マルチプロセッサの低消費電力化のための クロックON/OFFスケジューリング

横丸 敏彦, 高橋 篤司, 梶谷 洋司

東京工業大学 工学部 電気・電子工学科
〒152-8552 東京都 目黒区 大岡山 2-12-1
Tel : (03)5734-3572, Fax : (03)5734-2902
E-mail : {yokomaru, atushi, kajitani}@ss.titech.ac.jp

あらまし

多数のプロセッサからなるクロック駆動システムにおいて，プロセッサ集合をブロックと呼ぶ部分集合に分割しブロック単位でクロックの供給/非供給を自由に制御できるというモデルを考える．プロセッサはクロック供給時に主要な電力を消費するという仮定でプロセッサで消費される全電力の削減を目的とするジョブスケジュール問題を定式化する．そしてブロックごとにジョブの逐次的な最適化を適用するという比較的簡単で実装しやすいヒューリスティックアルゴリズム BF-ASAP (Best-Fit/As Soon As Possible) を考案し，100個のプロセッサが10個にブロック化されたランダムに生成された問題でおよそ30%の削減が観察されることを示し，この設計方式の将来性を示す．

キーワード　低消費電力，ジョブスケジューリング，NP完全，Best-Fit，ASAP

A Clock ON/OFF Scheduling for Low Power Multi-Processor Design

Toshihiko YOKOMARU, Atsushi TAKAHASHI and Yoji KAJITANI

Dept. of Electrical and Electronic Engrg., Tokyo Inst. of Tech.
Ookayama, Meguro, Tokyo, 152-8552 Japan
Tel : +81-3-5734-3572, Fax : +81-3-5734-2902
E-mail : {yokomaru, atushi, kajitani}@ss.titech.ac.jp

Abstract

On a clock-driven system with multiple processors, we consider the model that the set of processors is clustered into blocks and timing of clock-supply is controlled at block level. By the assumption that the power is mainly consumed by processors when they are supplied with clocks, we formulate a job scheduling problem to make the power consumption of the system small. We propose an easily implementable heuristic algorithm BF-ASAP (Best-Fit/As Soon As Possible) which optimizes each job sequentially in each block. We show about 30% power reduction on randomly generated instances in which 100 processors are clustered into 10 blocks, to show the method being promising.

key words　Low Power, Job Scheduling, NP-complete, Best-Fit, ASAP

1 Introduction

Reduction of power consumption of electric devices is required more and more recently because they are getting small and portable. There have been various approaches to improve power consumption in ideas such as to use low-voltage, multi-voltage, to reduce switching number of gates, to optimize gate sizing, to change the order of input terminals[6] and to control the power consumed by clock-supply. In [3], there is a report that the power consumed by clock-supply amounts to 30% of the power consumed in the whole system. Many literatures and experts are suggesting that the effective approach is to reduce the power consumed by clock-supply.

In this paper, we consider to reduce the total power consumed by processors by properly controlling the job schedule. So far, it has been optimized from the point of execution time and hardware cost. When a data flow graph is generated by high-level synthesis, the execution time of each job is decided in scheduling phase[6]. There have been two scheduling optimization problems: one is *time-constrained scheduling* which minimizes the hardware cost with a due time (period, cycle time) as a constraint, and the other is *hardware-constrained scheduling* which minimizes the execution time with a given hardware cost as a constraint. The goal of the former is to make the area of the system minimum and that of the latter the performance speed fast.

On the other hand, we aim to optimize the power consumption by scheduling jobs and controlling clock-supply to the processors with scheduled jobs with hardware cost fixed and period as a constraint. The timing of clock-supply is determined uniquely from the result of job scheduling.

Because controlling the clock-supply for each processor individually causes too much overhead, our framework is to design a system such that processors are clustered into blocks of proper size and clock-supply is controlled at block level.

It is a reasonable assumption that the power consumed by a processor is proportional to the duration while clock is supplied. If the clock-supply to processors which is not executing jobs is stopped, the power consumption of them is spared [1][7][8]. If no processors in a block are executing jobs, the block needs no clock to be supplied. Hence the clock should be supplied only when there exists at least one processor in the block which is executing job. The duration depends on the schedule of jobs. Hence the problem is to schedule jobs in order to minimize it.

This paper proposes an algorithm which schedules jobs minimizing power consumption when the period has been fixed to the minimum and jobs have been assigned to processors.

