A Survey of Current and Future Research Directions in Parallel CAD

Prithviraj Banerjee
Computational Science and Engineering Program
University of Illinois
1308 W Main Street
Urbana IL 61801
banerjee@crhc.uiuc.edu

Abstract

In view of the increasing complexity of VLSI circuits of the future, the requirements on VLSI CAD tools will continuously increase. Parallel computing is becoming gradually recognized as a popular vehicle to support the increasing computing requirements of future CAD tools. In this paper, we will provide a brief survey of parallel CAD research, and discuss some future directions.

1 Introduction

Parallel computing is becoming an increasingly cost-effective and affordable means for providing enormous computing power and represents a challenge to costly conventional supercomputers. Although it is relatively easy to build parallel machines whose peak performance are thousands of MFLOPS and MIPS, it is extremely difficult to harness the computational power effectively to actually have the machines deliver all those MFLOPS and MIPS to the user. A major challenge in parallel computing is to design efficient, parallel algorithms that can use the hardware resources to get the maximum performance. This paper will discuss the use of parallel processing for solving problems in a growing application area whose computational requirements are enormous: very large scale integrated (VLSI) circuit computer-aided design (CAD) applications.

In view of the increasing complexity of VLSI circuits, there is a growing need for sophisticated CAD tools to automate the synthesis, analysis, and verification steps in the design of VLSI systems. Future CAD tools have to enable designs that are too large or complex to undertake otherwise, shorten design time, improve product quality, and reduce product costs. Parallel processing offers an effective solution to handle the large complexity of VLSI designs of the future.

In this paper, we will provide a brief survey of parallel CAD research over the last decade. For more details in this topic, the reader is referred to a recent book on the subject [1]. Section 2 provides some motivation behind why it is important to develop parallel CAD algorithms. Sections 3 through 9 survey parallel implementations in various CAD applications such as cell placement, wire routing, layout verification, circuit simulation, logic simulation, test generation, fault simulation, logic synthesis and verification. We highlight various commercial implementations of parallel CAD tools in Section 10. Finally, we conclude with a look at the future of parallel CAD in Section 11.

2 Motivation for Parallel CAD

The use of parallel processing for VLSI CAD applications is beneficial for the following reasons.

2.1 Faster Runtimes

Almost all the VLSI CAD applications in the synthesis, analysis, and verification tasks at various levels in the VLSI design hierarchy (functional, logical, circuit, and physical levels) take large runtimes on existing sequential computers. Future CAD tools will require even more accuracy and computational capabilities from these tools. It is clear that CAD tools that take hours to run on current designs consisting of tens of thousands of gates, may take weeks or months to run on future designs consisting of millions of gates. Given such increased complexity of CAD tools of the future, one of the effective ways to address the complexity of the problem is to apply parallel processing to speed up the CAD tasks. Many problems are suitable for parallelization, hence parallel processing can offer faster runtime performance for many CAD applications.

2.2 Larger Problem Sizes

Very often, a particular CAD tool cannot handle a large problem because of its unaffordable runtime

*Acknowledgement: This research was supported in part by the Semiconductor Research Corporation under Contract SRC 94 DP-109, and by the Advanced Research Projects Agency administered by the U.S. Army Research Office under Contract DAAH04-94-G-0273.

requirements or memory limitations. Given the same amount of design turnaround time from a designer, parallel processing allows the user to address larger problems since they are solved faster. Also, since parallel machines come with larger memories, one can solve larger problem sizes than could be solved earlier on conventional machines. For example, it has been observed that circuit extractors that cannot extract large circuits containing millions of rectangles on a workstation due to memory limitations can be extracted on a parallel processor by suitably partitioning the rectangles of the chip area among the processors.

2.3 Better Quality of Results

Since many VLSI CAD problems can be formulated as optimization problems that are NP complete, heuristics are often used to solve the problems, which often give sub-optimal solutions. By using parallel search techniques, it has been observed that one can frequently obtain better quality results. For example, it has been observed that a parallel implementation of an automatic test pattern generator obtains higher fault coverage than the corresponding serial implementation.

2.4 Affordable and Available Technology

Parallel processing has become affordable and available recently owing to the availability of low-cost, high-performance microprocessors and memories. Medium-priced moderately parallel multiprocessors and multiprocessor workstations are now available readily.

2.5 Efficient Parallel Algorithm Design

It is often claimed that, rather than investigating parallel algorithms, it is perhaps better to investigate sophisticated techniques to automatically parallelize programs as is done by parallelizing compilers. However, the status of existing parallelizing compilers is such that they can only work on programs in languages such as FORTRAN with large numbers of independent DO LOOPS working on array-type data structures. While these techniques are appropriate for structured numerical applications involving dense matrices, they will not be able to extract sufficient parallelism automatically from VLSI CAD applications. By nature, algorithms for VLSI CAD applications are highly unstructured (for example, they do not have well-structured DO LOOPS and array data structures). Hence, to speed up VLSI CAD applications, it is necessary to investigate parallel algorithms for them.

