Parallelizing the Coarsening Phase of Hyper-Edge Partitioning on the GPU Platform

Atefeh Taheri, Ali Jahanian, and Behin Molaie

***Abstract*—Significant portion of digital design flow runtime is related to the physical design stages. Partitioning is a critical stage of physical design and its quality and runtime has considerable impact on physical design efficiency. In this paper, a new parallel partitioning algorithm is proposed and it is suitable for GPU system. In the proposed algorithm, coarsening phase of the partitioning is accelerated by parallelizing on GPU. Experimental results show that runtime can be improved up to 7x for attempted circuit with negligible quality degradation.**

***Index Terms*—GPU programming, multilevel partitioning, parallel algorithms, physical design.**

I. Introduction

Integrated circuits complexity has been grown in last decades with exponential rate and it is predicted that this flow will be continued in future years [1], [2]. This exponential growth of modern circuits is an important challenge for computer-aided design tools because their runtime will be increased dramatically coping with this complexity growth [3], [4]. On the other hand, the gap between complexity growth and productivity of CAD tools (i.e. productivity gap) will be increased for new ultra-large systems such as system-on-chips and multi-core systems.

Considerable portion of total design time is related to physical design stage. Physical design consists of some important steps such as partitioning, floor-planning, placement and routing. In Partitioning phase, design graph is partitioned to some sub-graphs to facilitate rest of algorithms. Floor-planning find location and shape of large blocks. Exact location and orientation of the standard cells are fixed in placement step and finally, the geometry of interconnects will be determined in routing stage. A bundle of physical design algorithms are NP-complete and NP-hard [5], [6]. Graph partitioning is a critical sub-stage of physical design. Quality and execution time of partitioning stage has considerable impact on total quality of design and total execution time of physical design flow. Therefore, we focused on parallelizing the graph partitioning stage to improve its runtime regarding to the partitioning quality.

Many contributions are reported on parallelizing the CAD algorithms on multi-processor systems in the last years of 90's decade. Most of the proposed algorithms were used in multi-processor and multi-computer systems that do not equipped to efficient shared memories and communication between processors, and inter-thread communications could be done only with message passing protocol. Another drawback of the previous algorithms is that they require expensive hardware resources and do not efficiently executed on ordinary personal computers.

Based on the previous researches which will be introduced in Section II, more than half of the partitioning algorithm runtime is spent in coarsening phase. Therefore, we focused on accelerating this phase of the algorithm in this paper. In this paper, a GPU-optimized parallel algorithm is proposed to reduce execution time of the coarsening phase of the hyper-graph partitioning without considerable quality degradation. Our algorithm is a combination of sequential and parallel phases. We analyzed the sequential algorithm and parallelized more time consuming section over the underlying GPU architecture. The important point is that the parallelization should have as less as possible negative impact on the quality of partitioning. In the proposed algorithm, circuit netlist database is divided into n sections that are executed as parallel threads on GPU platform. Each partition will be coarsened in separated environment and the result sets will merge together on CPU.

This paper organized as follows. Section III describes the concept and existing algorithms for Multi-Level Hyper-Graph Partitioning and the coarsening phase is illustrated in Section IV. The article continues to describe the GPU programming in Section V and Section VI describes the proposed coarsening algorithm. Section VII describes the experimental results and finally, Section VIII concludes the paper.

II. Literature Review

In last few years computing power of the main processing units has not grown in comparison with growth of cells count in the digital circuits. So the run time of the algorithms has increased. On the other hand, Graphics Processing Units revolutionized the general purpose algorithms performance with their high computing power and low cost. These days many of the software companies are trying to improve their software performance using GPUs. Our idea was to create a parallel GPU based algorithm to run the partitioning phase of the design flow on a high performance GPU to speed up the design process.

Actually, in these days, there are many [7] applications in scientific computing for hypergraph partitioning. Foad Lotfifar and Matthew Johnson presented a new partitioning method for hypergraphs. They innovated a sequential

Manuscript received September 5, 2016; revised December 12, 2016.

Atefe Taheri was with the Electronic and Computer Engineering Department, Shahid Beheshti University, Tehran, Iran (e-mail: atefetaheri1439@gmail.com).

Ali Jahanian is with Electronic and Computer Engineering Department, Shahid Beheshti University, Tehran, Iran (e-mail: jahanian@sbu.ac.ir).

