A Timing Optimization Method Based on Clock Skew Scheduling and Partitioning in a Parallel Computing Environment

Baris Taskin
Drexel University
Philadelphia, PA 19104
E-mail: taskin@coe.drexel.edu

Ivan S. Kourtev
University of Pittsburgh
Pittsburgh, PA 15261
E-mail: ivan@engr.pitt.edu

***Abstract*— This paper describes the implementation of a heuristic method to perform non-zero clock skew scheduling of digital VLSI circuits in a parallel computing environment. In the proposed method, circuit partitions that have low number of timing paths between partitions are formed. Clock skew scheduling is applied independently to each partition—sequentially or in parallel on a computing cluster—and results are iteratively merged. The scalability of the proposed method is superior compared to conventional non-zero clock skew scheduling techniques due to the reduction of analyzed circuit sizes (partition sizes) at each iteration step and the potential to parallelize the analyses of these partitions. It is demonstrated that after only the first iteration step of the proposed method, feasible clock schedules for 65% of the ISCAS'89 benchmark circuits are computed. For these circuits, average speedups of 2.1X and 2.6X are observed for sequential and parallel application of clock skew scheduling to partitions, respectively.**

I. Introduction

Mainstream integrated circuit design flow is normally tuned for zero clock skew circuit design. Zero clock skew design is popular due to its relative simplicity in various stages of the design flow, compared to non-zero clock skew circuit design. Nevertheless, non-zero clock skew design (scheduling) permits improved circuit performances, and is desirable in high-performance circuit design [1]. Various clock skew scheduling methods targeting edge-triggered and level-sensitive circuits have been proposed. The proposed techniques range from techniques that lead to an optimal solution [2], [3] to those that lead to a sub-optimal feasible solution [4], from those that employ accurate models [2], [5] to those that employ approximate models [6], from those that use graph theoretic approaches [7], [8] to those that employ linear or quadratic programming techniques [9], [10].

A common bottleneck for all these approaches is scalability. Scalability is an issue because of the interdependence of system timing paths of an entire circuit for clock skew scheduling. Unlike prevailing static timing analysis methods where a selected subset of timing paths (critical timing paths) must be investigated, the entire set of timing paths must be investigated for a full-scale application of clock skew scheduling. Such analysis is typically computationally intensive for very large-scale circuits. The computational intensity manifests itself in two main forms: (1) Hardship in or failure of path identification and enumeration, (2) Excessive run times for conventional clock skew scheduling application techniques.

In this paper, a heuristic method that improves the scalability of clock skew scheduling through partitioning and parallelization is proposed. A circuit is divided into multiple partitions such that clock skew scheduling can be applied independently to each partition. Furthermore, the application of clock skew scheduling (to partitions) is parallelized for improved scalability. An iterative solution method is proposed to merge the results and verify the compatibility (feasibility) of the solutions for each partition.

The rest of the paper is organized as follows. In Section II, the conventional clock skew scheduling algorithm used in this work is reviewed. In Section III, the motivation for the presented research is described. In Section IV, the partitioning and parallelization steps of the proposed method are described. In Section V, experimental results are presented. The paper is finalized in Section VI.

II. Clock Skew Scheduling

Synchronous circuits are built of local data paths. A local data path [1] is formed by two sequentially adjacent registers and a combinational logic block between them. The timing analysis of a synchronous circuit is performed by satisfying the relevant timing constraints for each local data path. A sample local data path between sequentially adjacent registers R_i and R_f is shown in Figure 1. The minimum and maximum propagation delays on the combinational logic block are denoted by D^{if}_{Pm} and D^{if}_{PM}, respectively. The internal delays of a register, not shown in Figure 1, are also defined: The minimum and maximum clock-to-output delays of R_i are denoted by D^{i}_{CQm}

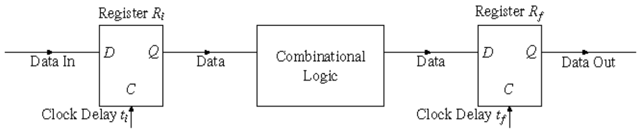

Fig. 1. A local data path.

and D^{i}_{CQM}, respectively. The setup and hold times of R_f are denoted by S_f and H_f, respectively.