In the following sections, we will briefly review parallel implementations of VLSI CAD tools across a large variety of applications including placement, routing, design rule checking, circuit and logic simulation, test generation and logic synthesis.

3 Parallel Placement

The VLSI cell placement problem involves placing a set of cells on a VLSI layout for a given netlist that provides the connectivity between each cell and a library containing layout information for each type of cell. The primary goal of cell placement is to determine the best location of each cell so as to optimize one or more objectives, such as minimization of the total area of the layout or minimization of the delays on interconnect lines. Since these problems are NP complete to begin with, heuristic approaches are used to solve such problems. Newer heuristics are being continually developed. Numerous parallel algorithms have been developed for the placement problem. We now summarize the basic features and advantages of each parallel algorithm.

The parallel algorithm based on iterative pairwise exchange assigns a cell or a group of cells to each processor of a N processor system, and considers the exchange of $N/2$ pairs of adjacent cells simultaneously. If the total wirelength associated with a pair of cells can be decreased by the exchange, these two cells are actually exchanged. Otherwise, they are not exchanged. During each proposed exchange of a cell pair, it is assumed that the other cells are all fixed in their current positions. The parallel algorithm is suitable for execution on SIMD and MIMD parallel machines. The advantage of this algorithm is that it has a large degree of parallelism. The main disadvantage is that the quality of the solutions produced by this algorithm is quite poor. A second disadvantage is that due to interaction among parallel moves, there is a chance of oscillation around the placement solution.

The parallel algorithm based on force-directed placement is based on the notion of using forces to bring cells with greater connectivity closer together. Cells are assigned to different processors, and at each step, a cell's new position is computed on the basis of the current positions of the cell's logical neighbors. The algorithm is suitable for SIMD parallel machines. The advantage of this algorithm is that there is a large degree of parallelism, and the solution quality obtained from the algorithm is fairly good, better than the pairwise exchange algorithm, but inferior to simulated annealing.

Several parallel algorithms based on simulated annealing have been proposed. At each iteration, a cell is randomly selected, and moved to or exchanged with a cell at another random location. If the resultant move decreases the overall cost function, the move is accepted. If the move results in an increase in the value of the cost function, the move is accepted with a probabilistic function based on the temperature. The parallel algorithm based on move decomposition breaks up the task involved in simulated annealing into a set of subtasks and allocates each task to a different processor. This approach is only suitable for shared memory MIMD machines and typically gives only small speedups of about 2 to 3. The parallel algorithms based on parallel move evaluation allow different processors to propose different moves in parallel; these approaches give large speedups, and are suitable for both SIMD and MIMD machines, but one has to be careful about interactive parallel moves. Parallel algorithms based on parallel evaluation and acceptance of noninteracting moves give low speedups. Parallel algorithms based on parallel interactive moves can give good speedups, but can give rise to some degradation in quality of the solution with an increasing number of processors. The main advantage of all these algorithms is that it gives the best quality of the solution among all parallel algorithms for the placement problem. The disadvantage is that the runtimes of the algorithms are quite large even after parallelization compared to some of the other approaches.

Some researchers have proposed parallel algorithms for placement based on simulated evolution which is based on the analogy of the placement process to a biological evolutionary process. Related to the placement problem, the three steps in an iterative loop consist of evaluation, selection, and allocation. First, each cell in the population is subjected to evaluation, a phase that determines its normalized goodness, a figure of merit that reflects how well a cell is placed in its current position. Next, the selection procedure selects cells for replacement. For each cell separately, a trial is performed in which the cell's goodness determines its probability of survival in its current location. Finally, the allocation procedure removes all selected cells from the placement and a search is performed to find an improved location in the vicinity of its old position. In the parallel algorithm, a processor is allocated a region of the chip area and its set of cells and runs the simulated evolution algorithm procedures of evaluation, selection, and allocation on its cells. The parallel algorithm is suitable for both shared memory and distributed memory MIMD machines and gives good speedups. However, the quality of the solution is not as good as simulated annealing.

The parallel algorithms for placement based on genetic algorithms allocate a subset of population of cells to each processor, and each processor runs a genetic algorithm on its own population, and periodically exchanges subsets of populations with other processors. The algorithm gives reasonably good quality of solution and large speedups. The disadvantage of this algorithm is its large memory requirements, which need to store several hundred different placement solutions simultaneously.

The parallel algorithms for placement based on hierarchical decomposition use a divide and conquer method of solution by recursively decomposing the placement solution on the entire chip into subproblems of the placement of cells on smaller chip regions. The advantage of the parallel algorithm is that the solution quality does not degrade with an increasing number of processors. However, the disadvantage is that the speedups are not scalable with a large number of processors, since the topmost nodes in the task graph that execute on a few processors take the longest time to execute, and it takes a long time for all the processors to become busy doing useful work.

4 Parallel Routing

After a given set of cells is placed optimally on a layout, the wires need to be routed together. This routing is typically done in two steps called global routing and detailed routing. We will briefly describe some of the parallel routing algorithms in this section.

