A Parallel Simulated Annealing Algorithm with Look-ahead Neighbor Solution Generation

Yusuke Ota Kazuhito Ito

Graduate School of Science and Engineering
Saitama University
255 Shimookubo, Sakura-ku, Saitama, 338-8570, Japan
{yuusuke,kazuhito}@elc.ees.saitama-u.ac.jp

Abstract— Simulated annealing (SA) is a general method to solve combinational optimization problems. SA generates a neighbor solution from a current solution randomly and evaluates the solution with a cost function. If the neighbor solution is better than the current solution, or otherwise stochastically, the neighbor solution is accepted as a new current solution. This process is iterated many times and therefore SA needs long execution time. We propose a fast SA method where some neighbor solutions are generated at a time in a look-ahead manner and evaluated in parallel. To increase the efficiency of the parallelized SA, a method to adaptively generate neighbor solutions is proposed to reduce void solutions not used in a SA chain.

I. Introduction

Simulated annealing (SA) is well known and widely used as a metaheuristic algorithm to find a locally optimal solution of a combinational optimization problem with large solution space [1]. In the area of LSI design, SA is applied to many kinds of design problems including floorplanning [2] and task scheduling [3, 4].

SA works as follows. A neighbor solution x' is generated, which is different in part from a current solution candidate x, and evaluated by computing a cost function of x'. By comparing the cost functions of x and x', if x' is better than x, x' is accepted as a new solution candidate and it replaces x. If x' is worse than x, x' is stochastically accepted at the probability calculated from the difference of the cost functions and the *temperature* which is a parameter controlling the SA process. Otherwise, x' is rejected. Then the series of the generation of a neighbor solution, evaluation of the cost function, and decision of accept/reject of the neighbor solution are performed again. This is illustrated in Fig. 1. Let the series be called a *step*. By iterating the step while gradually lowering the temperature, SA explores the solution space to find a solution with the optimal cost function. Beginning from an initial solution candidate with a starting temperature, the *step* is iterated until a termination condition is satisfied. This process is called an *SA chain.* To find a solution good enough, it is necessary to generate and evaluate many neighbor solutions within the solution space. Especially in LSI design, the solution space is increasing as the scale of the design increases. Thus the quite large number of solutions need to be evaluated to find a good solution within a large solution space. In addition, the criterion of the solution optimality is getting more and more complicated in the era of deep submicron technologies, which requires longer time to evaluate the cost function. Both the larger solution space and the complicated cost function increase the execution time of SA. An execution scheme of SA is desired which finds a good solution within a short execution time.

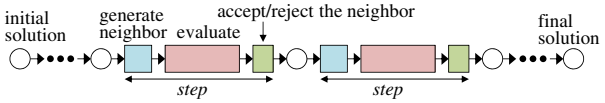

Fig. 1. Basic operation of simulated annealing (SA).

In this paper, a method to speed up an SA chain is proposed where more than one neighbor solution is generated in a *look-ahead* manner and these neighbor solutions are evaluated in parallel. Since the SA chain is not altered, the already tuned SA parameters can be reused. In addition, the parallelization of cost function evaluation is not necessary. Consequently, the proposed method has an advantage that the implementation of the given optimization problem is straightforward.

II. Simulated Annealing and its Fast Execution

A. Simulated annealing algorithm

The simulated annealing (SA) algorithm [1] is briefly reviewed. SA is a methheuristic algorithm to find an optimal solution to a given combinational optimization problem, by simulating the physical phenomenon where a material consisting of particles settles into a state with the minimum energy as the temperature of the material descends gradually. In SA, a neighbor solution is generated from a solution candidate, and the cost function of the neighbor solution is evaluated. Whether the neighbor solution is accepted and it replaces the current solution candidate or not is determined probabilistically depending on the temperature. The stochastic acceptance of the neighbor solution avoids the solution to be trapped to a local optimum and thus allows the exploration in the solution space.

Two schemes of SA algorithm exist which use different scheduling of descending the tempereture as shown in Fig. 2.