Due to process parameter variations or by design, clock (signal) delays at each synchronous component of a circuit may be nonidentical. Clock skew is defined as $T^{if}_{skew} = t_i - t_f$, where t_i and t_f represent the clock delays to registers R_i and R_f of a local data path, respectively. Clock skew scheduling is the process of computing the optimal clock delays to each register in order to meet a particular design objective. Popular design objectives are maximizing the operating frequency T and improving the tolerance of the circuit to secondary order effects. The clock skew scheduling method of interest in this work, shown in Table I and introduced in [2], targets maximizing the operating frequency of an edge-triggered circuit.

TABLE I
CLOCK SKEW SCHEDULING METHOD.

	LP Problem [2]
min	T
s.t.	$T_{skew}(i,f) \leq T - D^{if}_{PM} - D^{i}_{CQM} - S_f$
	$T_{skew}(i,f) \geq -D^{if}_{Pm} - D^{i}_{CQm} + H_f$

III. MOTIVATION FOR RESEARCH

In state-of-the-art VLSI circuits with high logic depth and design complexity, identification of timing paths and path enumeration cannot always be completed within reasonable time and computation resources. A heuristic partitioning method is proposed in this work in order to remedy this shortcoming. In the proposed method, very long paths are split with a cut and a level-sensitive latch operating in the transparent phase is inserted on the cut. The transparent phase latch has no effect on the functionality of the circuit because the data signal immediately propagates through the latch. This latch, however, shortens the logic depth of the original path.

Also, the linear programming (LP) clock skew scheduling problem formulated for a VLSI circuit (Table I) can be very large. If solvable, the run times of such large LP problems are typically manageable within the long design cycles of integrated circuits. However, some of these problems may not be solvable at all with common computing resources. In several industry applications, for instance, LP model problems for the clock skew scheduling of large-scale circuits exceed the practical limits of industrial strength computing resources [11]. In the method proposed in this paper, clock skew scheduling is applied to (smaller) circuit partitions in a parallel computing cluster, proving this process feasible.

Partitioning for improved scalability of clock skew scheduling is pursued partially due to an intertwined relationship to the resonant rotary clocking [12], [13] research. Resonant rotary clocking technology [12], [13] is a next-generation clocking technology that inherently supports—and requires the use of—non-zero clock skew scheduling. The implementation of high-speed digital circuits synchronized by the resonant rotary clock technology entail partitioning due to the mesh structure of the rotary clock distribution network. The details of such synchronization will not be explained in this paper. Nevertheless, this relationship is mentioned here due to comprising a driving motivation for the presented work.

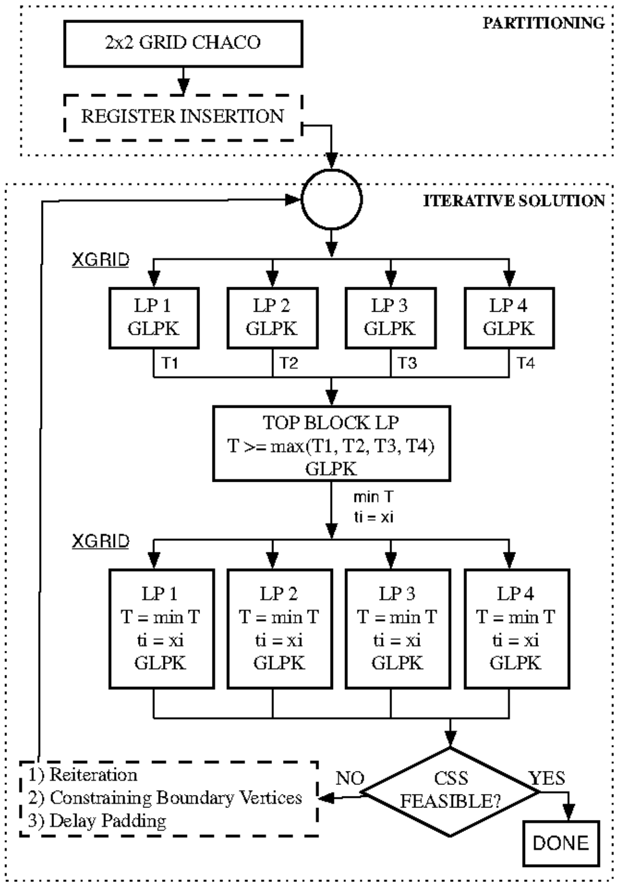

Fig. 2. Clock skew scheduling with partitioning method.