Parallel algorithms for detailed maze routing and detailed line routing have been proposed using different grid partitioning strategies. The maze routing algorithm consists of three phases: front wave expansion, path recovery, and sweeping. In the front wave expansion phase, a breadth-first search beginning at a start node S is performed, and consecutive nodes are labeled in increasing order until the target node T is reached. After the front wave expansion reaches the target cell T, the path recovery phase traces back the path from T to S, thereby identifying the wire path. The parallel algorithms are suitable for both shared memory and distributed memory MIMD machines. The advantage of the maze routing algorithm is that it is guaranteed to determine the shortest path for a given two-terminal net. Excellent speedups can be obtained on this algorithm since it naturally scales with the number of processors. The main disadvantage of this algorithm is its enormous memory requirement since we need to store the state of every grid point of the routing grid, which for large chips can be an extremely large number. The other disadvantages of this algorithm are (1) the order of routing the nets is very important to the solution quality and (2) multiple terminal nets are not easy to handle. The line routing algorithm minimizes memory and computation requirements at the cost of giving up on the routing quality.

Several parallel algorithms for channel routing have also been proposed. The advantage of these algorithms is that they handle the problem of net routing simultaneously. The parallel algorithm based on simulated annealing assigns tracks to different processors and moves nets among the tracks using an annealing algorithm to reduce the net overlaps. The algorithm gives good quality of the routing solution and excellent speedups; however, the sequential algorithm takes a very large amount of runtime. Hence even after running the algorithm on multiple processors, its runtime is slower compared to the best serial algorithms.

The parallel channel routing algorithm based on the greedy track assignment by columns partitions the channel region by column separators and routes each region independently. The choice of the columns for separating the regions is determined at the points of maximum channel density. This might give rise to load imbalances in the parallel algorithm. Also, by solving the channel routing problem independently, there is no guarantee that the nets will be assigned to the same tracks on the boundaries. When such a solution cannot be found, the algorithm uses the assumption of 45-degree bends at the boundaries to change tracks. Such a technology may not be allowed.

The parallel channel routing algorithm based on hierarchical decomposition creates two-by-N routing tasks recursively and solves each task on a processor. The algorithm has the advantage of giving good routing solutions, and the quality of the solution does not degrade with increasing number of processors. However, the disadvantage is that the task graph is a binary tree and, in the beginning, many processors remain idle since not enough tasks exist. Hence, the speedup using this algorithm is not very high.

Recently, a parallel algorithm for switchbox routing based on conflict resolution and iterative improvement has been proposed. The algorithm assigns each net to a different processor and routes each net independently. Conflicts in the routing are resolved and corrected by an iterative improvement phase. The parallel algorithm gives no degradation of quality with the increased number of processors and gives reasonably good speedups.

We will now describe some parallel algorithms for global routing. Global routing algorithms perform approximate routing of nets to routing resources. The parallel algorithms using graph search use a wave expansion algorithm similar to maze routing on a coarse routing grid to determine approximate routing assignments. The routing grid is partitioned among the processors using different methods. The advantage of the algorithm is that it gives very good routing quality and excellent speedups. The disadvantage of the algorithm is that the sequential algorithm takes a large computing time; hence, the runtimes on parallel processors are still quite large.

The parallel algorithm for global routing using iterative improvement considers a subset of possibilities for routing nets (L-shaped and a few Z-shaped layouts) and routes multiple wires simultaneously by assigning the routing task of a net to a different processor. The advantages are that the runtimes are quite low and excellent speedups are obtained. However, the routing quality is not as good, and the routing quality degrades slightly with increasing number of processors.

The parallel algorithm for global routing using hierarchical decomposition essentially divides the global routing problem on a grid into 2×2 subregions and solves the routing tasks recursively in a top-down manner. The advantages of the algorithm are that the runtimes are quite low and the routing quality is reasonably good. There is no degradation in routing quality with increasing number of processors. The speedups are reasonably good since, even though the task graph is in the form of a tree and at the top of the task graph there are not many tasks to keep the processors busy, the task graph expands very quickly and creates enough work for the later stages of the algorithm to effectively use all processors.

5 Parallel Layout Verification

Layout verification and analysis tools determines whether the polygons that represent different mask layers in the chip conform to the technology specifications and obtains circuit information about the circuit it implements. Design rule checkers (DRC) detect violations of rules that govern the technology in which the chip is to be fabricated. A circuit extractor determines the circuit implemented by the chip layout and estimates various electrical parameters, such as the resistances of lines, capacitances of nodes, and dimensions of devices. We will describe several parallel algorithms for DRC and extraction.

The parallel algorithm for design rule checking using area decomposition partitions a given chip layout by horizontal or vertical strips with an overlap of the maximum design rule interaction distance and performs DRC on each partition independently. The approach requires very little communication among the processors, and if each partition contains approximately the same number of rectangles, it gives good speedups. This approach is only suitable for flattened layouts, and is suitable for execution on shared memory and distributed memory MIMD multiprocessors.