A similar problem was defined with an approach[2]: The problem proposed is to cluster processors into blocks to minimize power consumption when the period, job assignment and job schedule are given. But it is not practical to formulate the problem independent of the functions of processors and layout. While our formulation is considered more practical because it is assumed that the job assignment is decided somehow reflecting the function and layout.

A method proposed in this paper is based on the concept Best-Fit and As Soon As Possible. The former has been used frequently in Bin Packing[4][5] and the latter the major principle in minimum duration scheduling. We implemented the method and experimented. The result showed that about 30% of power is reduced on the system that 100 processors are clustered into 10 blocks with 1000 execution-order constraints generated randomly. Though this paper is just offering a principle and must be enhanced theoretically as well as practically to apply to practical problems, it is believed promising by the result.

In section 2, we describe the model of the circuit and a way to calculate power, and formulate an optimization problem. In section 3, we formulate the Minimum Power Job Scheduling problem and show the computational complexity with some restrictions. In section 4 we propose a heuristic algorithm. Section 5 is devoted to experiments. The conclusion is in section 6.

2 Circuit and power model

Consider a system with multiple processors as shown in Figure 1. A processor is called ON at step i if the processor receives the i-th clock-edge, and OFF at step i otherwise. The set of processors are clustered into blocks. The timing of clock-supply is decided at block level through a clock-controller which is inserted for each block. The clock is supplied through a clock-controller which is either of states ON or OFF corresponding to a clock-edge being passed through to the block or stopped, respectively. A block is called ON at step i if the block receives the i-th clock-edge, and OFF at step i otherwise.

The system executes a task which consists of jobs. A processor is called *active* when it executes a job. A processor can execute a job at a time when it is ON. Each job is executed continuously by an ON processor. The number of steps to complete a job is called the length of the job.

A block should be ON at every step when it includes an active processor while it can be OFF when all processors in it are not active. We assume that the power consumption of an ON processor per unit time is constant, that is, independent of either active or not and of the type of processors. Moreover, we assume that the power consumption of an ON processor is far larger than that of an OFF processor.

Since the clock-supply is controlled at block level, the minimization of the ON length of each block leads to the minimization of the power consumption of the system.

If the job assignment to each processor is given (actually which we assume), the total number of steps of active state of a processor is constant. Therefore, it is the only way for power reduction that the active state of a processor keeps in step with the active state of the other processors in the block. We let the block OFF at

the step when all processors in it are idle, by controlling the clock-controller.

Let W_1 (W_2) be the power consumption of an ON (OFF) processor per step. Let n_i and l_i be the numbers of processors in block B_i and ON steps of B_i, respectively. Let T be the number of steps to complete the task. The power consumed in B_i is described as follows.

$$pow(B_i) = \{l_i W_1 + (T - l_i)W_2\}n_i \quad (1)$$

And the power consumed in the system is:

$$pow(B) = \sum_{\forall B_i} pow(B_i) \quad (2)$$

Each job may not be executed independent of the other jobs. The constraint we consider is the *preceding constraint* which is such a relation that one job must be started after the other job. The preceding constraint is called *inner-processor constraint, inner-block constraint* and *inter-block constraint* if it is defined between two jobs assigned to a processor, to different processors in a block and to different processors in different blocks, respectively. All these definitions are conventional.

In the following, we assume that an assignment of jobs to processors is given. We focus on deciding the start step for each job. The objective of our job scheduling is to minimize the power consumption of the system.

Our strategy is to schedule one job at a stage after another. A job is called *fixed* if the start step of the job has been determined by the algorithm. A processor is called *occupied* at step i if a fixed job is scheduled to be executed at step i at the processor. A block is called occupied at step i if at least one processor in the block is occupied at step i. A maximal continuous occupied steps of a block is called a *shadow* of the block.

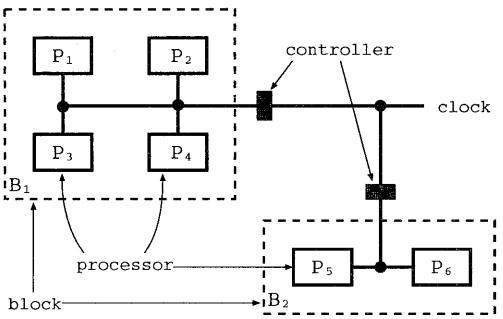


Figure 1: A multi-processor consisting of six processors clustered into two blocks, each supplied with controlled clock.