IV. HEURISTIC METHOD

The proposed method entails an initial partitioning stage and an iterative solution stage as illustrated in Figure 2. In the *partitioning stage*, instead of a traditional path-based or a net-based partitioning methodology, a fine-tuned, timing-driven partitioning methodology is used. In this partitioning methodology, partitioning criteria are selected such that partitions which are amenable to clock skew scheduling are generated. In the *iterative solution stage*, data paths that are local to each partition are identified and corresponding timing constraints are included in the clock skew scheduling problem for that partition. Timing constraints of local data paths which span different partitions are included in the clock skew scheduling problem of the "top block". The LP problem formulation for the application of clock skew scheduling in Table I is used to formulate the clock skew scheduling problem of each partition and the top block. The clock skew scheduling problems of each partition are independent of each other, so these analyses are parallelized. However, the solutions of each partition and the top block are dependent due to the timing paths between multiple partitions. Alternative methods of iteration are proposed to solve for the dependencies. The stages of this heuristic method is detailed in the following subsections.

A. Partitioning with Chaco

In practical implementation of partitioning, the partitioning tool *Chaco* [14] is used. Among the multiple criteria of partitioning for clock skew scheduling are the weight, number and location of the cuts amongst partitions, the weight of each partition, the relative mapping of sequentially-adjacent registers to partitions and the number of internal vertices per partition. Chaco tracks the quality of these partitioning criteria with user-defined priorities. In order to generate partitions amenable to clock skew scheduling, the number of cuts between partitions must be minimal and the number of internal vertices (vertices that do not have edges between partitions) must be maximal. Depending on particular design budgets, the priority of the criteria, or the weights of particular nets or vertices can be fine tuned.

For smaller circuits where the circuit graph [1] can be generated easily, partitioning is applied on the generated circuit graph. For larger circuits, where the circuit graph cannot be generated (or takes excessive time), partitioning is applied on the gate-level design. In this case, the partitioner is tuned such that registered-input, registered-output partitions are generated.

The partitioning tool is operated with different priorities assigned to the partitioning criteria. Experimentally, a *balanced* priority assignment between minimizing the total cut weight and maximizing the number of internal vertices is found sufficiently effective.

B. Clock Skew Scheduling of Partitions

After partitioning, clock skew scheduling problems are generated for the n circuit partitions, $(LP_1, LP_2, \ldots, LP_n)$. Each partition LP_i is solved (sequentially or in parallel) in order to compute the minimum clock period permitted by that partition. The maximum of the minimum clock periods reported from each partition LP is selected as the common clock period at which all the partitions are operable.

Next, the maximum of the minimum clock periods computed for the partitions is used to further constrain the clock frequency of top block LP. A constraint in the form

$$T \geq \max(T_1, T_2, \ldots, T_n) \tag{1}$$

is added to the top block LP, where $T_1, T_2, \ldots, T_n$ denote the minimum clock periods computed for partitions $LP_1, LP_2, \ldots, LP_n$, respectively (see Figure 2). If the top block LP problem is less constraining on the minimum clock period than the partition LP problems, then the maximum of the minimum clock periods of the partition LP problems is selected as the common clock period (for partitions and the top block). Otherwise, the minimum clock period computed for the top block is selected as the common clock period.

The top block LP problem is solved *after* the partition LP problems are solved, because the top block has *all* boundary vertices implied in its constraints. Each partition LP problem only has a fraction of the boundary vertices implied in their constraints. The solution of the clock delays to *all* boundary vertices, as computed by the partition LPs and the top block LP problem, are matched in order to verify the validity of the computed minimum clock period. In order to match these clock delays of boundary vertices, the solutions computed for the top block LP problem are enforced on the partition LP problems with equalities such as:

$$t_i = x_i, \tag{2}$$

where the clock delay t_i computed for register R_i in the top block LP problem is x_i time units (see Figure 2). If the partition LP problems return feasibility, the computation is complete.