The parallel algorithm for DRC using functional decomposition partitions the DRC tasks by the rules to be checked. Each rule can be potentially checked on the entire chip area by each processor. The approach is applicable to flattened and hierarchical circuits. However, the speedup of this approach is limited due to the fact that for many DRC operations we need to generate some common intermediate layers, which force a dependency on the application of the rules. This restricts the maximum parallelism available in the problem. The algorithm is suitable for shared memory and distributed memory MIMD multiprocessors.

A very efficient way to perform DRC in the sequential domain is to exploit hierarchical techniques. Parallel algorithms for hierarchical DRC essentially perform DRC tasks on individual cells in the hierarchical description of the circuit. The problem with this approach is that it is possible to get wide variances of load if we have cells of widely different sizes in a design. This can limit the speedups available in the problem. It is possible to use partitionable independent task scheduling algorithms to alleviate some of these problems. In this approach, a number of processors are assigned to solve the DRC problem of each large cell in parallel. The parallel hierarchical algorithm is suitable for shared memory and distributed memory MIMD multiprocessors.

The parallel algorithm for DRC using an edge based decomposition is suitable for SIMD massively parallel machines. The algorithm allocates each edge in the layout to a processor and performs DRC among processors owning adjacent edges. The approach gives good speedups, but is only applicable to flattened circuit designs.

Several researchers have developed numerous parallel algorithms for netlist and parameter extraction. The parallel netlist and parameter algorithms based on data decomposition partition the chip area into subregions and perform local extraction within each region, followed by a merge algorithm to perform globally extraction. These algorithms are suitable for flattened layouts and can execute on both shared memory and distributed memory MIMD multiprocessors. Excellent speedups can be obtained through this method.

The parallel algorithm for hierarchical netlist and parameter extraction combines the benefits of hierarchy and parallelism to speed up the extraction process. In the simplest parallel hierarchical algorithm, each cell in the hierarchy is extracted independently on a different processor. This approach may give poor speedups if the load balance is not uniform, for example, if there are cells that are very big. A more efficient parallel hierarchical algorithm has been proposed using the partitionable independent task scheduling method, in which a number of processors are assigned to solve the extraction problem of a hierarchical cell in parallel. This algorithm gives excellent speedups and combines the benefits of hierarchy and parallelism effectively. The algorithms are suitable for shared memory and distributed memory MIMD multiprocessors.

6 Parallel Circuit Simulation

Circuit simulators are used to verify circuit functionality and to obtain detailed timing information before the expensive and time-consuming fabrication process takes place. The behavior of an electrical circuit can be accurately described by a set of equations involving node voltages, currents, charges and fluxes. A variety of techniques for parallel circuit simulation has been proposed by researchers over the last few years. In this section, we will review several parallel approaches to circuit simulation based on: (1) direct methods, (2) nonlinear relaxation methods, and (3) waveform relaxation methods.

The parallel algorithms using direct methods are the most general and can be applied to any circuit. They are based on parallel solution techniques for linear algebraic systems. The direct methods constitute two parts: (1) the model evaluation part, which is relatively easy to parallelize, by assigning sets of devices to different processors and having each processor generate the model parameters of the matrices and vectors in parallel; and (2) the linear system solution phase, which is relatively hard to parallelize since these are sparse linear systems. There are some well-known parallel linear systems solution algorithms, and many of them are very efficient for parallel processing; but most are applicable to dense linear systems, or linear systems with well-defined sparse structures such as tridiagonal systems. For circuit simulation, since the systems are extremely sparse and irregular, parallelization is rather difficult. Several approaches such as element-based, row-based, and pivot based parallel approaches hasv been proposed. Element-based parallelism involves elemental operations such as multiplications and additions, and are difficult to exploit on MIMD multiprocessors since the grain-sizes of the parallel tasks are extremely small. Row-based parallelism involves operations which involve updating an entire row of a sparse matrix, and are easy to apply, but generates tasks of relatively medium grain. These methods do not generate appreciable speedups due to the small grain size of the computations. These methods are applicable mainly to shared memory MIMD multiprocessors. Pivot-based parallelism involves operations on independent pivots updating entire submatrices in parallel. The tasks are of relatively large grain and hence this approach is suitable for both shared memory and distributed memory MIMD multiprocessors. However, the issues with regard to identifying independent pivot-level parallelism are difficult. The amount of pivot-level parallelism is limited; hence such methods cannot give large speedups.

The major advantage of using direct methods is that they have proved to be accurate and reliable over a large class of circuits. There are, however, three major disadvantages. First, the sparse linear equation solution time grows faster than linearly with circuit size. Second, some waveform properties such as waveform latency and multirate behavior are hard to exploit. Hence, for large digital circuits that are based on MOS transistors, the direct methods constitute inherently slower approaches than relaxation-based methods, which have linear time complexity. Finally, direct methods are very hard to parallelize efficiently and get good speedups on large-scale parallel machines.