Behin Molaie is with the Computer Engineering Department, Sharif University of Technology, Tehran, Iran (e-mail: molaie@ce.sharif.edu).

DOI: 10.7763/IJCTE.2017.V9.1146

multi-level hypergraph partitioning algorithm. This algorithm uses a technique of rough set clustering to categorize the vertices of the hypergraph. They have paid attention so their algorithm doesn't make a greedy decision. In fact, this algorithm makes a trade-off between local decision and global decision. Results show that the algorithm makes better partitioning quality.

Hairong Liu [8] and his colleagues present a new partitioning framework. This framework is based on divide-and-conquer. Their new partitioning framework is called dense subgraph partitioning (DSP). DSP has properties such as revealing all meaningful clusters and etc. Moreover, to presenting a new framework they established a relationship with the densest k-subgraph problem (DkS). The results show that this approach is suitable for parallel processing because it is time-efficient and memory-friendly.

Many contributions are addressed in last few years to boost the performance of CAD problems using high performance GPUs. Authors of [9] have developed a parallel algorithm called mPL to fit on GPU so they can speed up the time they need to solve placement problems. mPL is an analytical global placement method and it can find a reasonable solution in a little time.

In [10] parallelizing the wire-length estimation on GPU with CUDA structure is presented. Authors of [10] created a parallel algorithm for calculating the wire-length of a solution on GPU and they got 160x speed up over a serial CPU algorithm.

Karypis and LaSalle implements the partitioning a graph on multi core systems using OpenMP and MPI [11]. They developed an algorithm to run on multithreaded CPU environment and they get some good result on speed up and memory usage and partitioning quality.

Authors of [12] proposed a parallel pathfinder global routing algorithm that is the most used FPGA routing algorithm. They introduce a parallel method to run on multi-core systems mainly to improve the runtime of the routing phase. They show with their experimental results that the runtime of the routing phase. They show with their experimental results that the run time can be reduces by 47.8% and 70.9% with dual and quad core systems.

In [6], a parallel Simulated Annealing algorithm on multi-core systems is proposed. Simulated Annealing is a known method to solve the optimization problems since 1953. The algorithm is used for complex and nonlinear combinatorial optimization problems. The algorithm searches the search space and finds a near optimal solution. The algorithm is taken a long time to finish if the search space is large. They show by their experimental results that they can improve the run time to 32% on average with considering the quality of the solution.

Caldwell *et al*. [13] introduced a new algorithm on multilevel partitioning. In their approach they introduce a technique of move-based hypergraph partitioning heuristic and they evaluated the performance of these heuristic in the context of VLSI design their first result was software architecture consist of 7 different reusable components. The formula allows a flexible, efficient and accurate assessment of the practical implications. Their second result was an assessment of the modern context for hypergraph partitioning research for VLSI design applications.

The biggest limitation of the algorithms mentioned before is the runtime to find a good answer. Another problem is that existing methods need a high resource environment to run.

III. Multi-level Hypergraph Partitioning

Hypergraph partitioning is an important problem in many engineering and optimization applications such as VLSI design flow. The main problem is to partition the nodes of a hypergraph into k different sets such that cut size of partitioning is minimized and balance criteria is not violated. In other words, this problem is an optimization problem whose goal is to minimize the cut size and its condition is the balancing rule. Hypergraphs are generalization of graphs that each edge can be a hyper edge. A hyper edge is an edge that connects a set of vertices (two or more vertices).

Hyper edge partitioning is an NP-Hard problem in general situation and an optimal solution is not viable practically for large circuits. However, many heuristic and randomized algorithms are developed for this problem to give a reasonable answer because of the importance of this problem.

As mentioned in the previous section, HMetis [14] solutions and tools are proposed by Karipis *et al* [14] on hypergraph partitioning and it uses a multilevel hyper edge paradigm. HMetis is a multi-level partitioner in which the main objective is that local cut size of each level is considered corresponding with the cut size of the next levels. In other means, HMetis is planned to use the global cut size information inclined with the local connection information of the netlist.

HMetis uses the hypergraph representation of the netlist in which nodes are standard cells and edges show the nets of the circuit. HMetis partitioning algorithm has three basic phases:

A. Coarsening Phase

In this phase highly connected hypergraph nodes are merged (coarsen) together and super nodes are generated. Each super-node consists of the nodes that have more connectivity and they should be clustered in a cluster. HMetis coarsen the hypergraph recursively until the number of super-nodes goes to 200 or less vertices in the hypergraph.

