Journal of Computer Science 8 (5): 705-710, 2012
ISSN 1549-3636
© 2012 Science Publications

Parallel Memetic Algorithm for VLSI Circuit Partitioning Problem using Graphical Processing Units

[1]Subbaraj, P. and [2]P. Sivakumar
[1]Sri Nandhanam College of Engineering and Technology, Tirupattur-635601, Vellore District, TamilNadu, India
[2]Department of Electronics and Communication Engineering, Arulmigu Kalasalingam College of Engineering, Anand Nagar, Krishnankoil-626126, Srivilliputhur, Virudunagar District, Tamilnadu, India

Abstract: Problem statement: Memetic Algorithm (MA) is a form of population-based hybrid Genetic Algorithm (GA) coupled with an individual learning procedure capable of performing local refinements. Here we used genetic algorithm to explore the search space and simulated annealing as a local search method to exploit the information in the search region for the optimization of VLSI netlist bi-Partitioning problem. However, they may execute for a long time, because several fitness evaluations must be performed. A promising approach to overcome this limitation is to parallelize this algorithms. General Purpose computing over Graphical Processing Units (GPGPUs) is a huge shift of paradigm in parallel computing that promises a dramatic increase in performance. **Approach:** In this study, we propose to implement a parallel MA using graphics cards. Graphics Processor Units (GPUs) have emerged as powerful parallel processors in recent years. Using of Graphics Processing Units (GPUs) equipped computers; it is possible to accelerate the evaluation of individuals in Genetic Programming. Program compilation, fitness case data and fitness execution are spread over the cores of GPU, allowing for the efficient processing of very large datasets. **Results:** We perform experiments to compare our parallel MA with a Sequential MA and demonstrate that the former is much more effective than the latter. Our results, implemented on a NVIDIA GeForce GTX 9400 GPU card. **Conclusion:** Its indicates that our approach is on average 5×faster when compared to a CPU based implementation. With the Tesla C1060 GPU server, our approach would be potentially 10×faster. The correctness of the GPU based MA has been verified by comparing its result with a CPU based MA.

Key words: Genetic Algorithm (GA), Graphics Processing Units (GPUs), Memetic Algorithm (MA), netlist partitioning, genetic programming, graphics cards, local search, physical design

INTRODUCTION

Physical design of VLSI circuits is the process of mapping structural representations of circuits into layout representation. Due to the complexity of the physical design phase it is usually broken down to sub problems like, partitioning, placement and routing which are then solved one after the other. This study is concerned with the circuit partitioning problem. Circuit net list partitioning is an important step in VLSI physical design. This involves the breakup of a circuit into smaller parts for ease of design, layout and testability. The main objective of circuit partitioning is minimization of number of interconnections between the partitions (Sait and Youssef, 1999; Sarabian and Lee, 2010; Go *et al*., 2010). Partitioning is a technique to divide a circuit or system into a collection of smaller parts (components). It is on the one hand a design task to break a large system into pieces to be implemented on separate interacting components and on the other hand it serves as an algorithmic method to solve difficult and complex combinatorial optimization problems as in logic or layout synthesis. Partitioning has been an active area of research for at least a quarter of a century. The main reason that partitioning has become a central and sometimes critical design task today is the enormous increase of system complexity in the past and the expected further advances of nano-electronics system design and fabrication. Soaring system complexities result from a combination of

Corresponding Author: Subbaraj, P., Sri Nandhanam College of Engineering and Technology, Tirupattur-635601, Vellore District, TamilNadu, India

reasons: Widely accepted powerful high-level synthesis tools allow the designers to automatically generate huge systems. By just changing a few lines of code in a functional specification the size of the resulting structural description (netlist) of a system can increase dramatically. Synthesis and simulation tools often cannot cope with the complexity of the entire system under development and designers want to concentrate on critical parts of a system to speed-up the design cycle. Thus, the present state of design technology often requires a partitioning of the system with fast and effective optimization. Moreover, fabrication technology makes increasingly smaller feature sizes and augmented die dimensions possible, thus allowing, a circuit to accommodate several million transistors. However, circuits are restricted in size and in the number of external connections. Thus, fabrication technology requires the partitioning of a system into components.