There are two points to note here. First, note that, the presented heuristic method does not guarantee a feasible solution. The following alternative approaches are proposed to solve for he infeasible cases:

- `Reiteration:` Iterations are performed on infeasible subproblems to search for a feasible answer. The clock delays whose values are changed from the optimal solution of the top block LP are tracked such that the feasibility of the remaining LP problems are not violated.
- `Constraining boundary vertices:` The clock delays of all boundary registers are set to an identical value. Synchronous circuits are typically built to operate at zero clock skew, thereby, this procedure guarantees proper circuit operation. A similar clock delay restriction procedure is applied to the timing of Intellectual Property (IP) blocks in [3]. In the experiments performed on ISCAS'89 benchmark circuits for IP blocks, restricting the clock delays of boundary vertices caused an average 27% improvement of conventional clock skew scheduling drop to 24% [3].
- `Delay padding:` The minimum D_{Pm}^{if} and maximum D_{PM}^{if} data propagation delays of a local data path are formulated with additional slack variables S_m^{if} and S_M^{if}, respectively. In the LP problem solution, slack values reported on each local data path are the amounts of delay that must be inserted along these logic paths. The clock delays of the boundary registers are fixed prior to the solution, such that, the solutions of the remaining LP problems are not violated.

The second point to note is that, without register insertion, the partition LP problems and the top block LP problem are subproblems of the original LP problem. As the solution of one of the subproblems (the top block LP problem in this case) is enforced on the remaining LP problems, the convex solution space of the original problem is not violated. Thus, if a feasible solution is found *without register insertion and without iterations*, it is optimal. If register insertion is performed or the alternative reiteration techniques described above are used, however, suboptimal solutions can be obtained or the convex solution space can be disturbed leading to suboptimal solutions.

V. EXPERIMENTAL RESULTS

The main focus in experimentation is demonstrating the *feasibility* of the application of clock skew scheduling with partitioning (sequentially and in parallel). Towards this goal, results for only *the first iteration step* are reported. In this paper, the alternative iteration methods (Section IV-B) are proposed as potential directions for circuit designers—they are not analyzed experimentally.

A. Clock Skew Scheduling Results

Experiments are performed on an ISCAS'89 suite of benchmark circuits and an industrial circuit `industrial1`. The timing information for ISCAS'89 circuits is generated with a pre-determined algorithm, in which the fanout, size and type of logic gates are considered. A single phase clock signal with a 50% duty cycle is selected for synchronization. The internal register delays are assumed negligible. The experiments are performed on an Xgrid [15] parallel computing cluster built with four (4) PowerMac computers with dual G5 1.8 GHz microprocessors (only one processor is used in clients) and 3 GB RAM. A constant partition size of four (4) is selected. The simplex optimizer of the GNU LP solver GLPK (version 4.8) [16] is used to solve the LP problems. The results are presented on Table II.

In Table II, the run times are reported in order to demonstrate the speedups achievable through partitioning and parallel application of clock skew scheduling. Run times of the conventional method are denoted by t_{conven}, the run times of the sequential solution of partitions method are denoted by t_{sequen} and the run times of the parallel solution of partitions method are denoted by t_{paral}. The observed run times t_{sequen} and t_{paral} record speedups over conventional clock skew scheduling application due to partitioning only, and, partitioning and parallelization, respectively. The feasibility of each solution (*Feas?*) is shown in Table II. Each feasible solution indicates an optimal minimum clock period, which provides approximately 30% shorter clock periods on average (see [3]), over conventional zero clock skew, edge-triggered circuits. Note that larger ISCAS'89 circuits have around 1.5k registers and 20k paths while `industrial1` has approximately 14k registers and 3.7M paths.

It is observed from Table II that t_{paral} is consistently and significantly (especially for large scale circuits) superior to t_{sequen} and t_{conven}. Similarly, t_{sequen} is consistently superior to t_{conven}. The run time improvement from t_{conven} to t_{sequen} and from t_{conven} to t_{paral} are listed under RTI_{sequen} and RTI_{paral}, respectively. The improvements are computed with the formula $[100(t_{old} - t_{new})/t_{old}]$. On the ISCAS'89 benchmark circuits, the average run time improvement via partitioning (RTI_{sequen}) is 25%. The average run time improvement via partitioning and parallelization RTI_{paral} is 28%. The circuits, for which the method is infeasible, are not considered in the computations of the average improvement. Overall, the application of clock skew scheduling to partitions is feasible for 28 (65%) of the total 43 circuits, whereas this method is not applicable to the remaining 15 circuits (35%). For these 15 circuits, the alternative methods described in Section IV-B can be used.