B. Initial Partitioning Phase

After the coarsening phase, hypergraph is reduced and a classical partitioning algorithm can be applied to partition the coarsened hypergraph into 2 parts. HMetis uses the Fiduccia-Mattheyses [15] algorithm to bisection the hypergraph. At the end of this phase, coarsened hypergraph is partitioned into two balanced partitions.

C. Uncoarsening and Refinement Phase

In this phase, the partitioned graph is un-coarsened and each super-node is expanded into basic nodes. After coarsening of each super-node, the balancing of the partitioned graphs may be violated. Therefore, a refinement algorithm should be applied in each level of the un-coarsening to balance the partitioned graph.

Fig. 1 shows the process of HMetis algorithm. In this figure, the various phases of the multilevel graph bisection. In the coarsening phase, the size of the graph is successively

decreased and the initial partitioning phase, a bisection of the smaller graph is computed and the Uncoarsening phase, the bisection is successively refined as it is projected to the larger graphs.

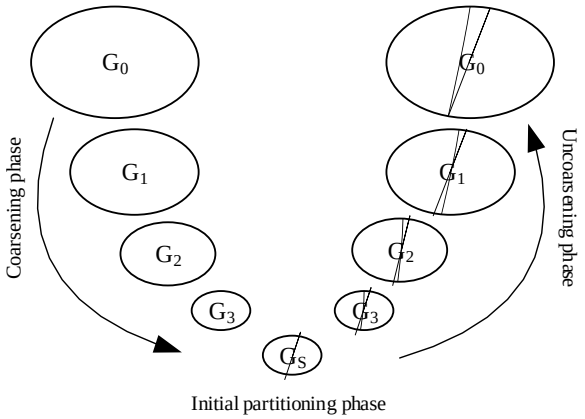

Fig. 1. HMetis hyper edge coarsening process.

As mentioned before, HMetis developed in the multilevel framework. The algorithm is very fast and has good results with high quality partitioning. Hypergraphs with over 100,000 nodes can be bisected in a few minutes. The most time consuming phase of the HMetis is coarsening phase. Therefore, we focused on accelerating this phase of the algorithm in this paper. The next subsection, describes this phase in more details.

IV. Coarsening Phase

In the coarsening phase, the goal is creating a hypergraph with smaller vertices to facilitate the initial partitioning phase. In general case, a hypergraph with more than 1000,000 vertices should be reduced to a hypergraph with fewer than 200 vertices. Coarsening process can be performed by three different methods as follows:

A. Edge Coarsening

In this method, a hyper edge with more than two vertices are selected and two of its vertices are combined together. In iteration a coarser graph with "v-1" vertices will be constructed. This process is shown in Fig. 2.

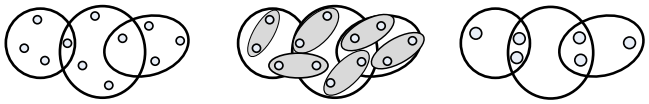

Fig. 2. Edge coarsening.

B. Hyper Edge Coarsening

In this method, hyper edges with fewer vertices are selected and their vertices are combined together to create a coarser graph and this coarsening is repeated until the goal of 200 vertices is achieved. This method is shown in Fig. 3.

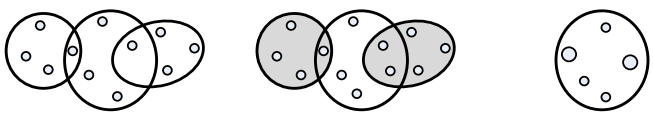

Fig. 3. Hyper edge coarsening.

C. Modified Hyper Edge Coarsening

In this method process is performed as hyper edge Coarsening method but after coarsening hyper edges, all the other vertices that are not combined with any other vertex, will be combined together. This technique is shown in Fig. 4.

We used the Hyper Edge Coarsening technique in our implementations.

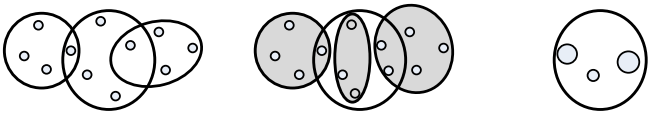

Fig. 4. Modified hyper edge coarsening.