MATERIALS AND METHODS

This study addresses the problem of VLSI netlist partitioning with the objective of optimizing cutset while considering the balance constraint (same as area constraint as unit area is assumed for every gate). Formally, the problem can be stated as follows: Given a set of modules $V = \{v_1, v_2 \ldots v_n\}$, the purpose of partitioning is to assign the modules to a specified number of clusters k (two in our case) satisfying prescribed properties. In general, a circuit can have multi-pin connections (nets) apart from two-pin and therefore it is better to represent it by a hypergraph. A hypergraph H (V, E) is defined where, V is a set of nodes and E is a set of hyper edges. Node $v_i \in V$ corresponds to an element (e.g., a gate) in the circuit and hyper edge $e_i \in E$ corresponds to a net in the circuit. Given a hypergraph H (V, E) with $E = \{e_1, e_2 \ldots e_m\}$ being the set of signal nets, each net is a subset of V containing the modules connecting the net. It is assumed that for each hyperedge $e \in E$, $|e| \geq 2$ (it connects at least two nodes). Our task is to divide V into 2 subsets (clusters) V0 and V1 in such a way that the objectives are optimized, subject to some constraints.

Cutsize: The set of hyper edges cut by a cluster C is given by $E(C) = \{e \in E: 0 < |e \cap C| < |e|\}$ i.e., $e \in E(C)$ if at least one, but not all, of the pins of e are in C. The set of nets cut by a partitioning solution p^K can be expressed as $E(p^k) = U_{i=1}^{k} E(c_i)$ or equivalently $E(p^k) = \{e \in E | \Xi\ u, v \in e;\ h \neq l \text{ with } u \in C_h \text{ and } v \in C_l\}$. We say that $|E(p^k)|$ is the cutsize of p^k. The cost function f can be written as follows Eq. 1:

$$f = \sum_{e \in \psi} w(e) \qquad (1)$$

where, $\psi \in E$ denotes the set of off-chip edges, i.e., nets cut. The weight w(e) on the edge e represents the cost of wiring the corresponding connection as an external wire. If all weights equal one, the cost function becomes simpler Eq. 2:

$$f = |\psi| \qquad (2)$$

where, $|\psi|$ denotes the cardinality of the set ψ.

Area or balance constraint: If we assume that the area of all cells is identical, then the problem reduces to balancing the two partitions in terms of the number of cells. The balance constraint is given below Eq. 3:

$$\frac{|\beta_1 - \beta_2|}{\phi} \leq \alpha \qquad (3)$$

where, β_i is the number of cells in partition i , φ is the total number of cells in the circuit, α is the tolerance which is equal zero in case of a perfect balance. When balance is used as cost, it will be $|\beta_1 - \beta_2|$.

Numerous partitioning algorithms have been developed by researchers over the years. Bui and Moon (1998) and Alpert *et al*. (1996) suggested a hybrid genetic approach for circuit partitioning. Yodtean and Chantngarm (2004) suggested a hybrid algorithm which combines Genetic and Simulated Annealing techniques, to improve the performance in circuit partitioning while using less resources. Coe *et al*. (2004) investigated the feasibility of using Reconfigurable Computing platforms to improve the performance of VLSI circuit partitioning problem.

Due to the fact that Memetic Algorithms (MA) aimed at drawing the attention from two communities of researchers with different agendas, aiming at hybridizations of their methods, this met heuristic had to suffer tough initial times. Today they are becoming increasingly popular due to their impressive success record and the high sophistication of the hybridizations proposed. Although Memetic Algorithms (MA) are effective in solving many practical problems in science, engineering and business domains, they may execute for a long time to find solutions for some huge problems, because several fitness evaluations must be performed.

A promising approach to overcome this limitation is to parallelize these algorithms using parallel, distributed and networked computers (Yussof *et al*., 2011). However, these computers are relatively more difficult to use, manage and maintain. Moreover, some people may not have access to this kind of computers. Consequently, we propose to implement a parallel MA using graphics cards which are available in all-pervading personal computers. Harding and Banzhaf (2007) demonstrates the benefit of using the graphics processor to parallelise the genetic evaluations. Robilliard *et al*. (2008) suggested about parallelizing both Genetic programs and training data, on GPU.

Graphics processing unit: Three major factors make the development of graphics hardware based on commodity PCs truly outstanding in recent years. First, the computational power of Graphics Processing Units (GPUs) for commodity PC hardware has grown much faster than for CPUs. Second, the high performance is available at a very good cost/performance ratio. Finally, within the last 4-5 years, GPUs have become programmable by high level languages. From an abstract point of view, the GPU is a parallel streaming processor, particularly suitable for the fast processing of large arrays. Thus, many researchers have started utilizing graphics processors to enhance the performance of their specific, in many cases, non-graphics applications and simulations. The special field of "General-Purpose computation on GPU (GPGPU)" has evolved offers a survey of this emerging research area. Although performance gains depend strongly on the application, one can say that speedup factors around 5 against algorithms on the CPU are commonly reported.

