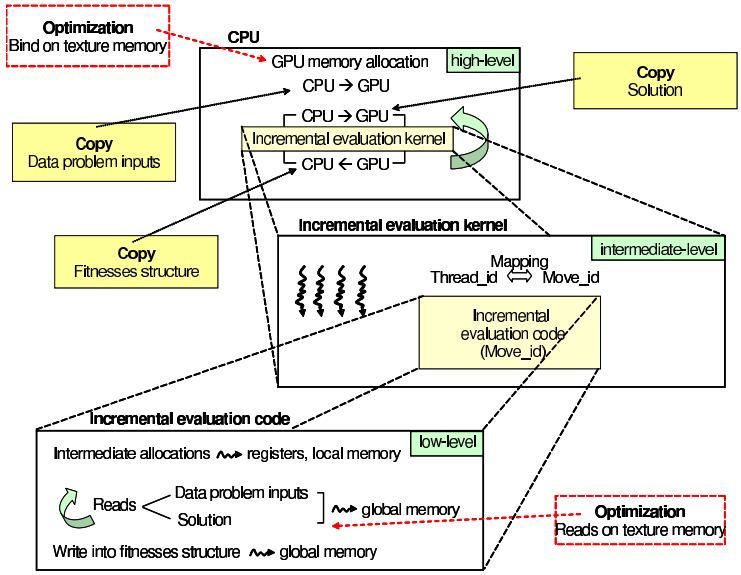

Figure 2. The three-level decomposition of the GPU hierarchy in accordance with the hybrid genetic process.

Table I
HYBRID GENETIC ALGORITHM FOR DIFFERENT QAP INSTANCES.

Instance	Best known value	Average value	Hits	CPU time	GPU time	Acceleration	GPUTexture time	Acceleration
tai30a	1818146	1818442_{823}	27/30	1h 15min	14min 25s	×5.2	8min 50s	×8.5
tai35a	2422002	2422437_{1650}	23/30	2h 24min	25 min 36s	×5.6	12min 56s	×11.1
tai40a	3139370	3146480_{2875}	18/30	3h 54min	39min	×5.9	18min 16s	×12.8
tai50a	4938796	4961204_{4126}	10/30	10h 2min	1h 37min	×6.2	45 min	×13.2
tai60a	7205962	7241224_{5845}	6/30	20h 17min	3h 9min	×6.4	1h 30min	×13.4
tai80a	13511780	13605896_{8833}	4/30	*66h*	9h 48min	×6.7	4h 45min	×13.8
tai100a	21052466	21190794_{14520}	2/30	*177h*	24h 29min	×7.2	12h 6min	×14.6

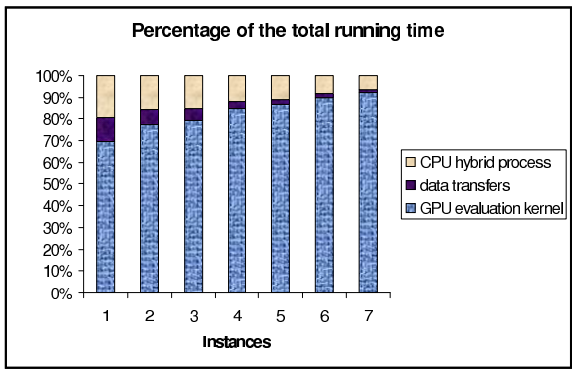

Figure 3. Average percentage of the time spent by each operation in the hybrid GA on GPU. The taillard instances are ordered according to their size.

quiet competitive. Indeed, Taillard instances larger than 30 are well-known for their difficulty and the proposed algorithm is able to find the best known value with a significant rate success for most instances.

C. Analysis of the Performances

It is well-known that CPU/GPU communication might be a major bottleneck in the performance of GPU applications. We propose to make an analysis of the percentage of the time spent by each major operation in our GPU-based implementation to evaluate the impact in terms of efficiency (see Fig. 3).

A first observation that can be made is about the data transfers between the CPU and the GPU. The time associated with the transfers keeps decreasing with the problem size increase. Indeed, from the first instance (tai30a), this time corresponds to 11% of the total running time and it reaches the value of 1% for the last instance (tai100a). As a consequence, the time dedicated to the data transfers in our scheme of parallelization is not significant in comparison with the LS evaluation process.

Another observation concerns the time spent by the generation and the evaluation of the neighborhood on GPU (evaluation kernel) which represents most of the total running time. For instance, for the fourth instance (tai50a), the time associated with the evaluation of the neighborhood corresponds to 85% of the total execution time. This time grows accordingly with the instance size (around 90% for tai60a, tai80a and tai100a). As a result, our proposed scheme of parallelization takes advantage of the GPU resource utilization.

III. Conclusion

We have proposed a new scheme of parallelization based on the generation and evaluation of the neighborhood on GPU. To the best of our knowledge, this approach has never been investigated for parallel hybrid GAs. Most of parallelization schemes are based only on 1) the parallel evaluation of the population for GAs [2] which is straightforward ; 2) the entire distribution of the LS process on GPU [4] which implies a lot of number of LS executions (threads) to cover the memory access latency. Indeed, the generation and evaluation of the neighborhood in parallel on GPU is not straightforward since it implies to find the association between a thread and a particular neighbor. Indeed, since a dynamic generation of the neighborhood is performed, no implicit memory structures is allocated and thread mappings must be handled.

The conclusion of the experiments indicates that the use of GPU provides an efficient way to deal with large neighborhoods. Indeed, since the LS neighborhood is based on a Hamming distance of 3 (two swaps), the proposed hybrid GA is unpractical in terms of single CPU computational resources for large instances such as tai80a or tai100a (more than 60h per run). So, implementing this algorithm on GPU has allowed to exploit parallelism in such neighborhood to improve the robustness/quality of provided solutions.

References

[1] E.-G. Talbi, *Metaheuristics: From design to implementation*. Wiley, 2009.

[2] O. Maitre, L. A. Baumes, N. Lachiche, A. Corma, and P. Collet, "Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea," in *GECCO*, F. Rothlauf, Ed. ACM, 2009, pp. 1403–1410.

[3] A. Misevicius, "A fast hybrid genetic algorithm for the quadratic assignment problem," in *GECCO*, M. Cattolico, Ed. ACM, 2006, pp. 1257–1264.

[4] W. Zhu, J. Curry, and A. Marquez, "Simd tabu search with graphics hardware acceleration on the quadratic assignment problem," *International Journal of Production Research*, 2008.