Alexander A. DOUDKIN [1], Albert I. PETROVSKY [1], Aleksey V. OTVAGIN [2], Rauf Kh. SADYHOV [2]
[1] United Institute of Informatics Problems of NAS of Belarus, Minsk, doudkin@newman.bas-net.by
[2] Belarusian State University of Informatics and Radioelectronics, 6 P. Brovka st, Minsk, Belarus

MULTI-AGENT PARALLEL IMPLEMENTATION OF VLSI CAD PROCEDURES

Summary

The integrated framework for parallel processing of data describing integrated circuits layouts that based on a graph-oriented parallel algorithm representation is represented. A parallel program is developed from single computational units (grains) in specialized visual editor. This visual schema is translated into XML form that is interpreted by multi agent runtime system, based on a MPI library. The runtime system realizes a dynamic optimization of parallel computations with the algorithm of virtual associative network. The proposed tools are well suited for rapid development, analysis and execution of parallel algorithms with adaptation to specific cluster architecture.
Key words*: manufacturing; software; parallel application models; graph models; optical inspection; image analysis*

1. Introduction

The modern semiconductor manufacturing needs to operational inspection all of the critical procedures of VLSI manufacturing. An optical inspection is the important part of such control solutions. It implies the presence of some operative analysis system [1] providing image registration, visual information processing and analysis.

The effective processing of large amounts of inspection data can be achieved only with use of modern software design technologies. The one of the leading modern technologies is a parallel processing. The great obstacle for the broad use of this technique is a necessity of solution of additional tasks about optimization of developed applications for greater performance. Providing and using of high-level abstractions can greatly increase a quality and speed of application development. Developers have a feasibility of a rapid transfer of existing applications and algorithms to parallel platforms.

There exists a necessity of a development of integrated tool suite for easy development, analysis and optimization of parallel applications. This tool must hide a specific mechanisms of parallelism realization that programmer can to attend to information processing algorithm. The tool also must provide a visual program presentation and features for reengineering and adjusting the program structure to specific cluster architecture.

The basis of parallel application development is a certain computational model. The one of a broadly used is a task-parallel model that is perfect suited for realization of many parallel applications. This model is very good expressed in graph-oriented visual representations. The extension of graph model by specification of its elements can be used for development of complicated applications.

In many cases the development of newly created parallel applications is made on the basis of existing sequential algorithms and their composition. The transfer and adaptation of existing code for parallel virtual machines requires many expenses. The realization of algorithm parts in form of independent components allows using component development paradigm, when the program is constructed from large blocks of code. In this case the time of development and testing of program can be reduced significantly.

The process of parallel program execution itself needs an application of sophisticated methods for load balancing, planning and optimization. In many cases the nodes of parallel system can have temporal or spatial heterogeneity. These systems must be equipped by means and tools of runtime control and dynamic reconfiguration to achieve a high performance rates.

We propose an integrated tool suite for development, analysis and support of parallel processing. This one has a visual editor, translator, optimization tool and a runtime system, based on MPI [2] library. The use of MPI makes our tools suitable for wide range of parallel computers. In the proposed system we realize next remarkable features:

1. Visual representation of algorithm, based on a graph-oriented form and task-parallel computation model. We introduce a concept of a computational grain that is a single unit of processing. The grain can be developed in different languages and must perform a dedicated interface for integration in parallel application.
2. Application mobility that is realized through a grain libraries that are attached to runtime system. The tools of our framework also are developed in platform-independent manner by use of open standards and tools (C++, Java, MPI).
3. Static and dynamic optimization of parallel algorithms with use of virtual associative network algorithm. This algorithm is a variant of hybrid genetic algorithm and ensures a fast search of optimal solutions. We use two modifications of algorithm for static and dynamic optimization respectively.
4. A dynamic mapping of grains on the processors of a computer cluster with use of information that is collected in runtime. This information can be used for scheduling of parallel processing application.

The tools for parallel processing support use modern architectures, in particular, an agent-oriented approach. This one allows realizing a complex behavior for optimization of parallel computing under circumstances of instable resource and computation load.

This paper is organized as follows. In section 1 the graph-oriented model of program construction is considered and the concept of computational grains is introduced. Section 2 describes the main parts of a framework and its cooperation in program development process. Section 3 is dedicated to design and implementation of runtime system for parallel applications support. Section 4 describes experimental results of practical use of proposed tools for parallel processing optimization.