TABLE II
CLOCK SKEW SCHEDULING RESULTS ON FOUR (4) PARTITIONS.

Circuit	**Run Time CSS (sec)**			**RTI (%)**		**Feas?**
Name	t_{conven}	t_{sequen}	t_{paral}	RTI_{sequen}	RTI_{paral}	y/n
s27	0	0	0	0	0	y
s208.1	0	0	0	0	0	y
s298	0	0	0	0	0	y
s344	0	0	0	0	0	y
s349	0	0	0	0	0	y
s382	0	0	0	0	0	y
s386	0	0	0	0	0	y
s400	0	0	0	0	0	y
s420.1	0	0	0	0	0	n
s444	0	0	0	0	0	y
s510	0	0	0	0	0	y
s526	0	0	0	0	0	y
s526n	0	0	0	0	0	y
s641	0	0	0	0	0	n
s713	0	0	0	0	0	n
s820	1	1	1	0	0	y
s832	0	0	0	0	0	y
s838.1	2	0	0	0	100	n
s938	1	1	1	0	0	n
s953	0	0	0	0	0	y
s967	0	0	0	0	0	y
s991	0	0	0	0	0	y
s1196	0	0	0	0	0	n
s1238	0	0	0	0	0	n
s1423	21	6	3	71	86	y
s1488	0	0	0	0	0	y
s1494	0	0	0	0	0	y
s1512	1	0	0	100	100	y
s3271	4	2	1	50	75	n
s3330	2	2	1	0	50	n
s3384	22	4	3	82	86	y
s4863	2	0	0	100	100	y
s5378	9	5	2	44	78	n
s6669	33	10	7	30	79	n
s9234	52	15	8	71	85	n
s9234.1	47	12	5	74	89	y
s13207	86	17	10	80	88	y
s15850	3545	735	447	79	87	n
s15850.1	1358	156	110	89	92	y
s35932	101	38	13	62	87	n
s38417	7707	3780	1845	51	76	y
s38584	1394	749	339	46	76	y
Industrial1	n/a	34680	25680	n/a	n/a	n
Average				25	28	

B. Overall Improvement Results

The run times of a complete non-zero clock skew timing tool are also analyzed to profile the speedups gained in overall program execution due to partitioning and parallelization.

On ISCAS'89 circuits, an average of 2.1X speedup is observed in run time due to partitioning (without parallelization). If the partitioned LP problems are solved in parallel, the average speedup is 2.6X. It is intuitive that as the size of a circuit increases, the clock skew scheduling step of a complete tool, which is the fraction of the task that is enhanced with partitioning and parallelization, increases as well. So, for larger

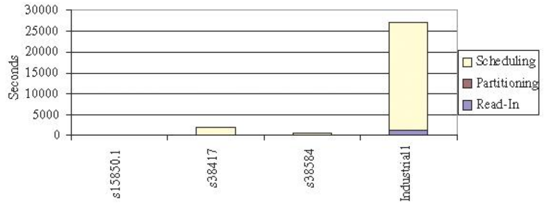

Fig. 3. Total run times with Xgrid on large circuits.

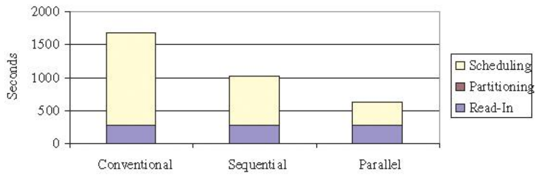

Fig. 4. Run time breakdown for s38584.

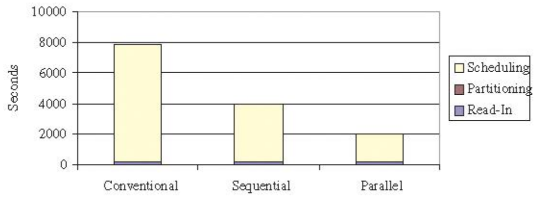

Fig. 5. Run time breakdown for s38417.