V. Programming on GPU

To use the power of GPU in a general purpose application a framework or a toolkit is required to provide the programming, debugging and testing capabilities. In this section, we provide a brief overview of these tools from early day until now. In general, this field is quickly advancing and we advise the readers to check the internet for newest resources [16].

A. SH

It is an open source project to write C++ programs on GPUs. This tool is an independent platform and supports many kinds of graphics cards. The language of the main program could be anything but the language of the source code of the GPU program should be in C++. In early days it was a great tool to use the performance of the GPU to accelerate general purpose applications [17].

B. Direct Compute

Microsoft invented its own tool to use all the DirectX supported GPUs in Windows Vista, Windows 7 and Windows 8 for general purpose programming. Microsoft Direct Computer is an API to program on GPU and it was released with the DirectX 11 but it is compatible with DirectX 10 devices [18].

C. OpenCL

Open Computing Language known as OpenCL is a programming framework for writing programs that can run on heterogeneous platforms consisting CPUs, GPUs, DSPs and other processors. The language of programming is based of C99 standards. This language is used to write kernels (Functions that execute on OpenCL Processors) and there is and API to define and control the run of the whole program. We can write programs that are task-based parallel or data based parallel with OpenCL. Currently Many Processors are OpenCL supported [19]. They can execute OpenCL kernels on them.

D. Nvidia CUDA

CUDA (Compute Unified Device Architecture) is a programming framework created by Nvidia to write parallel applications that run on CUDA enabled GPUs [20]. CUDA unleashed some virtual instruction sets and memory for parallel computation in CUDA GPUs. CUDA framework enables developers to write programs that run on GPUs. The

main difference of this framework is that the programming is easy and it is in higher performance compared to other frameworks. The CUDA framework compiler is available for C, C++ and FORTRAN languages. NVIDIA's compiler for C/C++ CUDA is a llvm based compile called nvcc. CUDA is provided in Microsoft Windows, Linux and Mac OS. The SDK and drivers are downloadable in the NVIDIA's developer's website. All new Nvidia GPU's are CUDA compatible. There are 3 lines of NVIDIA GPU's known as GeForce, Quadro and Tesla. CUDA Binaries are compatible to future cards but some instructions are not available in older GPUs [15].

VI. Parallelizing the Coarsening Phase of Hypergraph Partitioning on GPU

As mentioned before, partitioning is a widely used algorithm in many of physical design tools. HMetis is a high-quality and fast partitioning algorithm that is known as the best hyper-graph partitioner in various applications such as VLSI-CAD. One of the most time-consuming parts of this algorithm is coarsening phase. To parallelize coarsening phase and implement it on GPU, the circuit hypergraph database should be distributed on GPUs. After this step, all of these parts must send to GPU cores for processing and then, GPU cores will do the coarsening of all parts in parallel. It is noting that all parts will be coarsened separately. Since all of the nodes for any part chosen randomly and coarsening operation can be done on the basis of connection between nodes, we must pay attention it is possible that there are a few connections or no connection at all between nodes. Other point we must consider is that some edges eliminated in the partitioning phase. These edges must affect in the final result of the coarsening phase.

It is worth noting that the quality of the coarsened graph in GPU is lower than the coarsened graph in CPU normally because, parallel coarsening of the GPU cores may tend to incorrect decisions, because the graph is partitioned into smaller sub-graphs and each core doesn't have a global view. We divided the netlist based on its nets to divide the netlist with lower level of connections. As will be seen in experimental results, this database distribution makes good results with considerable runtime improvement. Fig. 5 shows the parallelized algorithm.

Parallel Coarsening pseudo code	
Step 1	Read input hyper graph *g*.
Step 2	Partition *g* into *n* parts.
Step 3	Create *n* sub-graphs of *g*.
Step 4	FOR each sub graph g_i of n sub graphs DO
Step 4-1	Copy sub graph g_i into GPU global memory.
Step 5	Call GPU kernel to parallelize following tasks.
Step 5-1	While (number of vertices in this part > k) LOOP
Step 5-1-1	Find an edge *e* in sub graph.
Step 5-1-2	Merge all vertices *e*.
Step 6	Wait for all kernels to complete.
Step 7	Take back all the results from GPU global memory.
Step 8	Merge all coarsened sets in g.
Step 9	While the number of vertices in g is greater than 200 LOOP
Step 9-1	Find an edge *e* in *g*.
Step 9-2	Merge all vertices of *e*.
Step 10	Write the output.

