Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 7, July 2014, pg.268 – 274

RESEARCH ARTICLE

Solution Level Parallelization of Local Search Metaheuristic Algorithm on GPU

Mr. S. V. Ghorpade, Prof. Mrs. S. M. Kamalapur

PG Student, Department of Computer Engineering, KKWIEER, University of Pune, India

Department of Computer Engineering, KKWIEER, University of Pune, India

swapnilghorpade@gmail.com; snheal_kamalapur@yahoo.com

Abstract— Local search metaheuristic algorithms are proven & powerful combinatorial optimization strategies to tackle hard problems like traveling salesman problem. These algorithms explore & evaluate neighbors of a single solution. Time Consuming LSM algorithms can be improved by parallelizing exploration & evaluation of neighbors of a solution. GPU architecture is suitable for algorithms of single program multiple data parallelism. Implemented algorithm reduces time consuming memory transfers and improves computational time by efficient use of memory hierarchy.

Keywords— Combinatorial Optimization, GPU, Local Search Metaheuristics, Parallel Computing

I. INTRODUCTION

Local search metaheuristics algorithms are among the most powerful strategies for solving hard problem from areas like design, logistics [8], biology [4] etc. Local search algorithms handle a single solution iteratively improved by exploring and evaluating neighborhood of a solution. Fig. 1 shows a general model of local search metaheuristic algorithms.

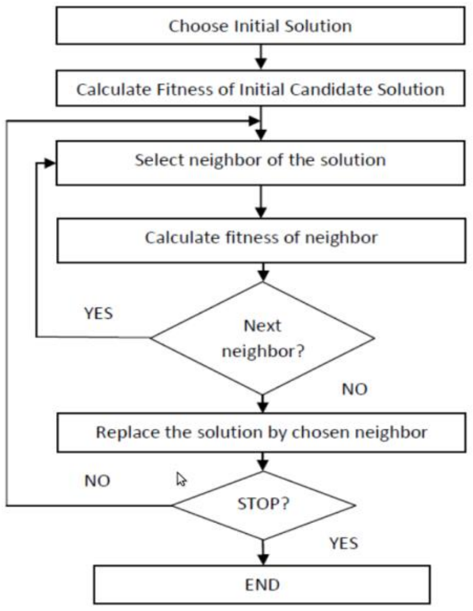
Choose Initial Solution
Calculate Fitness of Initial Candidate Solution
Select neighbor of the solution
Calculate fitness of neighbor
YES
Next neighbor?
NO
Replace the solution by chosen neighbor
NO
STOP?
YES
END

Fig. 1: General Model of LSM

Many NP hard problems from industry [4] and science [8] are successfully solved with such combinatorial optimization techniques.
General local search metaheuristic techniques are tabu Search [3], simulated annealing, guided local search, iterative local search and variable neighborhood search [9].
To implement local search, we need to encode our solution in suitable representation which affects performance. Binary encoding, vector of discrete values, permutation and vector of real values are four main encodings [5].
According to results and studies, Increase in neighborhood exploration improves quality of solution [10]. Although Local search metaheuristics (LSM) are effective techniques, problems are also becoming large and complex [11]. To solve large problems, we also need to equip with large computational power.
Initially, a local search metaheuristic algorithm starts with a randomly generated solution. After every iteration of the algorithm, the current solution is replaced by another solution selected from the neighboring candidates of a solution, and so on. An evaluation function calculates a fitness value to each solution as per the problem. Many strategies can be applied for the selection of a next solution: best improvement, first improvement, random selection, etc. Performance of local search is remarkably improved with parallel execution of exploration and execution process. Three major parallel models for local search are: solution level, iteration level and algorithmic level [12] [15] [11]. Solution level parallel model consists of parallelizing time intensive task of the algorithm. These kinds of models are problem dependant and are not always possible. Iteration level parallel model consists of parallel evaluation of all neighbors of a solution. It is a master-worker model which does not alter behavior of algorithm. Algorithmic-level parallel model consists of several self-configured algorithms in parallel. All instances may work in cooperation or independently [7] [9].