Input: Cost function $F(\cdot)$
Start temperature T_S and End temperature T_E
Temperature descending factor $\alpha(<1)$
Iteration counts N_1, N_2
Output: A solution y with the minimized $F(y)$

Get an initial solution candidate x and evaluate $F(x)$
$T \leftarrow T_S$
while $T > T_E$ **do**
  for $i = 1$ to N_1 **do**
    Generate a neighbor solution x' from x and evaluate $F(x')$
    if $F(x') \leq F(x)$ **then**
      $x \leftarrow x'$ (accept by improvement)
    else if ζ in Eq. (1) $> r$ **then** (r is a random value ranging from 0 to 1)
      $x \leftarrow x'$ (accept probabilistically)
    else
      Discard x'
    end if
  end for
  $T \leftarrow \alpha T$
end while

(a)

Get an initial solution candidate x and evaluate $F(x)$
$T \leftarrow T_S$
for $k = 1$ to N_2 **do**
  Generate a neighbor solution x' from x and evaluate $F(x')$
  if $F(x') \leq F(x)$ **then**
    $x \leftarrow x'$ (accept by improvement)
  else if ζ in Eq. (1) $> r$ **then** (r is a random value ranging from 0 to 1)
    $x \leftarrow x'$ (accept probabilistically)
    $T \leftarrow \alpha T$
    if $T < T_E$ **then** $T \leftarrow T_S$
  else
    Discard x'
  end if
end for

(b)

Fig. 2. The basic simulated annealing algorithms. (a) scheme 1. (b) scheme 2. The input and output are common to both schemes.

The SA algorithm in Fig. 2 minimizes the cost function. Although SA can be applied to the maximization of the cost function, only the minimization of the cost function is considered hereafter for simplicity.

The SA algorithm is described as follows. At first, a solution to the given problem is obtained. This is the initial solution of an SA chain and set it as a current solution candidate x. The cost function $F(x)$ is evaluated. The temperature parameter T is initialized to the start temperature T_S. A neighbor solution x' is generated which partly differs from x and $F(x')$ is evaluated. If $F(x') \leq F(x)$, x' is accepted because a better solution is found. On the other hand, if $F(x') > F(x)$, then

$$\zeta = exp\left(-\frac{F(x') - F(x)}{T}\right) \tag{1}$$

is calculated and x' is accepted at the probability of ζ. That is, x' is accepted if ζ is larger than a random value ranging from 0 to 1. If x' is accepted, it replaces x. Otherwise, x' is discarded. This procedure is called a *step*.

T is decreased using a descending factor $\alpha < 1$ as

$$T \leftarrow \alpha T \tag{2}$$

when the steps are iterated for a predetermined number of times (N_1 in scheme 1) or when x' is accepted probabilistically despite $F(x') > F(x)$ (scheme 2).

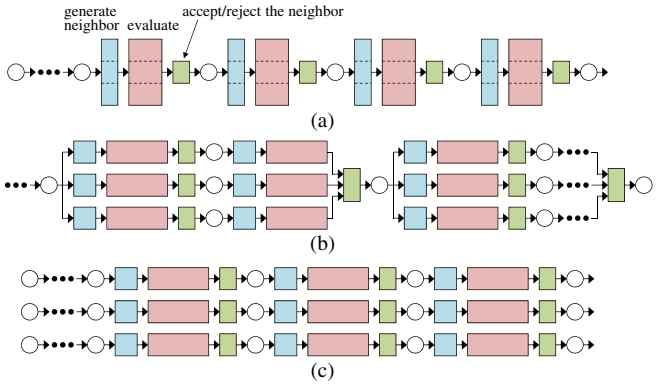

Fig. 3. Parallelization of SA. (a) parallelization of neighbor generation and evaluation. (b) evaluations in parallel. (c) parallel SA chains.

The start and end temperatures T_S and T_E are chosen so that ζ in Eq. (1) is about 1/2 when $T = T_S$ and almost zero when $T = T_E$. While T changes from T_S to T_E only once in scheme 1, T changes from T_S to T_E and then goes back to T_S for several times in scheme 2.