Fig. 5. Parallelized coarsening algorithm.

VII. Experimental Results

We implemented the proposed parallel algorithm in C++ using CUDA platform on Nvidia Geforce 295 GTX GPU system. Eleven benchmark are selected from IWLS suite [21] to evaluate the quality and runtime speedup of the proposed algorithm. Statistical information of the attempted benchmarks is shown in Table I.

TABLE I: General Characteristics of Attempted Benchmarks Overall

Benchmark	Cell Count	Net Count
S15850	685	685
S13207	1219	1214
SPI	3227	3274
S38584	6724	6704
S35932	7273	7277
S38417	8278	8220
Mem_ctrl	11440	11398
Wb_conmax	29034	28739
B17	37117	37102
B17_1	37383	37369
DES_perf	98341	98391

TABLE II: Execution Time and Quality of Coarsening on CPU Platform

Benchmark	Super Node	Internality	Runtime (s)
S15850	188	68%	0.049
S13207	190	81%	0.131
SPI	200	95%	0.129
S38584	200	98%	0.426
S35932	200	96%	0.587
S38417	200	98%	0.611
Mem_ctrl	199	99%	1.154
Wb_conmax	200	99%	6.382
B17	200	99%	10.132
B17_1	200	99%	10.192
DES_perf	200	99%	61.367

We proposed and implemented several methods for initial partitioning graph node for each GPU thread. The simplest idea was to assign the nodes randomly to each thread. The second approach was to divide nodes based on the occurrence of the nodes in the input file. Another idea was to pick nodes based on nets and its neighbors. The last idea was to use BFS algorithm to pick nodes for each thread. As we tested, we figured out that the method to divide nodes based on the input file wat the best one, because netlist files are generated such that the connected nodes are neighbors in the file and it is not random.

As mentioned before, the quality of a parallelized algorithm may be degraded because each GPU core should decide based on its local information. Therefore, quality of the coarsening is an important parameter that must be considered. In other words, in addition to the low execution time, coarsening phase should have considerable quality results. The quality of a coarsened graph does not have any standard measure. We defined a new metric called internality to evaluate the quality of a coarsened graph. Average Internality of graph g is defined as the average of internality of each hyper edge of the graph.

$$\text{AverageInternality}(g) = \frac{\sum_{e}^{`}\text{Internality}(e)}{|g|} \quad (1)$$

Internality of an edge is formulated as degree of locality of the nodes in the edge. For example, consider an edge with 2 vertices. If both of them are coarsened together the internality is 100% and if they are in different sets the internality is 0%. If an edge with 3 vertices has 2 vertices in a set and one vertex alone the internality is 33%. The internality of an edge can be computed as:

$$\text{Internality}(e) = \frac{\sum_{v-\text{in-groups}} |v|^2 - |e|}{|e|^2 - |e|} \times 100 \quad (2)$$

TABLE III: EXECUTION TIME AND QUALITY OF COARSENING ON GPU PLATFORM

Benchmark	PART8			PART16			PART32		
	Super Node	Internality	Runtime (s)	Super Node	Internality	Runtime (s)	Super Node	Internality	Runtime (s)
S15850	191	62%	0.218	193	63%	0.156	190	69%	0.14
S13207	195	77%	0.483	192	79%	0.25	189	72%	0.156
SPI	200	91%	3.12	200	92%	1.546	199	94%	0.789
S38584	198	95%	14.38	199	94%	6.418	1999	94%	2.879
S35932	199	91%	16.638	196	88%	7.058	197	96%	2.632
S38417	198	93%	20.673	193	95%	9.475	199	97%	3.972
Mem_ctrl	198	94%	40.641	191	93%	19.562	200	95%	9.908
Wb_conmax	200	95%	259.638	200	95%	108.769	200	96%	43.589
B17	200	97%	423.564	200	98%	198.812	200	98%	87.68
B17_1	196	93%	425.792	200	95%	199.171	200	97%	87.826
DES_perf	NR	NR	NR	200	99%	787.644	200	99%	333.079