In Figure 1, the set of processors $\mathcal{P} = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ is clustered into two blocks $B_1 = \{P_1, P_2, P_3, P_4\}$ and $B_2 = \{P_5, P_6\}$. Examples of scheduling jobs $\mathcal{J} = \{J_{1,1}, J_{1,2}, \ldots, J_{6,2}\}$ assigned to the processors in Figure 1 are shown in Figure 2. A job $J_{i,j}$ is represented by a thick line and assigned to processor P_i. A preceding constraint is represented by a directed thin edge. A shaded step represents that the step of the block is occupied and continuous shaded steps represent a shadow. The periods of the schedulings in Figure 2 (a), (b) and (c) are 12, 8 and 9, respectively. The powers of them are 504, 480 and 468, respectively, when $W_1 = 10, W_2 = 1$.

It is expected that the power consumption of a shorter period scheduling is smaller because at least the power W_2 of OFF processors is consumed. On the other hand, it is expected that the power can not be reduced enough because the flexibility of the scheduling is reduced. The former of these conflicting expectations is observed in (a) and (b) of Figure 2, and the latter in (b) and (c) of Figure 2.

It is interesting to study the tradeoff between period and power. But in this paper, we assume that the period is fixed. Then the value of W_2 does not affect the optimization in scheduling, we let W_1 and W_2 be 1 and 0, respectively, in the following discussion.

3 Problem Formulation and Complexities

The problem is formulated as follows.

Min-Power Job Scheduling Problem

Input : Set of processors $\mathcal{P}$, set of blocks $\mathcal{B}$ which is a partition of $\mathcal{P}$, set of jobs $\mathcal{J}$, assignment of a job to a processor $\phi : \mathcal{J} \rightarrow \mathcal{P}$, set of preceding constraints $\mathcal{R}$ and period T.

Output : Schedule of all jobs in $\mathcal{J}$.

Objective : Minimization of the power consumption of the system.

First we show that the problem is NP-complete in general considering a special case.

Consider the problems shown in Figure 3 and 4 where assignment of jobs in the shadows S_1, S_2 in B_1 is unique. Therefore in both cases the remaining problem is to schedule the jobs assigned to P. In other words, we have to solve:

Job Scheduling with Two Shadows

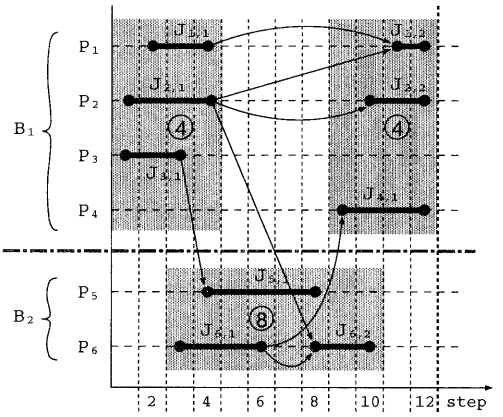
Instance : Set of jobs $\mathcal{J}$ assigned to P in block B, shadows S_1, S_2 of B.

Question : Are jobs in $\mathcal{J}$ scheduled within S_1 and S_2?

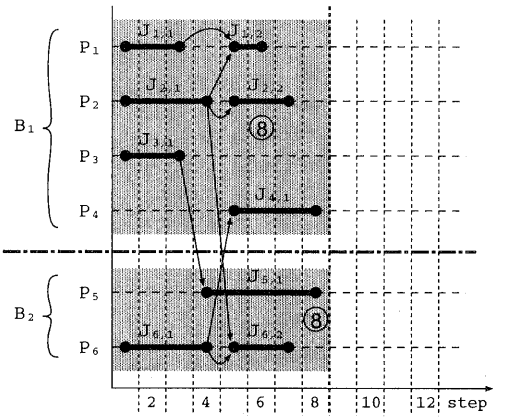
In the instance described in Figure 3, all the jobs assigned to processor P have no preceding constraint, while in Figure 4, they have linear-order inner-processor constraint.

We show two propositions that Job Scheduling with Two Shadows is NP-complete for the first case and that it is P for the second case.

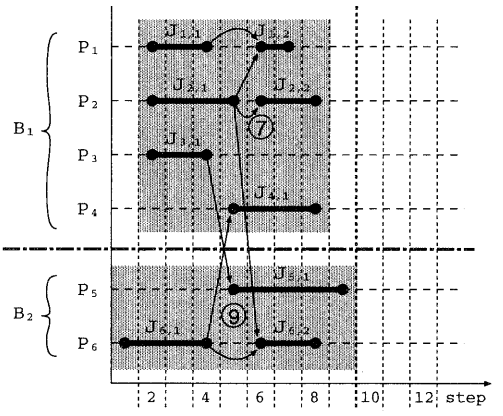
For the first proposition, we introduce the problem PARTITION which is known to be NP-complete for polynomial reduction[4].