GPU architecture provides a very high computation power and very high memory bandwidth compared to traditional CPUs. GPUs have evolved into a mulithreaded, highly parallel and many core architecture due to demand for 3D high-definition graphics [16].
Luong V. et. al [1] and Rocki, K. et. al [2] successfully tested local search algorithms on GPU. In which, Evaluation of neighborhood is performed on GPU, and rest of local search algorithm work is carried out on CPU. To do this, lot of transfer of solutions from CPU to GPU and vice versa need to be done.
We have implemented a solution level parallel model for local search metaheuristic algorithms on GPU. We tested our approach on travelling salesman problem for varying cities datasets.
Section II presents literature review of local search metaheuristic algorithms. Section III describes mathematical model. Section IV presents proposed work. Section V presents system architecture. Section VI discusses data sets and result sets. Section VII summarizes the method.

II. GPU ARCHITECTURE

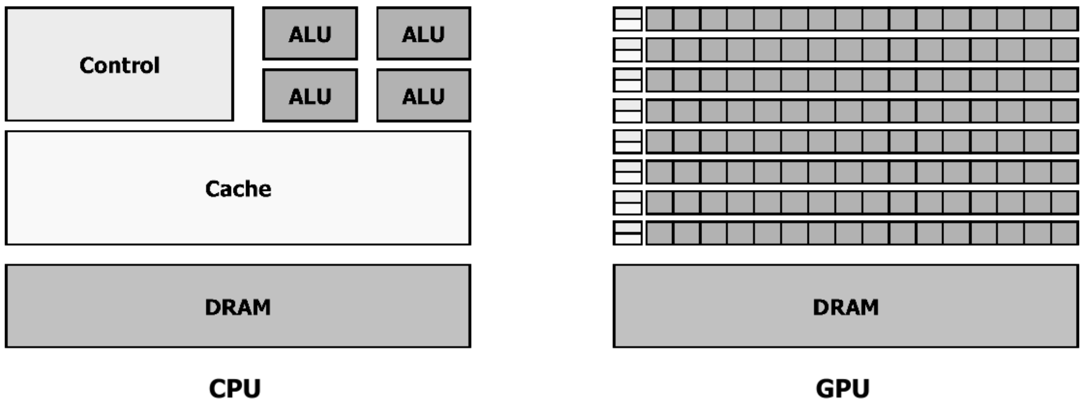

Fig. 2: CPU vs. GPU

A GPU has a large number of arithmetic units with a limited cache and few control units. This allows the GPU to calculate in a massive and parallel way the rendering of small and independent elements, while having a large flow of data processed. Since more transistors are devoted to data processing rather than data caching and flow control, GPU is specialized for compute-intensive and highly parallel computations. It is composed of streaming multiprocessors (SMs), each containing a certain number of streaming processors (SPs) i.e. processor cores. Each core executes a single thread instruction in a SIMD (single-instruction multiple-data) fashion, with the instruction unit distributing the current instruction to the cores [16] [17].

In GPU, multiple processors simultaneously execute the same program on different data. To achieve this, “kernel” concept is introduced. The kernel function is callable from CPU and executed on the GPU by several processors in parallel simultaneously. This kernel handling is dependent of the general-purpose language. CUDA (Compute Unified Device Architecture) is a parallel computing environment, which facilitates an application programming interface for NVIDIA architectures. GPU thread can be seen as an element of the data to be processed. CUDA threads are lightweight than CPU threads. Changing the context between two CUDA threads is not a costly operation. Threads are grouped within a structure called blocks. A kernel is executed by multiple equally threaded blocks. Blocks can be organized into a one or two-dimensional grid of thread blocks, and threads inside a block are grouped in a similar way. All the threads belonging to the same thread block will be assigned as a group to a single multiprocessor, while different thread blocks can be assigned to different multiprocessors. Thus, a unique id can be assigned for each thread to perform computation on different data [17] [18].

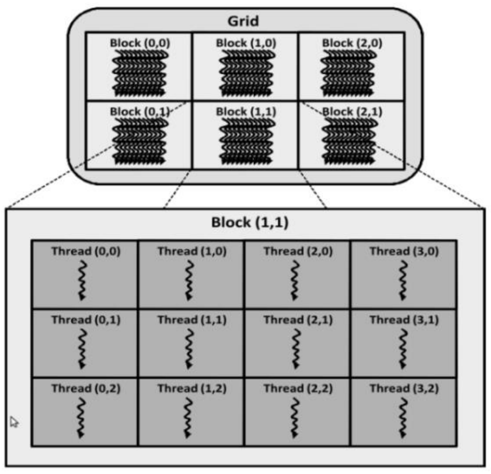

Fig. 3: GPU Threads Model

III. LITERATURE REVIEW

Parallel design and implementation of metaheuristics have been studied as well on different architectures. Some of them are using massively parallel processors [20], clusters of workstations [21] and shared memory or SMP machines [22], large-scale computational grids [19] .
Recently, Rocki, K., Suda, R. implemented 2-opt and 3-opt iterated local search metaheuristic algorithms on GPU to solve traveling salesman problem using standard set of libraries- TSPlib [2]. The core idea is to take a path which crosses over itself and reshuffle it so that it does not. Use of GPU greatly decreases the time needed to search best edges to be exchanged in a route.