Graphics Processing Units (GPUs) are fast, highly parallel processor units. In addition to processing 3D graphics, modern GPUs can be programmed for more general purpose computation. The GPU consists of a large number of 'shader processors' and conceptually operates as a Single Instruction Multiple Data (SIMD) or Multiple Instruction Multiple Data (MIMD) stream processor. A modern GPU can have several hundred of these stream processors, which combined with their relatively low cost, makes them an attractive platform for scientific computing.

Graphics processors are specialized stream processors used to render graphics. Typically, the GPU is able to perform graphics manipulations much faster than a general purpose CPU, as the processor is specifically designed to handle certain primitive operations. Internally, the GPU contains a number of small processors that are used to perform calculations on 3D vertex information and on textures. These processors operate in parallel with each other and study on different parts of the problem. First the vertex processors calculate the 3D view and then the shader processors paint this model before it is displayed. Programming the GPU is typically done through a virtual machine interface such as OpenGL or DirectX which provide a common interface to the diverse GPUs available thus making development easy. However, DirectX and OpenGL are optimized for graphics processing, hence other APIs are required to use the GPU as a general purpose device. Depending on the GPU, the number of instructions may be limited. In order to use more than this number of operations, a program needs to be broken down into suitably sized units, which may impact performance. Newer GPUs support unlimited instructions, but some older cards support as few as 64 instructions. GPUs typically use floating point arithmetic, the precision of which is often controllable as less precise representations are faster to compute with. Again, the maximum precision is manufacturer specific, but recent cards provide up to 128-bit precision.

The rapid increase in the number and diversity of scientific communities exploring the computational power of GPUs for their data intensive algorithms has had a key contribution in encouraging GPU manufacturers to design more powerful, easily programmable and flexible GPUs. In addition, the development of open-source programming tools and languages for interfacing with the GPU platforms has further fueled the growth of general purpose GPU (GPGPU) applications. Further, GPU architectures have been continuously evolving towards higher performance, larger memory sizes, larger memory bandwidths and relatively lower costs. This high computing power mainly arises from a fully pipelined and highly parallel architecture, with extremely high memory bandwidths.

The NVIDIA® Tesla™ C1060 computing processor enables the transition to energy efficient parallel computing power by bringing the performance of a small cluster to a study station. With 240 processor cores and a standard C compiler that simplifies application development, Tesla scales to solve the world's most important computing challenges more quickly and accurately.

The GeForce 9400 GTX architecture has 16 stream processors and access to 512Mb of RAM. The theoretical performance of this card is 44 Gflops. Although currently the GPUs in this setup are low end, we are confident that the approach detailed here will also be applicable to high-end and future devices.

Memetic algorithm for circuit partitioning: The Genetic Algorithm starts with a set of initial solutions called population that is generated randomly. When generating the random initial solution it is preferred that it is within the bounds of the balance constraint. PMX Crossover is used as a genetic operator. It offers better performance than most other crossover techniques. Basically, parent 1 donates a swath genetic material and the corresponding swath from the other parent is sprinkled about in the child. Once that is done, the remaining alleles are copied direct from parent 2. Depending on the mutation rate, a few nodes are selected randomly from the chromosome and replaced in the other possible nodes in the chromosome. This mutation process would permit population diversity to be maintained in later stages of the GA. Mutation also helps the GA to surmount any local optimum. Individuals for the next population are selected based on the elitist-random selection (ernd). $N_p/2$ (N_p is the population size) best chromosomes are selected and the remaining $N_p/2$ are selected randomly.

Initially, the Memetic Algorithm (MA) randomly generates a population of individuals using the technique described above. Then, the MA starts evolving the population generation by generation. In each generation, the MA uses the genetic operators probabilistically on the individuals in the population to create new promising search points (admissible partitioning) and uses the Simulated Annealing (SA) as a local search method to optimize them if the fitness of the admissible partitioning is greater than or equal to existing solution provided as input to the local search.

Simulated Annealing (SA) is a general iterative improvement algorithm that can be used for many different purposes. In partitioning, SA starts with a random partition from the GA. A new state is computed by selecting a gate at random from each of the two subsets and swapping them. As before, the swap remains tentative, until the quality of the new partitioning is computed. The number of nets cut is the measure of goodness. If the new state is better than the old state, it is accepted and the swap is made permanent. If the new state is worse than the old state, it might be accepted and it might not. In most cases the acceptance function is computed using the following function, $e\frac{-\delta s}{T}$, where δs is the change in the quality and T is the current temperature. For bad moves this function will produce a value between 0 and 1. A random number between 0 and 1 is generated and if the quality measure is larger than the generated random number, the bad move is accepted. Recall that in partitioning, negative values of δs are good and positive values are bad.