(a) $pow(\mathcal{B}) = 504, T = 12$



(b) $pow(\mathcal{B}) = 480, T = 8$



(c) $pow(\mathcal{B}) = 468, T = 9$

Figure 2: Examples of scheduling for Figure 1.

PARTITION

Instance : Finite set A such that each element $a \in A$ has its size $s(a)$ which is positive integer.

Question : Is there a subset A' of A with a restriction that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

Theorem 1 *Job Scheduling with Two Shadows is NP-complete.*

Proof It is easy to show that Job Scheduling with Two Shadows is in NP.

We reduce the PARTITION into Job Scheduling with Two Shadows. For a given instance of PARTITION, we prepare a job $J \in \mathcal{J}$ of length $s(a)$ for $a \in A$ which is assigned to processor $P \in B$ and two shadows S_1, S_2 of length $\sum_{a \in A} s(a)/2$. There are no preceding constraints on jobs in $\mathcal{J}$. So Job Scheduling with Two Shadows is NP-complete. □

Minimum Power Job Scheduling Problem is generally NP-complete because it includes Job Scheduling with Two Shadows.

For the second proposition, we introduce an algorithm which provides an optimal solution (Figure 5). Its validity is self-evident and omitted.

Lemma 1 *Algorithm Job_Scheduling_for_Linear_Order_Two_Shadow outputs an optimal solution.*

Finally, we show other cases for which we can solve the Min-Power Job Scheduling Problem. They are imposed linear-order inner-processor constraints on all jobs.

The key idea we use is the ASAP (described in Figure 6). It schedules jobs from the first job in the linear-order to the last one into as early step as possible. This algorithm is often used in scheduling problems.

If $|\mathcal{B}| = 1$, the Min-Power Job Scheduling problem is P. The following fact is well-known.

Lemma 2 *Let* $|\mathcal{B}| = 1$.

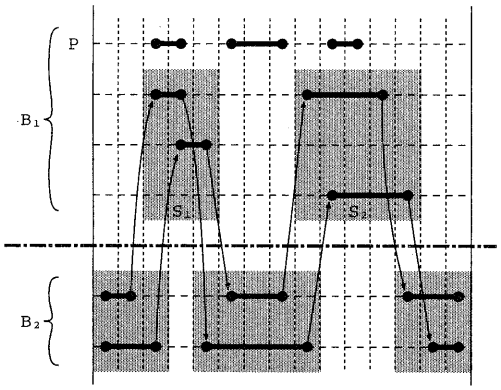


Figure 3: An instance of Job Scheduling with Two Shadows with no preceding constraint related to jobs in P.

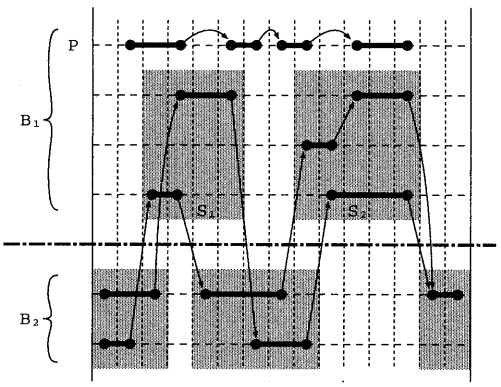


Figure 4: An instance of Job Scheduling with Two Shadows with only linear-order inner-processor constraints related to jobs in P.

1. *The power of the scheduling is minimum if and only if the period of the scheduling is minimum.*
2. *The minimum-period scheduling can be obtained by ASAP algorithm in $O(|\mathcal{J}| + |\mathcal{R}|)$ time.*

If there is no inter-block constraint, Minimum Power Job Scheduling Problem can be solved optimally even if $|\mathcal{B}| \geq 2$.