The nonlinear relaxation-based methods exploit the latency property of waveforms in MOS circuits. The parallel algorithms based on nonlinear relaxation solve the complete Newton-Raphson iteration of the nonlinear equation corresponding to one time-point for an entire subcircuit as a subtask. This approach has larger grain size compared to the grain sizes of tasks in parallel circuit simulation using the direct methods. It is therefore more suitable for parallel processing. This approach has been applied successfully to both shared memory and distributed memory MIMD multiprocessors, as well as SIMD massively parallel processors, with good speedups. Among the nonlinear relaxation approaches, the Gauss-Seidel relaxation methods are harder to parallelize, but converge faster than Gauss-Jacobi methods. For a very large number of processors, it is advantageous to use Gauss-Jacobi-based nonlinear relaxation methods since the advantages of more parallelism outweigh the increased time for convergence.

Waveform relaxation methods exploit the multirate property of waveforms and can give the least expensive approach to circuit simulation, faster than direct and nonlinear relaxation methods. Also, the parallelization of the circuit simulation problem is much easier using the waveform relaxation approach, since the grain size of tasks is even coarser than those obtained for the nonlinear relaxation approaches. A task in waveform relaxation constitutes the solution of complete voltage waveforms over an entire window for a subcircuit. This approach is therefore most suitable for large-scale parallelism. For distributed memory MIMD multiprocessors, for which the cost of sending messages is relatively high, but the cost of long messages is not significantly higher than the cost of short messages, the waveform relaxation approaches are the best. However, this approach has the disadvantage that its convergence properties depend a lot on the circuit characteristics and may take a large number of iterations to converge for some circuits.

7 Parallel Logic Simulation

Logic simulators are used in hardware design verification to verify the logical correctness and perform simple timing analysis of logic circuits. Logic simulators rely on abstract models of the functioning of a digital system. They yield discrete value outputs and crude timing information

We can classify the following types of parallelism in logic simulation. Functional parallelism deals with the exploitation of parallelism in logic simulation by assigning different functions associated with logic simulation to different processors. Input data parallelism assigns different sets of input patterns to each processor on the entire circuit. Circuit or model parallelism exploits the parallelism by assigning portions of a circuit to different processors and simulating the circuit partitions in parallel. In the compiled circuit parallel approach, the simulation code for the gates in the circuit partition assigned to each processor is compiled into machine code, and all the gates are evaluated at all time steps whether they are needed or not. In the event-driven circuit parallel approach, only those gates whose inputs have changed in the present time step are evaluated. There are two subapproaches within event-driven simulation. In the synchronous approach, we require that all the necessary nodes be evaluated at a given time point before moving on to the next time point. The problem with this approach is that the parallelism in logic simulation is limited to within one time point at a time. In the asynchronous approach, different portions of the circuit are allowed to evaluate up to different points in time. Such an approach has the maximum scope for exploiting parallelism. The problem with such an approach is in guaranteeing causality constraints. There are two basic ways to design such methods. In conservative methods, a parallel simulation is viewed as a series of sequential computations separated by synchronization messages. Suitable deadlock avoidance methods or deadlock detection and recovery methods have to be used. In optimistic methods, each processor is allowed to go ahead with its computations and to advance its local clock until a message is received that indicates an inconsistency. Then the processor state has to be rolled back to a point in time before the error.

While functional parallelism seems to be exploited best by special-purpose hardware, pattern parallel and circuit parallel approaches lend themselves shared memory and distributed memory MIMD multiprocessors. Typically, pattern parallel approaches take the least amount of recoding effort and can yield perfect linear speedup; however, their applicability is limited due to space and algorithm considerations. Pattern parallel algorithms are mainly restricted to combinational circuits with zero-delay gates.

Circuit parallel approaches are geared to handle large designs that need to be partitioned; they involve complex preprocessing and usually require complete redesign of the simulation system. Synchronous event-driven circuit parallel logic simulation gives reasonably good speedups on shared memory MIMD multiprocessors. These algorithms are not that complex.

The best speedups are obtained through asynchronous circuit parallel logic simulation algorithms. Both forms of asynchronous algorithms, conservative and optimistic, can give good speedups and are suitable for both shared memory and distributed memory MIMD multiprocessors. The algorithms are, however, much more complicated to implement than the simple synchronous parallel algorithms.

8 Parallel Test Generation and Fault Simulation

Once a digital circuit is designed and fabricated, the circuit needs to be tested for the presence of absence of physical defects or faults. Automatic test pattern generation (ATPG) deals with the problem of generating input patterns to test all faults automatically for a given circuit description. The objective of a fault simulation algorithm is to find the fraction of total faults (also referred to as the fault coverage) that are detected by a given set of input vectors. Often a fault simulation procedure is integrated within an automatic test pattern generation system. In this section, we discuss parallel algorithms for test generation and fault simulation of combinational and sequential circuits.

The parallel algorithms for test generation are based on fault parallelism, circuit parallelism, search parallelism, functional parallelism, and heuristic parallelism.