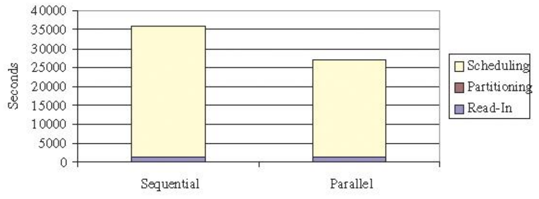

Fig. 6. Run time breakdown for industrial1.

size circuits, higher speedups are expected through partitioning and parallelization.

The complete tool operation has three main steps, *Read-in*, *Partitioning* and *Scheduling*. The scheduling step (in this work) consists of the first step of iterative application of the clock skew scheduling method. Figure 3 illustrates the relative run times of each step for three larger ISCAS'89 circuits and the industrial circuit `industrial1` for the parallel application of clock skew scheduling.

The breakdown of run times to the three steps of a complete tool is shown for the three largest circuits, `s38584`, `s38417` and `Industrial1` in Figures 4, 5 and 6 respectively. The run times for three application methods—conventional, sequential and parallel application—are shown. The run times for each step is shown with color codes, listed as read-in, partitioning and scheduling steps from bottom to top for each data bar. Partitioning step is not required in the conventional application method, thus is not shown in Figures 4, 5 and 6 for the "conventional" application methods. Even for methods where partitioning is necessary, the partitioning stages of the run time bars are indistinguishable, because the run times for the partitioning process are relatively very short.

The run times of the read-in and partitioning (where applied) steps are identical in all three application methods. The run time of the clock skew scheduling step is improved through partitioning and application of clock skew scheduling in parallel. These improvements lead to an overall speedup.

VI. Conclusions

In this paper, a heuristic methodology for the application of clock skew scheduling to VLSI circuits in a parallel computing environment is described. Partitioning is used to efficiently process larger circuits by generating smaller partitions for faster timing analysis. The method is applied to four-partition ISCAS'89 circuit systems on a parallel cluster of four computing clients, leading to speedups of 2.6X on average.

References

[1] I. S. Kourtev and E. G. Friedman, *Timing Optimization Through Clock Skew Scheduling*. Kluwer Academic Publishers, 2000.

[2] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. C–39, no. 7, pp. 945–951, July 1990.

[3] B. Taskin and I. S. Kourtev, "Linearization of the timing analysis and optimization of level-sensitive digital synchronous circuits," *IEEE Transantions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 1, pp. 12–27, January 2004.

[4] K. Ravindran, A. Kuehlmann, and E. Sentovich, "Multi-domain clock skew scheduling," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, November 2003, pp. 801–808.

[5] S. Held, B. Korte, J. Massberg, M. Ringe, and J. Vygen, "Clock scheduling and clocktree construction for high performance asics," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 232–239, November 2003.

[6] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "*checkT$_c$* and *minT$_c$* : Timing verification and optimal clocking of synchronous digital circuits," *Proceedings of the IEEE/ACM International Conference on Computer–Aided Design*, pp. 552–555, November 1990.

[7] R. B. Deokar and S. S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, May-June 1994, pp. 407–410.

[8] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Graph algorithms for clock schedule optimization," *Proceedings of the IEEE/ACM International Conference on Computer–Aided Design*, pp. 132–136, November 1992.

[9] I. S. Kourtev and E. G. Friedman, "A quadratic programming approach to clock skew scheduling for reduced sensitivity to process parameter variations," in *Proceedings of the 1999 IEEE ASIC/SOC Conference*, 1999.

[10] R. Mader, E. G. Friedman, A. Litman, and I. S. Kourtev, "Large scale clock skew scheduling techniques for improved reliability of digital synchronous circuits," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, May 2002, pp. 357–360.

[11] Ilog, "Private communication," 2004, iLOG Inc.

[12] J. Wood, T. Edwards, and S. Lipa, "Rotary traveling-wave oscillator arrays: a new clock technology," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1654–1665, November 2001.

[13] J. Wood, "Electronic circuitry," United States Patent Application Number 20030128075, July 2003.

[14] B. Hendrickson and R. Leland, "The chaco user's guide: Version 2.0," Sandia National Laboratories, Albuquerque, NM, Tech. Rep., Jul 1995.

[15] *Xgrid Guide*, Apple Inc., Advanced Computing Group, 2004.

[16] *GLPK (GNU Linear Programming Kit)*, Free Software Foundation (FSF), http://www.gnu.org/software/glpk/glpk.html, 2005, version 4.8.