2. A parallel application model and a computational grain concept

The basic principles of creation a graph-oriented parallel program representation are defined in previous paper [3]. The scenarios for data processing are represented in the form of Directed Acyclic Graph (DAG). DAG is represented as a tuple $G = (V, E, W, C)$, where:

- V is a set of graph vertices $v_i \in V, 1 \le i \le N$. Each vertex is associated with data processing operation. A set of graph vertices represents decomposition of parallel dataflow processing program on the separated operations;
- E is a set of graph edges $\{e_{i,j} = (v_i, v_j)\} \in E, i = \overline{1,N}, j = \overline{1,N}, i \ne j$. An edge represents a precedence relation between operations in scenario and determines a data transfer between these nodes;
- W is an operation cost matrix;
- C is an edge cost set, where $c_{i,j} \in C$ determines the communication volume between two data processing operations, which is transferred by edge $e_{i,j} \in E$. We consider those operations, which are related and connected by the edge, use an identical data format for a predecessor output and a successor input. For all scenarios, particular edge has an equal cost.

The development of dataflow processing application includes few stages:

- building of a scenario DAG that describes logical structure of application;
- assignment of operations to graph vertices and editing of operations parameters for each datatype;
- mapping of scenario DAG to cluster architecture.

Each computational operation in scenario is realized as a separate unit called grain. These grains uses specific interface for integration into framework and data exchange. The design of grain makes possible a rapid adaptation of existing processing algorithms into parallel application. These algorithms are transformed to objects that are capable to form its own calling context on the base of received parameters. Each operation interprets its parameter string by convenient way and converts parameters to variable name or to constant value. The order and rules of parameter transform are determined by operation specification.

All of the operations work with specific data storage facility that is incorporated into framework architecture. The storage realizes a shared memory abstraction for source data and results of processing. A variant of shared memory is realized on a shared file system that is common for many cluster architectures. The storage interface provides operations for writing and reading of variable with defined name that is used for its identification. There exists also a intermediate storage mechanism in local memory of each processor, where the result of this processor operations are stored. This one allows reducing time expenses for variable reading in case of repeated access.

The parameters of operation are read from storage. A single parameter is identified by object name, represented as a string. Each parameter value is placed into corresponding internal grain variable, thus all the parameters forms a calling context. Further the operation is executed and results of processing are placed in storage. At this moment these values are accessible to another grains in parallel application.

The grains are collected in specific libraries that are dynamically linked to parallel application. The grain is loaded from library when it's needed and is identified by operation name. The realization of specific grain libraries for different classes of processing algorithms easy expands the application area of proposed framework.

An example of the parallel program graph is presented in Fig. 1, where each operation is denoted as *Ci* with a cost vector. A cost of an information transfer between contiguous operations is equal for all data types. Some operations are strictly oriented on a specific processor while others can be placed on each processor in cluster. If the operation cost for some type of data is equal to zero, then this operation must be skipped for the selected type of data.

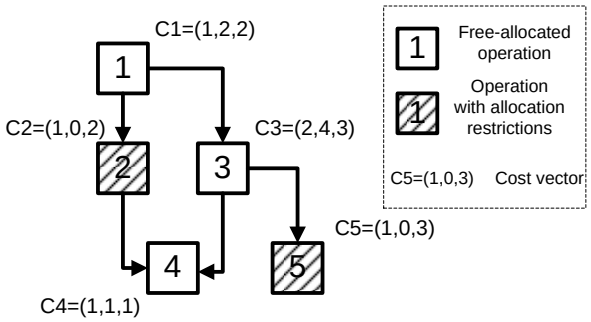

Fig. 1. An example of program graph

A matrix of restrictions $Z(O,P)$ is formed according to the following rules:
$Z(O,P) = 1$, if processor p allows execution of operation O;
$Z(O,P) = 0$, otherwise.

The matrix of restrictions is used in optimization procedures and prevents an erroneous allocation of the specified operations on some processors. The restrictions arise because of a heterogeneous cluster structure and different operations requirements.

3. The framework architecture

The architecture of framework and its main components are shown on the Fig. 2.