TABLE IV: EXECUTION TIME AND QUALITY OF COARSENING ON GPU PLATFORM

Benchmark	PART64			PART128			PART256		
	Super Node	Internality	Runtime (s)	Super Node	Internality	Runtime (s)	Super Node	Internality	Runtime (s)
S15850	188	65%	0.109	192	69%	0.109	190	69%	0.141
S13207	192	72%	0.198	191	72%	0.193	192	72%	0.204
SPI	199	92%	0.438	200	93%	0.256	200	93%	0.242
S38584	199	93%	0.941	199	95%	0.412	200	95%	0.271
S35932	199	93%	0.771	200	96%	0.381	200	96%	0.283
S38417	200	95%	1.301	200	96%	0.475	200	96%	0.294
Mem_ctrl	200	96%	2.513	200	97%	0.756	200	97%	0.381
Wb_conmax	200	99%	11.608	200	99%	4.657	200	99%	2.373
B17	200	99%	23.105	200	99%	6.472	200	99%	2.813
B17_1	200	99%	23.511	200	99%	6.248	200	99%	2.844
DES_perf	200	99%	95.488	200	99%	26.988	200	99%	8.588

TABLE V: COMPARSION BETWEEN VARIOUS IMPLEMENTATION OF THE PARALLEL ALGORITHM VS. SERIAL ALGORITHM

Benchmark	64 PARTS		128 PARTS		256 PARTS	
	RT	INT	RT	INT	RT	INT
S15850	0.45	-4.41%	0.45	1.47%	0.35	1.47%
S13207	0.66	-11.1%	0.68	-11.11%	0.64	-11.11%
SPI	0.29	-3.16%	0.50	-2.11%	0.53	-2.11%
S38584	0.45	-5.10%	1.03	-3.06%	1.57	-3.06%
S35932	0.76	-3.12%	1.54	0.00%	2.07	0.00%
S38417	0.47	-3.06%	1.29	-2.04%	2.08	-2.04%
Mem_ctrl	0.46	-3.03%	1.53	-2.02%	3.03	-2.02%
Wb_conmax	0.55	0.00%	1.37	0.00%	2.69	0.00%
B17	0.44	0.00%	1.57	0.00%	3.60	0.00%
B17_1	0.43	0.00%	1.63	0.00%	3.58	0.00%
DES_perf	0.64	0.00%	2.27	0.00%	7.15	0.00%

We executed the algorithm on both CPU and GPU on each test case to check the results of our work. We used an Intel Core i7 2670QM on Windows7 OS with 8 Gigabytes of Main memory as CPU platform and the GPU that is utilized, is an Nvidia GeForce 295 GTX with 2 Gigabytes of Dedicated memory. Table II represents the runtime of and internality of coarsening phase on CPU in each test case. In this table, column "Super Node" represents the number of super-nodes

in coarsened graph.

Experimental results of running the algorithm on GPU are shown in Table III and Table IV. The Part variable shows the number of parallel threads of CUDA.

It is worth noting that execution of the last benchmark with 8 parts has been too time consuming to make reports.

As can be seen in Table IV, Runtime can be reduced considerably without significant quality degradation. It is worth noting if the quality is reducing because we need both runtime and quality. From our results we can see that if the problem size is small GPU overheads takes advantage over computing the result but in larger problems GPU shows its capability and the run time improves much better.

Table V shows the Overall comparison between various implementation of the parallel algorithm vs. serial algorithm. In these algorithm columns RT and INT show the percentage of runtime improvement and quality degradation of parallel algorithms compared to serial implementation on CPU.

VIII. Conclusion

Significant portion of total digital design flow runtime is related to various stages of the physical design such as partitioning, floor planning, and placement and routing. In this paper, a new parallel partitioning algorithm was proposed for GPU system. In the proposed algorithm, coarsening phase of the partitioning was accelerated by parallelizing on GPU. Experimental results show that runtime can be improved up to 7x luck circuit with negligible quality degradation. Our analyses show that the results are better for larger circuits with more part number.

References

[1] K. A. Sumithra. Algorithm for CAD tools VLSI design. [Online]. Available: http://www.intechopen.com/books/vlsi-design/algorithms-for-vlsi-cad-tools

[2] M. F. Chang, "CDMA/FDMA-interconnects for future ULSI communications," in *Proc. International Conference on Computer Aided Design*, pp. 975-978, 2005.

[3] ITRS reports. [Online]. Available: http://www.itrs.net

[4] S. Sapatnekar *et al*., "Reinventing EDA with many core processors," in *Proc. the 45th Annual Design Automation Conference*, pp. 126-127, 2008.