Memetic algorithm on GPU: We have implemented our Memetic algorithm in MATLAB® program. The GPU is especially well-suited to address problems that can be expressed as data-parallel computations; one of the most important things we can do to prepare for GPU computing with MATLAB is to understand those segments of our target application where data parallel computations take place. This is our first indication of place in our code that is prospects for GPU computing.

Next, profiling our application to identify the segments of our code that represent the most time consuming regions will provide further indication of those segments of our code that could benefit from GPU computing. The MATLAB Profiler tool helps tremendously in determining where best to focus your energy when moving code to the GPU. Looking at the results of the profiler, a user can determine where the program is spending most of its time and focus transformation time to the area of code to get the biggest return.

GPUmat, developed by the GP-You Group, allows Mat lab code to benefit from the compute power of modern GPUs. It is built on top of NVIDIA CUDA. The acceleration is transparent to the user, only the declaration of variables needs to be changed using new GPU-specific keywords. Algorithms need not be changed. A wide range of standard Mat lab functions have been implemented. GPUmat is available as freeware for Windows and Linux from the GP-You download page.

GPUmat uses a technology developed by NVIDIA called CUDA SDK which allows programming the GPU for general purpose applications. The GPUmat core is based on CUDA libraries, such as CUFFT and CUBLAS and many other functions developed and optimized by the GP-you Group for the GPU architecture.

RESULTS AND DISCUSSION

We experimentally evaluated the quality of the bisections produced by our GPU based parallel Memetic algorithm on a large number of hyper graphs that are part of the widely used ISCAS circuit partitioning benchmarks suite. All experiments were carried out on Pentium Quad core 2 processor 2.6 GHz withNVIDIA Tesla C1060 computing processor and GeForce GTX 9400 GPU display card, with 8GB main memory and 512MB GPU memory.

Table 1: Performance of MA for VLSI Circuit Partitioning problem

Benchmark circuit	Number of cells	Number of nets	MA on CPU Cut	MA on CPU T(S)	MA on GTX 9400GPU card Cut	MA on GTX 9400GPU card T(S)	MA on Tesla C1060GPU processor Cut	MA on Tesla C1060GPU processor T(S)	Speed up GTX 9400	Speed up Tesla C1060
S298	136	130	19	123	11	30	10	20	4.100	6.150
S386	172	165	36	163	28	50	26	40	3.260	4.075
S641	433	410	45	1868	15	300	13	200	6.227	9.340
S832	310	291	55	1055	39	500	37	400	2.110	2.638
S953	440	417	96	618	45	200	43	150	3.090	4.120
S1196	561	547	123	375	75	250	74	200	1.500	1.875
S1238	540	526	127	397	79	150	77	104	2.647	3.817
S1488	667	648	104	1238	80	454	78	406	2.727	3.049
S1494	661	642	120	1345	75	500	74	410	2.690	3.280
S2081	122	121	50	354	13	150	12	104	2.360	3.404
S3330	1962	1888	55	756	46	350	44	307	2.160	2.463
S5378	2994	2944	171	5201	151	854	140	590	6.090	8.815
S9234	5845	5822	231	9654	191	1723	180	1500	5.603	6.436
S13207	8652	8530	340	9789	311	1874	300	1604	5.224	6.103
S15850	10384	10296	421	8534	411	985	390	685	8.664	12.460
Avg.									3.897	5.202

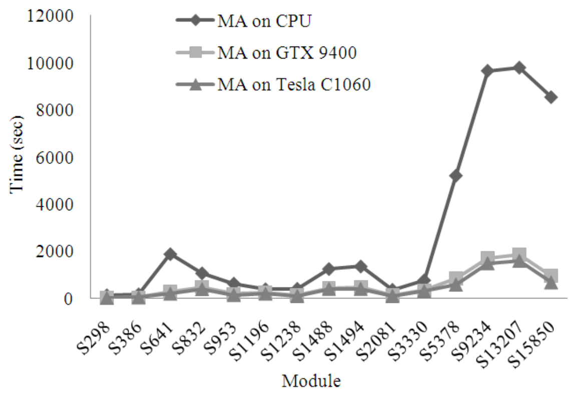

Fig. 1: Execution time of the GPU and CPU approaches

The GPUstart command is used to start GPUmat. The system might have more than one GPU installed. By default GPUstart selects the first available GPU device. The command GPUinfo prints information about installed GPUs. GPU-based implementation was compared with software implementation running on single CPU.