Lemma 3 *ASAP algorithm outputs an optimal solution in polynomial time for an instance with linear-order inner-processor constraints if there is no inter-block constraint.*

Proof Jobs can be scheduled for each block independently. The power consumed in a block can be optimized from Lemma 2. The total power is the sum of the powers consumed in each block. □

Algorithm
Job_Scheduling_for_Linear_Order_Two_Shadow
(Input: $\mathcal{J}$, $\{S_1, S_2\}$, T; Output: scheduling)

1. Schedule jobs assigned to P at as early step as possible from the first job through the last one while steps they are scheduled are properly contained in S_1.
2. Schedule unfixed jobs at as late step as possible from the last job through the first one while steps they are scheduled are properly contained in S_2.
3. If there is *one* unfixed job, then schedule it at as early step as possible if the number of unoccupied steps of P in S_1 is greater than that in S_2, at as late step as possible otherwise.
4. Otherwise, schedule the latest unfixed job as late step as possible and the other unfixed one as early step as possible.

Figure 5: The algorithm which solves Job Scheduling with Two Shadows problem with linear-order inner-processor constraint optimally.

The above discussion holds analogously along with ALAP (As Late As Possible) strategy which schedules jobs from the last job through the first one into as late step as possible. ALAP works only if the period T is given.

4 Best-Fit/ASAP Algorithm

We propose a heuristic algorithm BF-ASAP for the Minimum Power Job Scheduling Problem, which is described in Figure 7. We have learned the issues that the strategy Best-Fit and ASAP works preferably in reducing the power and that a single block can be handled easily. Our proposing algorithm is based on them. More precisely, the algorithm is sketched as follows: We follow the sequential scheduling. We relax the schedule of a single block and optimize it keeping the schedule of other blocks fixed. In a single block optimization, we apply Best-Fit with ASAP. That is, scheduling jobs into unoccupied steps aims to maximize the overlap of the jobs and the shadows of the block. This is one cycle and we repeat it appropriate times.

5 Experimental Result

We programmed BF-ASAP with C language. Also ASAP is programmed to produce the initial solution. It is also used to compare the performance with BF-ASAP. We generated 100 instances for the experiment as follows. Numbers of processors, blocks and jobs are 100, 10 and 1000, respectively. The number of processors in each

Algorithm ASAP (Input: $\mathcal{P}$, $\mathcal{B}$, $\mathcal{J}$, $\mathcal{R}$, p; Output: scheduling)

1. Let $L = \emptyset$.
2. Find all the first jobs without preceding jobs.
3. Let the start step of them be 1.
4. Add into L unfixed jobs not in L such that their preceding jobs are all fixed.
5. Schedule any job of L in such a way that its start step is as early as possible and delete it from L.
6. Go back to 4.

Figure 6: The algorithm which solves Min-Power Job Scheduling problem optimally when the number of blocks is 1.

block is evenly 10. The variety of length of jobs is between 1 and 10.

Associate a linear-order inner-processor constraint with jobs assigned to the same processor. Also associate an inner-block constraint or an inter-block constraint with some pairs of two jobs so as not to generate any loops. The period T is set to the minimum which is determined by applying ASAP to all the jobs.

Experimental result is shown in Table 1. Values in the table is the average of the solutions.

The columns in the table show from left to right sum of the number of inner-block constraints and inter-block constraints ($\times 1000$), lower bound of the power, the power and the execution time (ms) of the schedule solved by ASAP, the power and the execution time (ms) of the schedule solved by BF-ASAP, and the rate of power reduction of BF-ASAP to ASAP (%). Here the lower bound of the power is the sum of the minimum power calculated for each block neglecting the inter-block constraints. This calculation is possible because the single block scheduling is solved optimally by ASAP.

From the experimental result, it is observed that the power solved by BF-ASAP algorithm is about 30% smaller than ASAP for all cases except the case that the number of constraints is 0. It is remarkable that every value is near to the lower bound.

In this paper, we have set the condition of termination for BF-ASAP algorithm in the following way: "Terminate if power is larger than or equal to previous one continuously times of the number of blocks." We show experimental results to give the validity. First, for the instances generated in the same way, we compared the resultant power in five cases of the termination conditions according to the state that the power is larger than or equal to the previous one continues for: the number of blocks, twice the number of blocks, three times the number of blocks, four times the number of blocks and five times the number of blocks. But we got the solutions of

Algorithm BF-ASAP (Input: $\mathcal{P}$, $\mathcal{B}$, $\mathcal{J}$, $\mathcal{R}$, p, T; Output: scheduling)