Fault parallel test generation refers to the evaluation of the test patterns for the given fault set in parallel. In this method, the fault set is divided equally among the processors. Each processor can now generate tests for its own fault set independently. This method has the advantage that the approach is independent of the actual ATPG and fault simulation algorithm used and is therefore the easiest to use. Excellent speedups can be achieved with a dynamic partitioning of the fault list using a multipass approach. In such an approach, each pass of test generation and fault simulation is performed using a fixed time limit per fault. The time limit is increased in successive passes, so the easier faults are detected in earlier passes and harder faults are detected in later passes. The main disadvantage of this approach is that the test set size increases significantly over serial algorithms. The approach is applicable to both combinational and sequential circuits.

Heuristic parallel test generation involves letting each processor use a different heuristic to guide the search for the same fault. But the main disadvantage of this method is that the parallelism is limited by the number of heuristics available for search (at most five to six). Also, by using different heuristics there is no guarantee that the search spaces are disjoint. This may lead to redundant work. Also, no improvement is possible if a fault remains undetectable for all the heuristics.

Search space parallel test generation assigns disjoint search spaces to multiple processors to search for a test vector or test vector sequence of a particular fault. Each processor can be allocated a portion of the search space, either statically or dynamically, and the search for the test vector can proceed concurrently. The advantage of this approach is that the really hard to detect faults are detected by a truly parallel test generation strategy; hence this approach reduces the runtime for particularly hard faults. The other advantage is that, when integrated with a fault simulation phase after each test generation phase, the test set size can be kept almost as small as the corresponding serial algorithm. The disadvantage of this approach is that not all faults are hard to detect, and many faults are detected with a few number of backtracks, for example 10 if good heuristics are used.

Functional parallel test generation uses a divide and conquer type approach by which a task is divided into a series of subtasks, all of which have to be completed. These subtasks can be completed in parallel if they do not access or modify any common variables. If they have common variables as they do in test generation, execution of the AND-parallel subtasks becomes complicated due to variable binding conflicts. The overheads due to variable binding conflicts may essentially reduce the AND-parallel tasks to sequential execution due to consistency checking, and the like. Hence this approach is not very attractive for parallel test generation.

In all the parallel approaches discussed so far, each processor keeps a copy of the entire circuit. For extremely large circuits, the memory of each processor may not be able to store the entire circuit. In a circuit decomposed approach, each processor keeps a partition of the circuit and performs various operations (such as backtracing, justification and implication) on its own subcircuit to satisfy various test generation objectives. While this approach is attractive for really large circuits, it is extremely hard to achieve efficient speedups with this method.

For test generation, it appears that the most successful approach to parallelization is to use a combination of fault parallelism (for the first phase to detect easy faults) and search parallelism to detect the hard faults. In any case, we should include fault simulation as part of the test generation process for efficiency.

We will now describe various approaches to parallel fault simulation: fault parallel, pattern parallel, circuit parallel, and pipelining.

We can use fault parallel fault simulation by partitioning a list of faults across the available processors and having each processor perform fault simulation on all the input vectors for each sublist. It is possible to obtain almost linear speedups in the case of fault parallel implementations. The problem is that for each partition of faults we have to perform a good machine simulation. Depending on the partitioning of the faults, the fault activity of each partition for a particular input vector may not be uniform across all partitions. This approach can also give poor load balance due to widely different fault dropping characteristics. One way to provide better load balance is to partition the fault list into smaller sized partitions, but then the good machine simulation (which represents wasted work) increases significantly.

Pattern parallel fault simulation involves partitioning the input patterns and letting each processor perform fault simulation on the entire fault list but for a subset of the input patterns in parallel. This approach gives excellent speedups; however, it is restricted to combinational circuits. The approach cannot be used in sequential circuits since the future behavior of a sequential circuit depends on all previous vectors. The approach can be used if we can identify independent subsets of patterns even in sequential circuits or if we use an iterative approach to solve the simulation problem until convergence.

Circuit parallel fault simulation involves partitioning a given circuit among processors and having all processors cooperate in the fault simulation of all faults and all inputs together. The advantage of this approach is that it is memory efficient and can handle very large circuits. However, the communication overheads of this approach are quite high, and it is difficult to get good speedups.

In summary, for parallel fault simulation, the fault parallel approach is a general approach applicable to both combinational and sequential circuits. For very large circuits, we should use circuit parallel approaches. For medium-sized combinational circuits, the pattern parallel approach is the best.

9 Parallel Logic Synthesis and Verification

Logic synthesis tools provide the automatic synthesis of near-optimal logic netlists, for which the goal is minimum delay, minimum area, or maximum testability. Logic verification tools compare the logic design of integrated circuits at different levels to make sure that, in the synthesis process, no logic errors have been introduced. In this section, we review parallel algorithms for logic synthesis and verification.