B. *Related work*

A two-stage SA method [5] is proposed as a technique to reduce the execution time of SA. In this method, first a solution with a good cost function is obtained by some method other than SA. Then using the solution as the initial solution candidate, an SA chain is executed with a low starting temperature. For the initial solution nearer to the optimal solution, a lower starting temperature can be chosen and hence the SA chain is executed in a shorter time. The two-stage SA is successfully applied to some area of optimization problems [6]. Conversely, the reduction of the execution time of SA cannot be expected unless a fairly good initial solution can be obtained. The application of the two-stage SA is generally a difficult task because it requires the analysis of the target optimization problem and the development of a rather simple algorithm to obtain a good initial solution.

In general, parallel processing is effective in speeding up tasks. Computers equipped a microprocessor with a multiple core are used in common these days and hence the parallel processing can be realized without special facilities. The parallelization of SA is possible in three levels as illustrated in Fig. 3. Those are, (a) parallelizing the generation of a neighbor solution and/or parallelizing the evaluation of the cost function, (b) generating two or more neighbor solutions and evaluating the solutions in parallel, and (c) executing two or more SA chains in parallel (referred to as asynchronous method in [7]). In (a), the procedure of neighbor solution generation and/or the procedure of evaluating the cost function need to be parallelized. Extracting the parallelism from the procedures would require deep understanding of them and the development of a parallel processing algorithm is also necessary. To make matters worse, the given optimization problem might not contain enough parallelism in nature. In (c) (the asynchronous

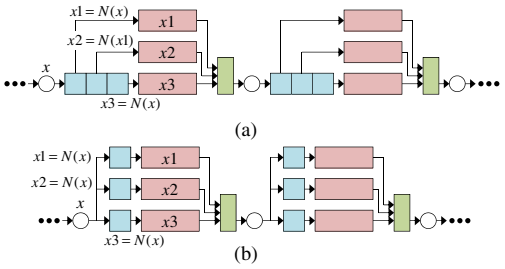

Fig. 4. Proposed parallelization methods of SA. (a) Look-ahead generation and parallel evaluation of neighbor solutions. (b) Parallel generation and evaluation of neighbor solutions.

method), simply more than one SA chain is executed in parallel without altering the SA procedures. If the total number of steps is fixed at a constant, e.g., K, each of $P > 1$ SA chains tries only K/P solutions within the same solution space. While the parallel execution at very high efficiency is possible with the asynchronous method, the solution optimality may be degraded because the exploration by each SA chain is more sparse than (a) and (b). As a method of paralleling in (b), SOEB is proposed [7]. The SOEB is illustrated in Fig. 3(b). In this method, more than one SA chain diverges from a current solution candidate. After some steps are iterated in each diverged SA chain, the solutions are gathered from the SA chains and compared, and the best is selected as a new solution candidate. Then SA chains are diverged again. SOEB performs the solution exploration differently than the single SA chain. Therefore, tuning of the SA parameters might be necessary to obtain a good solution.

III. Proposed Method

In this paper, a method to speed up an SA chain is proposed where more than one neighbor solution is generated in a *look-ahead* manner and these neighber solutions are evaluated in parallel. Then, according to the evaluated cost functions and the temperature, one of the neighbor solutions is accepted or all are rejected. This is illustrated in Fig. 4(a). $x1 = N(x)$ means $x1$ is a neighbor solution generated from x. This method is categorized into the type (b) of SA parallelization. The difference of the proposed method to SOEB (Fig. 3(b)) is that the neighbor solutions are generated in a look-ahead manner and therefore only a signle SA chain is treated, while two or more SA chains (diverged from a solution) are treated in SOEB.

A. Acceptance and rejection of solutions in SA

Figure 5 shows the relation among solutions of a problem. In this figure, the neighboring relation is represented by an arrow. x is a solution and $x1$, $x2$, $x3$, and so on are the neighbor solutions of x. Similarly, $x11$, $x12$, $x13$, and so on are the neighbor solutions of $x1$. In SA, a neighbor solution $x1$ is generated from a current solution candidate x, the cost function of $x1$ is evaluated, and $x1$ is either accepted or rejected. When $x1$ is accepted, $x1$ becomes a new solution candidate. Then a neighbor solution $x11$ is generated from $x1$ and evaluated. When $x11$ is rejected, $x1$ remains as a current solution candidate and another neighbor solution $x12$ is generated from $x1$.