[5] B. Catanzaro, K. Keutzer, and B. Y. Su, "Parallelizing CAD: A timely research agenda for EDA," in *Proc. Design Automation Conference*, pp. 12-17, 2008.

[6] M. Sanjabi, N. Miralaei, S. Amanollahi, and A. Jahanian, "ParSA: parallel simulated annealing placement algorithm for multi-core systems," in *Proc. International Symposium on Computer Architecture and Digital Systems (CADS)*, pp. 19-24, 2012.

[7] F. Lotfifar and M. Johnson, "A multi-level hypergraph partitioning algorithm using rough set clustering," in *Proc. International Conference on Euro-par*, pp. 14-23, 2015.

[8] H. Liu, L. Latecki, and S. Yan, "Dense subgraph partitioning of positive hypergraphs," in *Proc. IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 541-554, 2015.

[9] C. Fobel, G. Grewal, and D. Stacey, "GPU-accelerated wire-length estimation for FPGA placement," in *Proc. International Conference on IEEE Computer Aided Design*, pp. 14-23, 2011.

[10] J. Cong and Y. Zou, "Parallel multi-level analytical global placement on graphics processing units," in *Proc. International Conference on Computer Aided Design*, pp. 681-688, 2009.

[11] D. LaSalle and G. Karypis, "Multi-threaded graph partitioning," *IEEE International Parallel & Distributed Processing Symposium*, Boston, Massachusetts USA, May 20-24, 2013.

[12] A. Farkish and A. Jahanian, "Parallelizing the FPGA global routing algorithm on multi-core systems without quality degradation," *Institute of Electrical, Information and Communication Engineers Electronic Express Journal*, vol. 8, no. 24, pp. 2061-2067, 2012.

[13] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Design and implementation of move-based heuristics for VLSI hypergraph partitioning," in *Proc. Asia and South Pacific Design Automation Conf.*, pp. 661-666, Jan. 2000.

[14] G. Karypis, R. Aggarwal, and V. Kumar, "Multilevel hypergraph partitioning: Applications in VLSI domain," in *Proc. International Conference on IEEE Transaction on very Large Scale Integration (VLSI) Systems*, pp. 69-79, 1999.

[15] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. 19th IEEE Design Automation Conf.*, pp. 175-181, 1982.

[16] A. Sheppard, *Programming GPUs*, 2013.

[17] Embedded meta programming language. [Online]. Available: http://www.libsh.org/about.html

[18] D. Tarditi, S. Puri, and J. Oglesby, "Using data parallelism to program GPUs for general-purpose uses," *Technical Report, Microsoft Research*, 2006.

[19] OpenCL supported products. [Online]. Available: https://www.khronos.org/conformance/adopters/conformant-products

[20] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed., 2010.

[21] IWLS benchmarks. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

Atefe Taheri was born in Babol, Mazandaran, Iran in 1987. She received the bachelor degree in hardware engineering from Shahid Beheshti University, Tehran, Iran in 2010 and her master's degree in hardware architecture from Shahid Beheshti University, Tehran, Iran in 2013.

She then worked for some companies such as SAAT Co, Tehran, Iran as a hardware designer. Currently she is employed as hardware/software designer in Intelligent Information Solutions Center in Sharif University, Tehran, Iran. Her current research interest is CAD, High Performance Computing and Hardware/Software Codesign.

Ali Jahanian received the B.Sc. degree in computer engineering from Tehran University, Tehran, Iran in 1996, and the M.Sc. and Ph.D. degrees in computer system architecture from Amirkabir University of Technology, Tehran, Iran in 1998 and 2008, respectively.

He is currently an assistant professor of Electrical and Computer Engineering Department of Shahid Beheshti University. His current research interest consists of VLSI design automation, Emerging on-chip interconnect technologies, and embedded system design.

Behin Molaie received his bachelor's degree in computer engineering from Sharif University of Technology, Tehran, Iran in 2009 and the M.S. in software engineering from Sharif University of Technology, Tehran, Iran in 2015. He is currently working toward the Ph.D. degree in Software Engineering at Sharif University of Technology, Tehran, Iran.

He worked for Kamasystem company, Tehran, Iran as a technical manager for 5 years. Currently he is employed as a technical team manager in Intelligent Information Solutions Center in Sharif University of Technology, Tehran, Iran. His current research interest is IoT, big data and parallel computing.

Mr. Molaie won a gold medal in Iran National Olympiad in informatics in 2004