The following parameters were used in the experiments: For the Memetic Algorithm, the population size was set to 10, the probability for crossover is 0.95 and the probability for mutation is 0.05 for all test problems as it was the best configuration found empirically for the Genetic Algorithm. Table1 shows the statistics for the experiment.

The Figure 1 shows the comparison of time required to execute the modules on CPU platform and GPU platforms. The speedup obtained is on average of 3.89 on GTX 9400 GPU card. By using the NVIDIA Tesla C1060, the available global memory increases by 8GB. The speedup obtained in this case is on average of 5.202. Note that the commercial tool can be run on several CPUs using a distributed option. If each of these CPUs had a 9400 GTX GPU on board, then the GPU approach could also exploit a distributed option and the above speedup numbers would be increased.

CONCLUSION

In this research, we have implemented a parallel MA on consumer-level graphics cards and proposed indirect indexing and many optimization skills to achieve maximal efficiency. The parallel MA is a hybrid of master-slave and fine-grained models. Competition and selection are performed by CPU (i.e., the master) while fitness evaluation, mutation and reproduction are performed by GPU which is essentially a massively parallel machine with shared memory. Unlike other fine-grained parallel computers such as Maspar, GPU allows processors to communicate with any other processors directly, thus more flexible fine-grained EAs can be implemented on GPU. We have done experiments to compare our parallel MP on GPU and a Sequential MA on CPU. It is found that the speed-up factor of our parallel MA ranges from 1.5-8.6 while using GTX 9400 GPU card and 1.875-12.46 while using Tesla C1060 GPU processor. A couple of other important factors can help to get the best performance out of our transformation to GPUs such as avoiding of excessive memory transfer , inherent parallelism and computation dependency between CPU and GPU. The first time GPUs see a new piece of code from us, it spends some time analyzing it and compiling various instruction sequences for faster lookup on subsequent runs. That first run will be a little slower, but for long-running computations (several minutes) there won't be any noticeable lag. This is often referred to as "warm up".

There are still several constrains while using GPUmat. The performance of our method will be seriously limited because of the bottleneck GPUmat functions. For future study, we plan to implement the same study using either openGL or CUDA language. It will give better performance while compare with GPUmat functions.

REFERENCES

Alpert, C.J.L.W. Hagenand A.B.Kahng, 1996. A hybrid multilevel/genetic approach for circuit partitioning. Proceeding of the IEEE Asia Pacific Conference on Circuits and Systems, Nov. 18-21, IEEE XPlore, Seoul, South Korea, pp: 298-301. DOI: 10.1109/APCAS.1996.569275

Bui, T.N. and B.R. Moon, 1998. GRCA: A hybrid genetic algorithm for circuit ratio-cut partitioning. IEEE Trans. Comp.-Aided Design Integrated Cir. Sys., 17: 193-204. DOI: 10.1109/43.700718

Coe, S., S. Areibi and M. Moussa, 2004. A Genetic local search hybrid architecture for VLSI circuit partitioning. Proceedings of the 16th International Conference on Microelectronics, Dec. 6-8, IEEE XPloor, Canada, pp: 253-256. DOI: 10.1109/ICM.2004.1434259

Go, T.F., D.A. Wahab, M.N.A. Rahman and R. Ramli, 2010. A design framework for end-of-life vehicles recovery: Optimization of disassembly sequence using genetic algorithms. Am. J. Environ. Sci., 6: 350-356. DOI: 10.3844/ajessp.2010.350.356

Harding, S. and W. Banzhaf, 2007. Fast genetic programming on GPUs. Lec. Notes Comput. Sci., 4445: 90-101. DOI: 10.1007/978-3-540-71605-1_9

Robilliard, D., V. Marion-Poty and C. Fonlupt, 2008. Population parallel GP on the G80 GPU. Lec. Notes Comput. Sci., 4971: 98-109. DOI: 10.1007/978-3-540-78671-9_9

Sarabian, M. and L.V. Lee, 2010. A modified partially mapped multicrossover genetic algorithm for two-dimensional bin packing problem. J. Math. Stat., 6: 157-162. DOI: 10.3844/jmssp.2010.157.162

Sait, S.M. and H. Youssef, 1999. VLSI Physical Design Automation: Theory and Practice. 1st Edn.World Scientific, Singapore Europe. pp: 1-460.

Yodtean, A. and P. Chantngarm, 2004. Hybrid algorithm for bisection circuit partitioning. TENCON. IEEE Region 10 Conf., 4: 324-327. DOI: 10.1109/TENCON.2004.1414935

Yussof, S., R.A. Razali and O.H. See, 2011. An investigation of using parallel genetic algorithm for solving the shortest path routing problem. J. Comput. Sci., 7: 206-215.