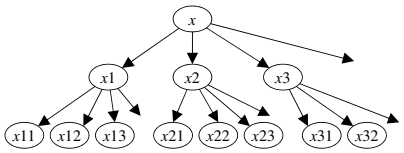

Fig. 5. Neighbor relations among solutions.

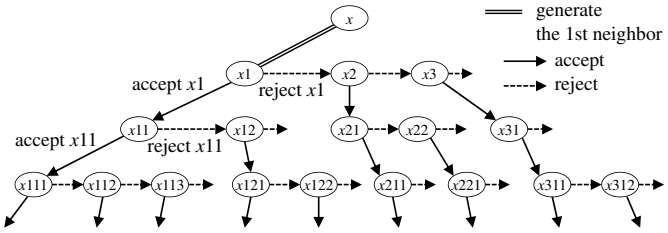

Fig. 6. The SA tree of solutions traversed by an SA chain.

As the neighbor solution is generated and either accepted or rejected, a different combination of solutions is traversed by SA steps. Thus a tree, where the nodes denote solutions and the directed edges denote acceptance/rejection relations, is constructed as shown in Fig. 6. Let the tree called an *SA tree*. A directed path rooted from the initial solution is selected in the SA tree according to the acceptance and rejection of neighbor solutions. The path in the SA tree is the SA chain. The solutions not on the SA chain are not visited during the exploration and hence generation and evalution of the cost functions for those solutions are not necessary.

B. Look-ahead processing of SA steps

Here a method is proposed where two or more neighbor solutions on an SA chain are generated in a look-ahead manner. Figure 7(a) shows a set of four neighbor solutions A, B, C, and D generated from a current solution candidate x and the relation among them. The solution A is a neighbor solution of x. The solutions C and D which are neighbor solutions of A are generated by expecting A would be accepted. In addition the solution B which is another neighbor solution of x is generated to prepare for the case of A being rejected. The cost functions of A, B, C, and D are evaluated in parallel. Then the combination of the acceptance or rejection of solutions is checked. First the acceptance/rejection of A is decided based on the cost functions $F(x)$ and $F(\mathrm{A})$. If A is accepted, then the acceptance/rejection of C is decided based on $F(\mathrm{A})$ and $F(\mathrm{C})$. In this case (A is accepted), the solution B is never visited and hence originally the generation and evaluation of B would have been unnecessary. Let the solution like B be called a *void* solution. Similarly in the case that C is accepted, D becomes a void solution. If C is rejeced, then the acceptance/rejection of D is decided based on $F(\mathrm{A})$ and $F(\mathrm{D})$. If A is rejected at first, then the acceptance/rejection of B is decided based on the cost functions $F(x)$ and $F(\mathrm{B})$. In this case (A is rejected), two solutions, C and D, are void. The combination of the acceptance and rejection of A and C and the resultant void solutions are summarized in Table I.

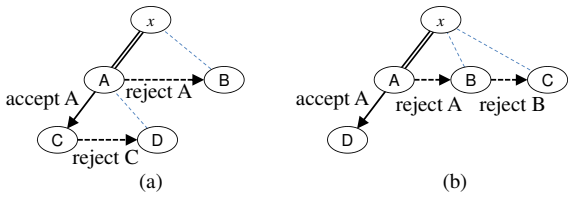

Fig. 7. Example sets of neighbor solutions generated from x in a look-ahead manner.

TABLE I
COMBINATION OF ACCEPT/REJECT OF SOLUTIONS FOR FIG. 7(A)

A	C	void solutions	ratio	new candidate
accept	accept	B, D	p^2	C
accept	reject	B	$p(1-p)$	D (D accepted) A (D rejected)
reject	–	C, D	$1-p$	B (B accepted) x (B rejected)