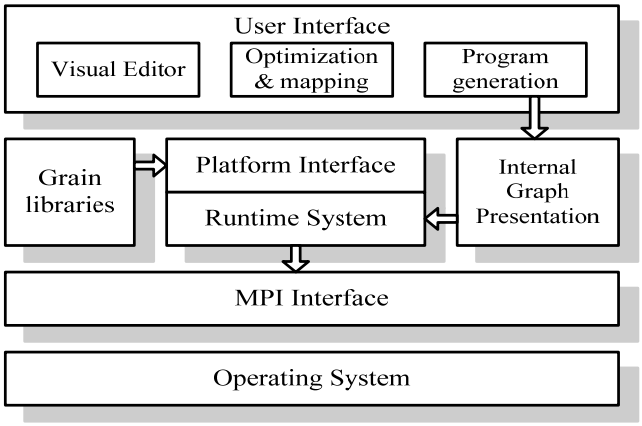

Fig. 2. The architecture of a framework

The user interface is a collection of tools for visual development and analysis of parallel application. It contains a specific visual editor that uses a graph-oriented model of algorithm representation. With this editor user can draw a logical algorithm graph, map operations to graph vertices,

and define parameters for operation execution to process different data objects. For analysis of deterministic data lows editor makes possible to define a dataflow pattern as a queue of objects with corresponding types. Programmer also can create a cluster topology and define performance characteristics of each node.

The optimization and mapping tools contains a simulation model and optimization algorithm. The simulation model is used for evaluation of schedules, produced by optimization algorithm and for visual representation of best one. User can compare two different schedules in graphical form as a Gantt chart and manually change schedule to achieve best results.

There exist many algorithms of DAG scheduling that use various optimization techniques and heuristics. The techniques include priority based list scheduling, for example, algorithms HLF (Highest Level First), LP (Longest Path), CP (Critical Path) [4-6]. Another technique is clusterization, and such algorithms, as DSC (Dominating Sequence Clustering) [7], and Sarkar algorithm [8], belong to this technique. Another perspective search techniques use evolutionary optimization. These techniques are based on such algorithms, as a tabu search [9], simulated annealing, and genetic algorithms [10]. The most powerful is a genetic algorithm (GA) technique, and many of algorithms are proposed in this field. However, the classical genetic algorithm is a blind search technique. To speedup genetic algorithms we proposed an algorithm of virtual associative network [11-13], which belongs to a class of hybrid algorithms.

The solutions, created by the virtual network algorithm, are stored in XML form together with the parallel algorithm representation. This file is interpreted by runtime system that is constructed as a multi agent application. The runtime system is build on top of MPI library. The architecture of runtime system isolates parallel application logic from basic tools for parallel process creation and control. This approach makes the parallel applications more platform-independent and flexible.

4. Design and implementation of runtime multi agent system

The runtime subsystem for dataflow processing is designed as distributed multi-agent system [14]. The system consists of two types of program agents: a coordinator and an executor. All agents are realized as MPI processes and use MPI facilities for execution control and data exchange. The principles of system functioning allow to use it for processing both deterministic and stochastic image flows. The architecture of runtime system is presented in Fig. 3.

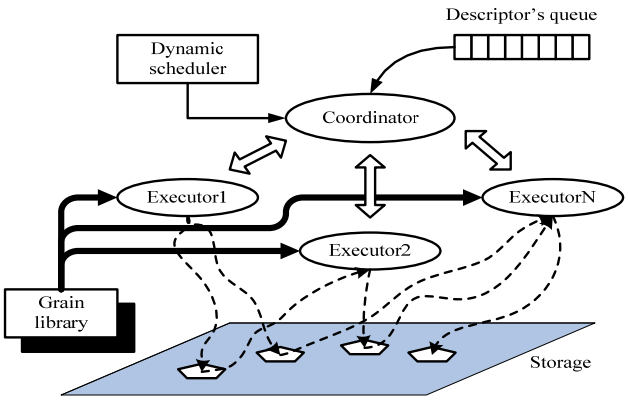

Fig. 3. Runtime system architecture

The system uses a "master-slave" approach. Coordinator is a main process that controls the logic structure of a parallel algorithm. It contains a descriptor queue for all of processing objects. Each descriptor determines a type of object and a current grain that must be executed for this one. The descriptor queue contains descriptors for all of ready operations. The main task of coordinator is in transferring ready descriptors to free executors accordingly to operation to processor mapping. The coordinator also checks a moment of some operation finish and places new ready descriptors to the queue.

An executor agent is an abstraction of a real physical processor. The main task of this one is execution of computational grains that are received from coordinator. The executor works with the grains library and loads required grains for execution on dedicated processor. After completion of grain, the descriptor is returned to the coordinator. The executor works while stop instruction is not received from the coordinator. The interaction between agents is performed through shared memory storage interface. All synchronization tasks are performed by coordinator thus ensures deterministic parallel computations.