1. (Initial solution) Schedule all jobs by ASAP.
2. (Initial setting) Let $L = \emptyset, i = 1$ and $j = 1$.
3. (Rescheduling) For each B_i,
   (a) Let all jobs assigned to the processors in B_i be unfixed.
   (b) For every job J, compute $margin(J)$ which is the number of possible schedules of J. (The earliest schedules is by ASAP and the latest by ALAP.)
   (c) Add into L the unfixed jobs in B_i all of whose preceding jobs are fixed.
   (d) Go to 5 if $L = \emptyset$.
   (e) Choose a job J in L whose $margin(J)$ is minimum.
   (f) Schedule J in such a way that the overlap with shadows is maximum. (If there is an arbitrariness, then schedule it into earliest step.)
   (g) Delete J from L and go back to (c).
4. (Power evaluation stage) Calculate the power.
5. (Setting of parameters for termination) Let $j = j + 1$ if the power is larger than or equal to the one in the previous stage (the power when B_{i-1} is rescheduled), and let $j = 1$ otherwise.
6. (Decision for termination) Terminate if $j > |\mathcal{B}|$.
7. (Increment) Let $i = \{(i + 1) \bmod |\mathcal{B}|\} + 1$ and go back to 3.

Figure 7: Our heuristic algorithm. Best-Fit strategy is applied to ASAP.

almost same quality. Next, for the instances generated in the same way, we compared the resultant power in the cases of the termination conditions according to the state that the power is larger than or equal to the previous one continues for: 1, 2, ..., 20 times, one-third the number of blocks, half the number of blocks, two-third the number of blocks, and the number of blocks. Again we noticed that the last case, which leads the proposed algorithm, output the best of all.

6 Conclusion

In this paper we assumed a multi-processor model, formulated the problem to decide the clock-supply step to subsets of processors, and proposed a job scheduling algorithm for low power design. Though the algorithm is not guaranteed to provide an optimal solution,

Table 1: Comparison of the power solved by ASAP and BF-ASAP.

# (k)	LB power	ASAP power	ASAP time	BF-ASAP power	BF-ASAP time	(%)
0	8525.7	8525.7	3	8525.7	8	0.0
2	10253.1	16998.4	5	11081.9	37	34.8
4	11483.4	18685.2	9	12126.7	60	35.1
6	12312.4	19541.4	14	12775.0	86	34.6
8	12988.8	20176.2	18	13389.2	119	33.6
10	13536.0	20740.5	24	13893.2	164	33.0
12	14097.6	21312.3	31	14407.8	226	32.4
14	14336.0	21433.6	38	14651.1	315	31.6
16	14586.1	21526.5	48	14889.5	408	30.8
18	14935.3	21862.4	57	15222.7	592	30.4
20	15233.5	22025.3	70	15540.4	837	29.4
22	15440.7	22036.0	83	15742.7	1118	28.6
24	15613.8	22316.9	98	15897.5	1853	28.8
26	15842.0	22362.2	114	16131.1	5015	27.9

it achieved about 30% lower power than that obtained by simply applying ASAP. This result is very significant in the field of development of LSI as a technique to enable the low power design of LSI chip, which shall be developed urgently because the problem about power is fatal to the quality and the integration of LSI.

All the discussions in this paper have been made based on a good result by a simple example. To make our idea more convincing, we need to pile up the data of various features. And, it is required to confirm the effect of application to practical problems.

Acknowledgment

This research is part of a project in CAD21 at Tokyo Institute of Technology.

References

[1] A. H. Farrahi and M. Sarrafzadeh. System Partitioning to Maximize Sleep Time. In *Proc. of International Conference on Computer Aided Design*, pp. 452–455, 1995.

[2] A. H. Farrahi, D. T. Lee, and M. Sarrafzadeh. Two-way and multi-way partitioning of a set of intervals for clique-width maximization. Private communication, to be published in Algorithmica.

[3] E. G. Friedman. *Clock Distribution Networks in VLSI Circuits and Systems.* IEEE Press, 1995. A Selected Reprint Volume, IEEE Circuits and Systems Society, Sponsor.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H.Freeman and Company, 1979.

[5] T. C. Hu. *Combinatorial Algorithms.* Addison-Wesley, 1982.

[6] T. Kim, N. Yonezawa, J. W. S. Liu, and C. L. Liu. A scheduling algorithm for conditional resource sharing–a hierarchical reduction approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 4, pp. 425–438, April 1994.

[7] T. Kitahara etc. A clock-gating method for low-power LSI design. In *Proc. of Asia and South Pacific Design Automation Conference*, pp. 307–312, 1998.

[8] J. Oh and M. Pedram. Power reduction in microprocessor chips by gated clock routing. In *Proc. of Asia and South Pacific Design Automation Conference*, pp. 313–318, 1998.