The 2-opt algorithm basically removes two edges from the tour, and reconnects the two new sub-tours created. This is often referred to as a 2-opt move. There is only one way to reconnect the two sub-tours so that the tour remains valid. The steps are repeated only as long as the new tour is shorter. Figure 8 shows example of 2-opt step. The following step will be taken on following condition. If addition of distances of edges (B,F) and (G,D) is greater than addition of distances edges (B,D) and (G,F).

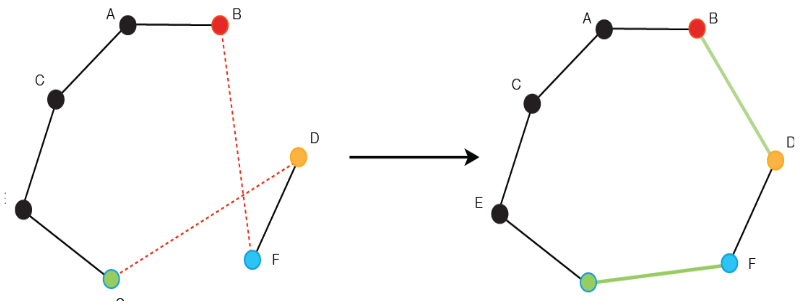

Fig. 4: 2-opt

Using GPU, the time needed to find the best edges to be swapped in a route is greatly decreases. Their results show that by using GPU algorithm, the time can be decreased approximately 3 to 26 times compared to parallel CPU code using 32 cores.

Overview of 2-opt and 3-opt GPU algorithm

1. Copy the tour and the coordinates to the GPU global memory (CPU)
2. Execute the kernel
3. Copy the tour and coordinates to the shared memory
4. Calculate swap effect of one pair
5. Find the best value and store it in the global memory
6. Read the result (CPU)

Results show that more than 90% of the time during iterated Local Search is spent on the 2-opt itself its increasing with the size of problem size. Distance between nodes need to be calculated to know effect of edges exchange. Calculation of distance based on nodes coordinates than using Look up table is proved to better way in terms of memory usage. First way is further optimized by placing nodes coordinate's data on on-chip shared memory which limits number of cities to 4800. Overall, GPU acceleration to 2-opt and 3-opt search compared to the 32 CPU cores acceleration ranged from 3 to 20 and 12-26 for the respectively.
Luong, Melab and Talbi [1] proposed very efficient approaches for design and implementation of LSM algorithms. Iteration-level parallel model is used and tested for the traveling salesman problem (TSP), quadratic assignment problem (QAP), permuted perceptron problem (PPP) and continuous weierstrass function.

Algorithm 1: Local Search Template Proposed by Luong, Melab and Talibi

```
1: Choose an initial solution
2: Evaluate the solution
3: Specific LSM initializations
4: Allocate problem inputs on GPU memory
5: Allocate a solution on GPU memory
6: Allocate a fitnesses structure on GPU memory
7: Allocate additional structures on GPU memory
```

```
8: Copy problem inputs on GPU memory
9: Copy the initial solution on GPU memory
10: Copy additional structures on GPU memory
11: repeat
12:     for each neighbor in parallel do
13:         Incremental evaluation of the candidate solution
14:         Insert the resulting fitness into the fitnesses structure
15:     end for
16:     Copy the fitnesses structure on CPU memory
17:     Specific LSM selection strategy on the fitnesses structure
18:     Specific LSM post-treatment
19:     Copy the chosen solution on GPU memory
20:     Copy additional structures on GPU memory
21: until a stopping criterion satisfied
```

Neighborhood generation on CPU and its evaluation on GPU is a straightforward approach in which for every iteration associated structure is copied from CPU to GPU. Neighborhood generation and its evaluation on GPU is more complex for thread mapping to neighbor but significantly reduces data transfers. For some LSMs like hill climbing parallel reduction techniques are applied to find proper minimum fitness structure which additionally optimizes data transfer.