C. Generation of look-ahead neighbor solutions

The number of void solutions depends on the combination of the acceptance and rejection of the neighbor solutions. In addition, the average number of void solutions depends on the ratio of the combination of acceptance and rejection of the solutions. Table I also shows the ratio of each combination of acceptance/rejection of A and C for the solutions of Fig. 7(a) assuming the acceptance probability of A and C are both p. Note that when the probability of accepting a solution is p, the probability of rejecting it is $1-p$. The average number of void solutions is given as p^2-p+2. For another set of neighbor solutions show in Fig. 7(b), the number of void solutions depends on the acceptance/rejection of A and B, and its average is given as $-p^2+2p+1$. For $p=0.4$ and $p=0.7$, the average number of void solutions are calculated as shown in Table II. In the case of $p=0.4$, the set of neighbor solutions in Fig. 7(b) is preferable because it achieves the less number of void solutions than the set of neighbor solutions in Fig. 7(a). On the other hand, in the case of $p=0.7$, Fig. 7(a) is preferable. These results imply that the set of neighbor solutions with the least void solutions varies depending on the probability p of the solution acceptance (either the cost function is improved or stochastically). In order to maximize the efficiency of the solution exploration by minimizing void solutions, it is important to appropriately determine the set of look-ahead neighbor solutions to be generated based on the solution acceptance probability p.

Since an SA chain becomes a path on the SA tree, the solutions not on the path are void. Therefore, to minimize the void solutions, it is desired that the neighbor solutions generated in a look-ahead manner forms a directed path. However, the path is revealed only after the steps are executed on the solutions and it is difficult to predict which neighbor solutions should be generated in advance because the acceptance and rejection of solutions are influenced by a stochastical factor.

The acceptance probability of a solution is considered to be equal for continuous M solutions if M is not so large. Thus a method to generate M neighbor solutions is proposed so that the average number of void solutions would be minimized for

TABLE II
COMPARISON OF THE AVERAGE NUMBER VOID SOLUTIONS

neighbor solutions	average # void solutions	# void solutions	
		$p=0.4$	$p=0.7$
Fig. 7(a)	p^2-p+2	1.76	1.79
Fig. 7(b)	$-p^2+2p+1$	1.64	1.91

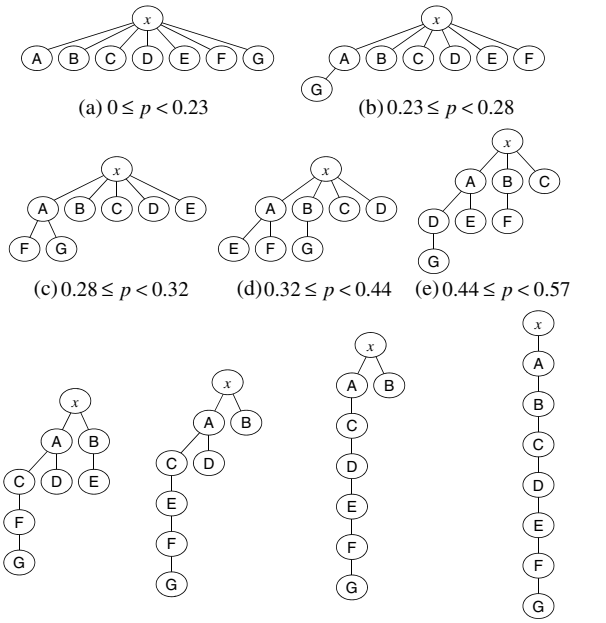

Fig. 8. Neighbor solutions with the least void solutions for the acceptance probability p ($M=7$).

a given acceptance probability p ($0 \le p \le 1$). All the possible sub-trees consisting of M neighbor solutions are enumerated for the SA tree shown in Fig. 6. For $M=7$, there exsit 429 different sub-trees. Then for a given value of p, the sub-tree with the least void solutions is identified. Figure 8 shows the identified sub-trees for $M=7$. If two or more sub-trees are with the least void solutions, one is chosen to minimize the number of different sub-trees. By using the average of the cost functions of the previously generated M solution as $F(x')$ in Eq. (1), ζ is calculated with the cost function $F(x)$ of the current solution candidate x and the temperature T. Then one of the sub-trees shown in Fig. 8 is selected assuming the acceptance probability $p=\zeta$ to generate the next M neighbor solutions.