The parallel algorithm for two-level logic synthesis is based on the well-known ESPRESSO approach. The ESPRESSO algorithm iteratively performs the operations of reducing certain cubes, expanding other cubes, and removing redundant cubes of a multiple output Boolean function, until the circuit shows no improvement. The parallel algorithm for two level synthesis involves parallelizing each procedure within the ESPRESSO sequential algorithm: the complementation of cubes, the reduction of cubes, the expansion of cubes, and elimination of redundant cubes. To preserve the correctness of the cover of the circuit, several interesting heuristics have been developed to prevent simultaneous elimination of cubes during parallel reduction, and creation of multiple identical cubes during parallel expansion of cubes. Good speedups have been reported using the parallel algorithm with slight degradation in circuit quality.

The parallel algorithm for logic synthesis using the partitioning approach initially partitions a given circuit into as many partitions as there are processors. Subsequently, it performs bipartitioning, merging and resynthesis of each partition with different pairings of processors in an iterative manner. The approach has three advantages. First, it is possible to get excellent speedups by partitioning the circuit into as many parts as there are processors. Second, the approach is independent of the logic synthesis algorithm used. Hence it is possible to use the best sequential algorithms for logic synthesis available at any time. Finally, the approach minimizes the memory requirements needed in the synthesis algorithm. Hence it is possible to perform logic synthesis on very large circuits. The main disadvantage of this approach is that the quality of the synthesized circuit deteriorates significantly with an increasing number of partitions.

The parallel algorithm for logic synthesis using the MIS approach involves parallelizing each synthesis procedure within the MIS system. Specifically, parallel algorithms have been developed for kernel and cube generation, kernel and cube extraction, resubstitution, and node simplification. A key feature of the algorithm is in the use of partitioning of the circuit for workload distribution, while the logic synthesis is performed on the entire network. Consistency is maintained among concurrent updates to the network due to various transformations by using the notion of version numbering on the nodes in the circuit. Excellent speedups have been reported using the parallel algorithm with very little degradation in circuit quality. The MIS approach is one of the best-known methods of multilevel logic synthesis, and is known to produce circuits of excellent quality, and is also very efficient in time and memory requirements. Hence the parallel algorithm for logic synthesis using the MIS approach has very high practical value.

A parallel algorithm for logic synthesis using the transduction method has also been proposed. The advantage of the transduction approach is that it can produce circuits of excellent quality, but the disadvantage is that it has large computing and memory requirements. The parallel algorithm involves parallelizing each procedure used in the transduction method. Specifically, parallel algorithms have been developed for the evaluation of output and permissible functions and for various transformations such as pruning, gate substitution, and gate merging. Good speedups were reported using the parallel algorithm with slight degradation in circuit quality.

We finally describe two parallel algorithms for logic verification. The first algorithm is based on an implicit enumeration approach, and the algorithm uses a recursive decomposition of the search space of verification for all possible input combinations implicitly. The dynamic decomposition of the search space gives excellent load balance and speedups.

The second parallel algorithm for logic verification is based on tautology checking. Again, the algorithm is based on a recursive divide-and-conquer decomposition of the search space of input combinations. The parallel algorithm gives excellent speedups.

10 Commercial Parallel CAD Products

While the previous sections outlined research in parallel CAD applications, parallel processing for VLSI CAD has become a reality in industry as well. Hardware vendors such as Sun Microsystems, Solbourne, Hewlett-Packard and Digital have already announced products with multiple CPUs in a single workstation. Several software CAD vendors have already announced products that exploit parallel machines.

Mentor Graphics has a product called PARADE, which performs cell placement in parallel. Mentor Graphics also has a parallel implementation of design rule checking and extraction called CHECKMATE.

Cadence Design Systems has a parallel design rule checker called PDRACULA. Cadence Design Systems also has a product called VERIFAULT that runs fault simulation in parallel on a network of workstations.

Crosscheck Technology has a parallel test generator called the AIDA-II for combinational circuits. Sunrise Test Systems recently announced a parallel test generator calle TGEN on a network of workstations.

Chronologic Simulations has announced a parallel VHDL simulator which runs on multiprocessor SUN workstations.

Silvaco markets a product for lithography simulation called VIRTUAL WAFER FAB and, for 3D device simulation, one called THUNDER on massively parallel MasPar MP-1 systems.

In the future, numerous other parallel implementations of VLSI CAD tools will become available. This is especially true in view of the multiprocessor workstations that are now available from various vendors.

11 Future Directions of Parallel CAD

In view of the increasing complexity of VLSI circuits of the future, the requirements on VLSI CAD tools will continuously increase. Parallel processing for CAD applications is becoming gradually recognized as a popular vehicle to support the increasing computing requirements of future CAD tools. Recent research on parallel CAD applications have been reported for a wide variety of applications such as placement, routing, layout verification, circuit simulation, test generation, fault simulation, and logic synthesis. This paper has surveyed many of these parallel algorithms.

A major limitation with almost all previous work is that the parallel algorithms have been targeted to run on specific machines like an Intel iPSC hypercube-based message-passing distributed memory multicomputer or an Encore Multimax shared memory multiprocessor. Such work, although interesting, is not usable by the rest of the VLSI CAD community since the algorithms are not portable to other machines.