In case of processing a stochastic dataflow the fixed operation to processor mapping can be ineffective. The runtime agents continuously checks system state and characteristics. These characteristics are collected in the coordinator and used for runtime optimization. The optimization is based on the measuring of data processing speed. When the dataflow changes its pattern significantly, the system must adapt to this situation. The adaptation procedure performs reconfiguration of the operation mapping for all processors. When this reconfiguration is done, the coordinator applies new scheme to transfer the descriptors. The system tries to adapt to changed conditions and to achieve a high processing speed.

5. Experimental results

The proposed tools for parallel processing of data flows are used for development of applications to process integrated circuits layers images. The library of grains contains in this case a set of preprocessing operations, and operations for contour detection and uniforms area selection. The data types for these operations (images, filters and so on) are placed in library too.

For evaluating of proposed algorithms and a framework two series of experiments have been made. First, we studied the deterministic flows, which had a fixed number of objects with known types. The second group of experiments was run with stochastic image flows, where the input data were generated randomly.

The results of a first group experiments shows that the algorithm of virtual networks for static scheduling finds better solutions and the performance of the algorithm is increased, when the complexity of schedule is increased too. The algorithm based on virtual networks finds solutions faster, than classical GA and requires fewer computations.

The second group experiments with stochastic image flows shows that the dynamic optimization of operation mapping in dataflow processing can significantly improve the processing rate. The modified virtual associative network algorithm brings very small overhead to parallel application execution.

6. Conclusions

The development of parallel applications with specific integrated component frameworks makes possible to significantly increase the speed and quality of all project procedures. The use of visual editing, automatic analysis and optimization can attract new users to this area of computing.

The architecture of framework based on a multi-agent paradigm can be easy adapted to many industrial applications that requires a parallel processing. The flexible application construction and runtime adaptation allows obtaining a high performance of application. The proposed tools are easy expandable with the new grain libraries, storage mechanisms and optimization algorithms.

7. References

[1] Voganti M., Ercal F., Dagli C., Tsunekawa S.: Automatic PCI Inspection Algorithms: A Survey, Computer Vision and Image Understanding, 1996, 63, p. 287-313.

[2] Gropp W., Lusk E., Skjellum A.: Using MPI: Portable Parallel Programming with the Message Passing Interface. MIT Press, 1995.

[3] A Framework for Parallel Processing of Image Dataflow in Industrials Applications / Aleksej Otwagin, Alexander Doudkin // Proceedings of the fourth International Conference on Neural Networks and Artificial Intelligence (ICNNAI'2006), May, 31-June, 2, Brest, Belarus /Brest: BSTU, 2006, p. 162-167.

[4] Macey S., Zomaya A.Y.: A performance evaluation of CP list scheduling heuristics for communication intensive task graphs // Proc. of IPPS/SPDP. 1998, p. 538-541.

[5] Menasce A., Saha D. et al.: Static and dynamic processor scheduling disciplines in heterogeneous parallel architecture. Journal of Parallel and Distributed Computing, 1995, Vol. 28, pp. 1-18.

[6] Oh H., Ha S.: A Static Scheduling Heuristic for Heterogeneous Processors. Second International EuroPar Conference Proceedings. Vol. II. Lyon, France, 1996, p. 573-577.

[7] Gerasoulis A., Yang T.: A comparison of clustering heuristics for scheduling directed acyclic graphs onto multiprocessors. Journal of Parallel and Distributed Computing, 1992, №4 (16), p. 276-291.

[8] Sarkar V.: Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. The MIT Press, 1989.

[9] Porto S., Ribeiro A.C.: A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints. International Journal of High-Speed Computing, 1995, №2 (7), p. 45-71.

[10] Michalewicz Z.: Genetic Algorithms + Data Structures = Evolution Programs. Second, Extended Edition. Springer-Verlag, 1994.

[11] Yufik Y.M., Sheridan T.B.: Virtual Networks: New framework for operator modeling and interface optimization in complex supervisory control systems. A Rev. Control, Vol. 20. p. 179-195.

[12] Sadykhov R.Kh., Otwagin AV.: Solution search algorithm of solution search for systems of parallel processing based on a virtual neural network model. Automatic Control and Computer Science, Vol. 35 (1), Allerton Press Inc., New York, 2001, p. 25-33.

[13] Sadykhov R.Kh., Otwagin A.V.: Algorithm for optimization of parallel computation on the basis of genetic algorithms and model of a virtual network. Proceedings of the International Workshop on Discrete-Event System Design DESDes'01, Przytok, Poland, June 27-29, 2001, p.121-126.

[14] Poslad S., Buckle P., Hadingham R.: Open Source, Standards and Scaleable Agencies. International Workshop on Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, June 03-07, 2000, Manchester, UK, p. 296-303.