D. Parallel generation of neighbor solutions

In general, there are dependencies among neighbor solutions and hence there exists precedence in generating neighbor solutions. For example in Fig. 8(b), the generation of solution G depends on the solution A, and it implies that A must be generated before G is generated. The set of neighbor solutions in Fig. 8(a) is an exception where there is no dependencies among the neighbor solutions. Therefore in case of the set of neighbor solutions in Fig. 8(a) is to be generated, the generation of neighbor solutions as well as the evaluation are done in parallel as shown in Fig. 4(b) to further speed up the procedure of

TABLE III
SA PARAMETERS FOR FLOORPLANNING PROBLEM

	Basic	Async.	SOEB-F	Proposed
N_1	10480	1497	1497	1497
N_2	1.2×10^7	1714285	1714285	1714285
# sync.	—	—	1145	—

the SA steps. Otherwise, the neighbor solutions are genereted serially as shown in Fig. 4(a).

IV. EXPERIMENTAL RESULTS

The proposed method was implemented using C++ programming language and run on a PC with a microprocessor AMD FX-8120 (3.1GHz, 8 physical cores) and 8 GB of main memory working with CentOS 6.2. The parallel processing was programmed as the multithreading using boost::thread routines of Boost C++ Libraries. The proposed method was implemented in two ways. In "Proposed 1," all the neighbor solutions are serially generated as shown in Fig. 4(a). In "Proposed 2," neighbor solutions are generated partly in parallel as described in Sect. III.D. For comparison, the basic (non-parallelized) SA, the asynchronous method [7], and SOEB-F method [7] were also implemented for both schemes 1 and 2. In the proposed method, the performance was strangely degraded when 8 neighbor solutions are evaluated using all the 8 cores, and the reason of the degradation is unknown. Therefore the experiments were executed so that 7 neighbor solutions were generated and evaluated in parallel using 7 cores.

A. *Floorplanning*

N300 in GSRC benchmark [8] was used for the experiment. 300 modules are placed without overlap and the area of the rectangle bounding the modules is minimized. The sequence-pair [2] is used to express solutions. The SA parameters are $T_S = 1.0 \times 10^4$, $T_E = 0.1$, $\alpha = 0.99$, and as shown in Table III. The parameters N_1 and N_2 were determined so that the total number of neighbor solutions is about 1.2×10^7 for all the methods. The average CPU time and the dead space are measured by executing the SA 10 times for each method.

The CPU times are compared in Fig. 9. The conventional methods achieve 4.2 times speed up and the proposed method achieves 3.9 times speed up. Since the neighbor solutions are generated sequentially in a single thread and more synchronizations among threads are necessary in the proposed method, the CPU time was longer than the conventional methods.

The dead space of the obtained floorplan is compared in Fig. 10. The smaller the dead space is, the more compact the floorplan is. In the case of scheme 1, the proposed method obtained a results as good as the basic method. The number of neighbor solutions generated for an SA chain is reduced in the conventional parallel method and therefore the solution exploration becomes sparse. On the other hand, in the proposed method, the exploration of solutions is essentially the same as the basic method. Thus similar solutions were obtained by the basic and the proposed methods.

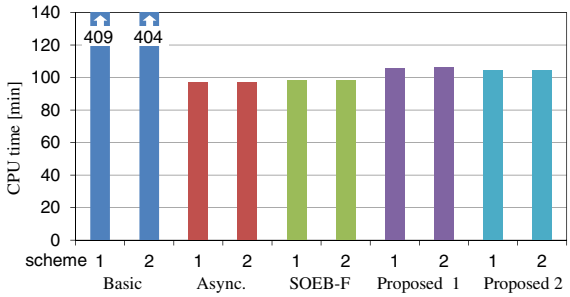

Fig. 9. CPU time for floorplanning.

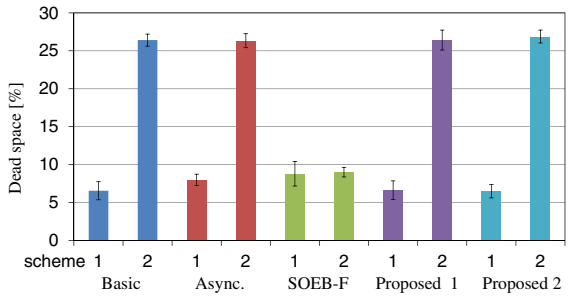

Fig. 10. Dead space of the floorplan.