A second serious problem also presents itself in the design of parallel algorithms. The software development cycle for parallel algorithms is considerably longer than for sequential algorithms. This has two important implications. The first is they are considerably more costly to develop than sequential algorithms. This is exacerbated by the lack of portability across parallel machines. The second implication is a more pragmatic. Given the fast pace of progress in the development and improvement of sequential algorithms for CAD applications for a given application, sequential algorithms frequently outperform parallel algorithms due to the longer development time of the latter.

Finally, and most importantly, all the parallel CAD algorithms that have been developed so far are mainly academic tools solving simple versions of complex real-world problems, and do not have the industrial strength required of the corresponding conventional tools running on uniprocessor computers. For example, the parallel placement and routing programs only work for regular-sized cells, and can handle two layers of interconnect, whereas in the real world, we have to deal with up to seven layers of interconnect, with complex interactions between the layers. In order for parallel CAD algorithms to really become practical, it should be possible to readily leverage off the recent advances in the sequential algorithms.

Two important questions that need to be addressed in the development of parallel algorithms for CAD are how we can design parallel algorithms for CAD that are truly portable across parallel machines and how we can exploit good sequential algorithms in the design of parallel algorithms for CAD. The ProperCAD project (**P**ortable object-oriented **p**arallel environment for **CAD**) tries to address these goals. The approach to parallelism taken in this research is the development of a C++ runtime library that provides, simultaneously, a high-level, object-oriented abstraction to application programmers and an underlying runtime implementation that utilizes to the maximum extent and highest efficiency possible the resources available to the program [2]. Parallel CAD applications are developed on top of the standard parallel programming interface implemented as a C++ class library. This approach facilitates the incremental application of parallel constructs to develop a parallel application while maintaining compatibility and code-sharing with the serial implementation. An overview of the ProperCAD-II project is shown in Figure 1. Successful implementations of several CAD applications have been reported on the ProperCAD environment: *ProperHITEC*, a test generator for sequential and combinational circuits, *ProperPLACE*, placement for combinational circuits, *ProperSYN*, synthesis for combinational circuits.

It is our opinion that the future of parallel CAD lies in machine independent parallel CAD algorithms with well-defined interfaces to sequential algorithms so that latest advances in sequential algorithms can be readily incorporated into parallel CAD algorithms. A problem that is commonly faced in the design of efficient parallel algorithms for CAD applications such as placement is that every year better and better sequential heuristics are being discovered, which improve on previous methods in terms of quality of layouts produced, better objective functions, and runtimes. Newer sequential placement algorithms handle better and improved

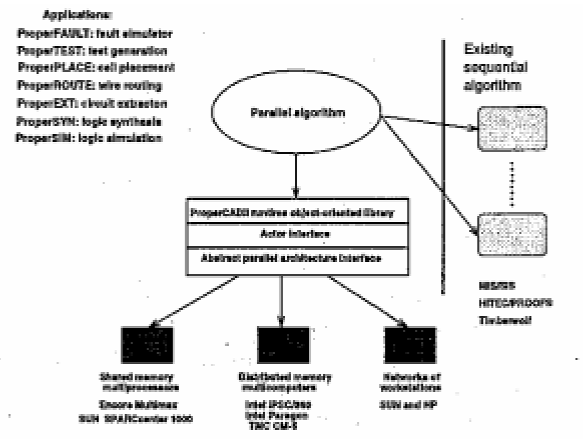
Applications:
ProperFAULT: fault simulator
ProperTEST: test generation
ProperPLACE: cell placement
ProperROUTE: wire routing
ProperEXT: circuit extraction
ProperSYN: logic synthesis
ProperSIM: logic simulation
Parallel algorithm
Existing sequential algorithm
ProperCAD II runtime object-oriented library
Actor interface
Abstract parallel architecture interface
MIS/SIS
HITEC/PROOFS
Timberwolf
Shared memory multiprocessors
Encore Multimax
SUN SPARCcenter 1000
Distributed memory multicomputers
Intel iPSC/860
Intel Paragon
TMC CM-5
Networks of workstations
SUN and HP

Figure 1: Overview of the ProperCAD-II project

objective functions, that is, performance driven layout. Unless the parallel algorithms can be designed such that these newer efficient techniques can be rapidly incorporated into the parallel implementations, the latter will become rapidly obsolete in terms of practical use.

We believe that for the application of parallel processing to CAD to be successful, all of the above objectives need to be met. It is important for industry to work together with academia to develop such industrial quality tools.

References

[1] P. Banerjee, *Parallel Algorithms for VLSI Computer-aided Design Applications.* Englewoods-Cliffs, NJ: Prentice Hall, 1994.

[2] S. Parkes, J. Chandy, and P. Banerjee, "A Library-Based Approach to Portable, Parallel, Object-Oriented Programming: Interface, Implementation, and Application," *Proc. ACM Supercomputing 94 Conf.*, Nov. 1994.