IV. Proposed Solution

Solution level parallel model is suitable for reducing data transfers between CPU and GPU memories which is time ingesting task [20] [23]. Evaluation of neighbourhood is performed on GPU before every iteration. This needs to copy solution from CPU memory to GPU memory. The results obtained need to be copied back to CPU memory. Best solution is chosen using the copied results. Thus there are many memory transfers between CPU and GPU memory [25]. Proposed solution modifies algorithm which will reduce data transfers and all operations on GPU.
Increase of local search in neighbourhood of a solution also increases quality of solution. By parallelization of 2-opt step for a single tour to a single thread block instead of threads reduces memory usage per block. This strategy is helpful to scale up algorithm as per capacity of GPU for simultaneous blocks execution. Based on number of cities and shared memory availability, algorithm calculates best number of threads per block.
Algorithm 2 shows our proposed algorithm.

Algorithm 2: Proposed Algorithm

```
1: Choose an initial solution
2: Evaluate the solution
3: Allocate GPU memory for problem inputs
4: Allocate GPU memory for initial solution
5: Allocate GPU memory for generated neighbours from initial solution
6: Allocate GPU memory for storing results of neighborhood evaluation that is fitness structure
7: Copy problem inputs and initial solution on GPU memory
8: repeat
9:      Launch kernel to generate neighbors of chosen solution at calculated memory locations
10:     Launch second kernel for evaluation of all neighbors in parallel. Insert the result to fitness structure
11:     Launch third kernel of selection strategy over calculated fitness structure for selection of next solution
        to evaluate
12:     Set chosen solution as a next candidate for generation and evaluation of neighborhood
13: until a termination criterion satisfied
14: Copy best solution to CPU memory
```

Initial solution is chosen to evaluate on CPU. GPU memory is allocated for initial solution, problem inputs (for example, lookup table for TSP problem), and neighborhood. As results of every neighbor evaluation need to be stored for comparisons, fitness structure is allocated for storing results on GPU. Initial solution and problem inputs are copied on GPU memory. To generate neighborhood of candidate solution, first kernel is launched. Every neighbor is stored in a mapped memory location. Second kernel is launched to evaluate every neighbor in parallel by mapping a thread to it. Third kernel is launched to select best solution among all evaluated neighbors. Reduction techniques can be employed for this if minimum fitness needs to be computed. Chosen solution is configured a next candidate for evaluation. Advantage of this approach over iteration-level is reduction memory transfer operations and utilization of GPU computing power for useful work. This approach can be parallelized for more than one candidate solution in which availability of memory can put a limit on large neighbourhood LSM algorithms.

To validate our approach, proposed algorithm is implemented for travelling salesman problem (TSP) [6] [13] [14]. NVIDIA GPU GTX 680 graphics card using CUDA 5.5 platform architecture is used.

V. RESULTS

TSPlib is a benchmark library contains set of instances of real life travelling salesman problems [24]. We chose datasets of varying number of cities ranging from 127 to 24978. We tested our approach with varying number of blocks. As per the results, increase in number of blocks also increases quality of solution that is minimizes tour length in this case.

TABLE I: RESULTS

Dataset	No. of cities	No. of Blocks	Time	Cost of Tour
bier127	127	100	0.0139 s	119892
		10000	1.0561 s	119109
		20000	1.957 s	118607
ch130	130	100	0.0146 s	6321
		200	0.0290 s	6263
		400	0.0551 s	6258
		500	0.0600 s	6258
		10000	1.0689 s	6159
		20000	2.1223 s	6157
		50000	5.2541 s	6153
		100000	10.5240 s	6148
		200000	21.0184 s	6110
pr1002	1002	100	1.5762 s	278381
		200	3.2850 s	278621
		300	4.8314 s	276084
		2000	29.1702 s	274778
		10000	144.6441 s	274025
fnl4461	4461	2	77.0407 s	203096
		10	77.9637 s	202929
		20	80.57525 s	202587
		30	83.5190 s	202488
		100	139.6203 s	202347
usa13509	13509	2	2310.1165 s	22090071
		50	9321.2440 s	21999132
d18512	18512	2	5770.9347 s	719576
sw24978	24978	16	15071.5507 s	949792

VI. CONCLUSIONS

In this paper we have presented a solution level parallel model implementation for 2-opt local search of Travelling Salesman Problem. Parallel implementation of local search metaheuristics allows to improve the quality of solution by exploring into more neighbourhood of solution. GPU computing has been revealed to be a good way to provide such high performance computational power.

ACKNOWLEDGMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the kepler architecture (GTX 680) GPU used for this research.

REFERENCES

[1] The Van Luong; Melab, N.; Talbi, E.-G., "GPU Computing for Parallel Local Search Metaheuristic Algorithms," Computers, IEEE Transactions on , vol.62, no.1, pp.173,185, Jan. 2013

[2] Rocki, K.; Suda, R., "Accelerating 2-opt and 3-opt local search using GPU in the travelling salesman problem," High Performance Computing and Simulation, 2012 International Conference on , vol., no., pp.489,495, 2-6 July 2012

[3] E.D. Taillard, “Robust Taboo Search for the Quadratic Assignment Problem,” Parallel Computing, vol. 17, nos. 4/5, pp. 443-455, 1991.
[4] E. Lutton and J.L. Ve´hel, “Holder Functions and Deception of Genetic Algorithms,” IEEE Trans. Evolutionary Computation, vol. 2, no. 2, pp. 56-71, July 1998.
[5] E.-G. Talbi, Metaheuristics: From Design to Implementation. Wiley, 2009.
[6] Helsgaun, K.; An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, European Journal of Operational Research, 2000, vol 126, pages 106-130
[7]V. Boyer, D. El Baz, M. Elkihel, Solving knapsack problems on GPU, Computers & Operations Research, Volume 39, Issue 1, January 2012, Pages 42-47
[8] Lee M.,Jeon J., Bae J., Jang H. S., Parallel implementation of a financial application on a GPU. Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, 2009.
[9] E. H. L. Aarts and J. K. Lenstra, editors. Local Search in Combinatorial Optimization. John Wiley & Sons, Chichester,UK, 1997
[10] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation,6(5):443462, October 2002.
[11] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, and K. Skadron, “A Performance Study of General-Purpose Applications on Graphics Processors Using Cuda,” J. Parallel Distributed Computing, vol. 68, no. 10, pp. 1370-1380, 2008.
[12] T.-T. Wong and M.L. Wong, “Parallel Evolutionary Algorithms on Consumer-Level Graphics Processing Unit,” Proc. Parallel Evolutionary Computations, pp. 133-155, 2006.
[13] M. Dorigo and L.M. Gambardella, “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem,” IEEE Trans. Evolutionary Computation, vol. 1, no. 1, pp. 53-66, Apr. 1997.
[14]E. Lutton and J.L. Ve´hel, “Holder Functions and Deception of Genetic Algorithms,” IEEE Trans. Evolutionary Computation, vol. 2, no. 2, pp. 56-71, July 1998.
[15] T. James, C. Rego, and F. Glover, “A Cooperative Parallel Tabu Search Algorithm for the Quadratic Assignment Problem,” European J. Operational Research, vol. 195, pp. 810-826, 2009.
[16] NVIDIA, CUDA Programming Guide Version 5.0, 2013.
[17] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, “GPU Computing”, Vol. 96, IEEE ,May 2008 ,Proceedings of the IEEE
[18] John Nickolls, William J. Dally, “GPU Computing Era”, IEEE MICRO, MARCH/APRIL 2010
[19] Enrique Alba, Francisco Luna, Antonio J. Nebro, and Jos´e M. Troya. Parallel heterogeneous genetic algorithms for continuous optimization. Parallel Computing, 30(5-6):699–719, 2004.
[20] J. Chakrapani and J. Skorin-Kapov. Massively Parallel Tabu Search for the Quadratic Assignment Problem. Annals of Operations Research, 41:327–341, 1993.
[21] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. Annals of Operations Research, 63:277–299, 1995.
[22] T. James, C. Rego, and F. Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. European Journal of Operational Research, 195:810–826, 2009.
[23] Alexandru-Adrian Tantar, Nouredine Melab, and El-Ghazali Talbi. A comparative study of parallel metaheuristics for protein structure prediction on the computational grid. In IPDPS, pages 1–10. IEEE, 2007.
[24] Reinelt, G.: TSPLIB - A Traveling Salesman Problem Library. ORSA Journal on Computing, Vol. 3, No. 4, pp. 376- 384. Fall 1991.
[25] Phuong Hoai Ha; Tsigas, P.; Anshus, O.J., "The Synchronization Power of Coalesced Memory Accesses,"Parallel and Distributed Systems, IEEE Transactions on ,vol.21, no.7, pp.939,953, July 2010