The number of the sub-trees of neighbor solutions generated in the proposed method and the number of void solutions in each sub-tree were counted. Table IV shows the ratio of the generated 9 types of sub-trees and the percentage of which solution was the resultant (the next solution candidate) in each sub-tree. For example, 1.27% of all the generated sub-trees were the sub-tree of Fig. 8(d), the resultant solution was the current solution candidate x in 2.3% (the solutions A, B, C, and D were rejected and 3 solutions E, F, and G were void), the resultant solution was A in 8.1% (the solutions E and F were rejected, 4 solutions B, C, D, and G were void), and so on. The average ratio of void solutions was 62.5% for the sub-tree (d). The total ratio of void solutions was 37.1%. If the neighbor solutions traversed in the basic SA were divided into sets of 7 consecutive solutions and the best sub-tree of 7 neighbor solutions were assigned to each set, the ratio of void solutions is 24.7%. This is the lower bound of the ratio of void solutions and it can be seen that the proposed method generates appropriate combinations of neighbor solutions.

B. *Task scheduling*

In high-level synthesis for LSI, the minimization of energy dissipation in data communication among functional units (FUs) and registers (Regs) is considered. It can be achieved by concentrating the data communications onto a small number of pairs of FUs and Regs and highly communicating FUs and Regs are placed as near as possible. The solution to this problem requires optimizing the combination of the operation (task) scheduling, the binding operations to FUs and data to Regs, and the placement. In [3, 4], SA is used to find the task schedule which results in the binding where the sum of square of the number of data communications on each data communication link between FUs and Regs is maximized. The proposed

TABLE IV
THE RATIO OF SUB-TREE PATTERNS AND VOID SOLUTIONS IN FLOORPLANNING

Sub-tree	Generated Ratio (%)	Percentage (%) of the resultant solution (in parenthesis is # void solutions)								Void solutions (%)
		x	A	B	C	D	E	F	G	
(a)	94.13	47.2 (0)	15.8 (6)	10.4 (5)	7.6 (4)	6.0 (3)	5.0 (2)	4.3 (1)	3.7 (0)	29.9
(b)	0.72	0.6 (1)	22.5 (5)	22.7 (5)	8.4 (4)	3.6 (3)	1.5 (2)	0.7 (1)	40.0 (5)	67.8
(c)	0.51	1.2 (2)	8.5 (4)	21.8 (5)	8.1 (4)	3.5 (3)	1.3 (2)	41.5 (5)	14.0 (4)	65.0
(d)	1.27	2.3 (3)	8.1 (4)	7.9 (4)	7.7 (4)	3.0 (3)	42.5 (5)	14.3 (4)	14.2 (4)	62.5
(e)	1.04	4.8 (4)	7.4 (4)	7.2 (4)	7.5 (4)	13.8 (4)	14.4 (4)	14.2 (4)	30.6 (4)	57.1
(f)	0.76	11.3 (5)	7.3 (4)	6.8 (4)	14.5 (3)	14.2 (3)	13.9 (4)	9.3 (3)	22.6 (3)	54.2
(g)	0.23	11.3 (5)	6.7 (4)	20.6 (5)	14.4 (4)	13.3 (4)	9.5 (3)	6.9 (2)	17.3 (2)	53.4
(h)	0.27	10.3 (5)	21.7 (5)	21.8 (5)	14.2 (4)	9.2 (3)	6.6 (2)	4.8 (1)	11.5 (1)	54.7
(i)	1.08	31.5 (6)	20.9 (5)	13.8 (4)	9.7 (3)	6.9 (2)	4.8 (1)	3.3 (0)	9.0 (0)	56.7

TABLE V
SA PARAMETERS FOR TASK SCHEDULING PROBLEM

	Basic	Async.	SOEB-F	Proposed
N_1	350	50	50	50
N_2	401100	57300	57300	57300
# sync.	—	—	1146	—

method is applied to the SA in this problem. The example processing algorithm is the 5th order wave filter unfolded by factor 3. The SA parameters are $T_S = 1.0 \times 10^3$, $T_E = 0.01$, $\alpha = 0.99$, and as shown in Table V. The parameters N_1 and N_2 were determined so that the total number of neighbor solutions is about 4×10^5 for all the methods. The average CPU time and the sum of square of number data communication are measured by executing the SA 10 times for each method.

The CPU times of SAs are compared in Fig. 11. For scheme 1, the asynchronous, SOEB-F, and Proposed 2 methods achieve 3.2 times, 4.6 times, and 2.8 times speed up, respectively. The optimized results are shown in Fig. 12. The larger the score is, the better the result is. A trend can be seen that scheme 1 obtains the better result than scheme 2. Generating a neighbor solution in the exploration of the task scheduling takes a rather long time. Therefore Proposed 2 is faster than Proposed 1 since neighbor solution generation is done in parallel in more cases and hence the necessary CPU time is reduced. No significant difference of results between the methods exists. The proposed method achieves the speed up of SA without degrading the quality of the solution.

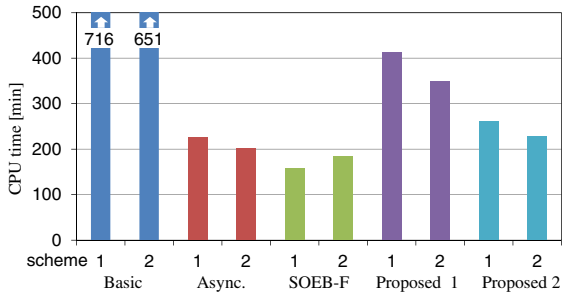

Fig. 11. CPU time for operation scheduling.

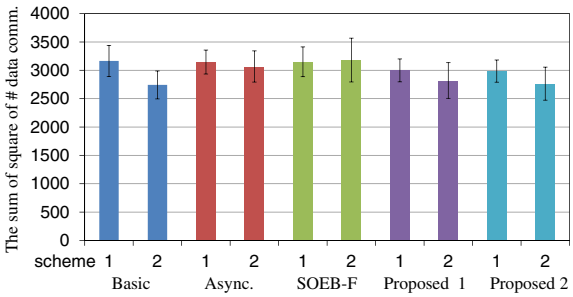

Fig. 12. The square sum of data communications.

V. CONCLUSIONS

In this paper, a method to parallelize a SA chain is proposed to reduce the execution time where two or more neighbor solutions are generated in a look-ahead manner and these solutions are evaluated in parallel. A method to generate neighbor solutions with minimized void solutions is also presented. In the case of the floorplanning problems, 3.9 times speed up against the basic SA was achieved with 7 cores of CPU without degrading the result. For the task scheduling problem, 2.8 times speed up was achieved and a result of comparable quality was obtained with the proposed method.

Studying the effectiveness of the proposed method in the case of higher parallelism and the parallel execution models other than multithreading remains as future work.

REFERENCES

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, no. 4598, pp. 671–680, 1983.

[2] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, "VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 12, pp. 1518–1524, 1996.

[3] K. Ito and H. Seto, "Reducing Power Dissipation of Data Communications on LSI with Scheduling Exploration," IPSJ Trans. System Level Design Methodology, vol. 2, pp. 53-63, 2009.

[4] H. Seto and K. Ito, " A Resource Binding Method to Reduce Data Communication Power Dissipation on LSI," IPSJ Trans. System Level Design Methodology, Vol. 3, pp. 257-267, 2010.

[5] J. Varanelli and J. Cohoon, "A Two-Stage Simulated Annealing Methodology," Fifth Great Lakes Symposium on VLSI, pp. 50-53, 1995.

[6] Y. Sheng, A. Takahashi, and S. Ueno, "2-Stage Simulated Annealing with Crossover Operator for 3D-Packing Volume Minimization," Proc. SASIMI 2012, pp. 227–232, 2012.

[7] E. Onbaşoğlu and L. Özdamar, "Parallel Simulated Annealing Algorithms in Global Optimization," Journal of Global Optimization, vol. 19, pp. 27–50, 2001.

[8] GSRC Benchmarks, http://vlsicad.eecs.umich.edu/BK/FPUtils/.