Institut für Technische Informatik

Abteilung Eingebettete Systeme

Universität Stuttgart
Pfaffenwaldring 5b
D-70569 Stuttgart

Masterarbeit Nr. 01936 - 00x

Heuristics for Design Time Optimization of System-on-Chip Memory Power Consumption

Li Jinpeng

Studiengang:	INFOTECH (Information Technology)
Prüfer:	Prof. Dr.-Ing. Martin Radetzki
Betreuer:	M.Sc. Manuel Strobel
begonnen am:	20.06.2016
beendet am:	02.12.2016
CR-Klassifikation:	B.8.2

Acknowledgment

Acknowledgment goes here...

Abstract

Abstract goes here...

Contents

1 Introduction

Nowadays in the field of embedded systems design , power consumption has become one of the most important design factors especially in the domain of Systems-on-Chip. One of the important issues to design power-efficient embedded system is the power consumed by memories and memory related components. Some researchers have claimed that large fraction of power is dissipated by memories [Mai+07; BMP00]. Thus, memory power optimization plays a significant role in the design of power-efficient embedded systems. One of the most effective and common approaches to reduce memory power consumption is the memory partitioning method which is proposed in several articles and books [BMP00; His05; MBP02, p.43].

The rationale of memory partitioning is , on the one hand, to split one single large memory into several small memory instances which can be accessed individually [Mai+07]. On the other hand, according to the memory access patterns, frequently accessed address ranges are grouped to smaller memory instances while rarely accessed address ranges are grouped to the larger ones [SER16]. The memory partitioning is one of the combinatorial optimizations since the process is to find the optimal memory configuration from a set of memories and applications. There are many methods that can be applied in the domain of combinatorial optimization. One approach is the integer linear programming (ILP) which solves the optimization problem through a mathematical model described by certain integer linear relationship. This approach is proposed in [SER16] for the memory power optimization using memory partitioning method. Another commonly deployed approach is the heuristic which is based on searching mechanisms. In many classical combinatorial optimization problems such as traveling sales man (TSP) problem, heuristics have been deployed and near optimal solutions have been provided by using them.

When dealing with an optimization problem with a very large solution space, the algorithms that are used to find the exact optimal solution may be ideal. However, the required execution time of such algorithms may be unacceptable in practice. Even in some problem sets, the exact optimal solution can not be found by the conventional algorithms. In such cases, the user of heuristics can obtain a near optimal solution within a reasonable time frame. Though the solution provided by heuristics may be not the exact optimal one, it still can be considered as a valuable solution of the optimization problem. One key feature of heuristics is the trade-off between algorithm efficiency and precision. The solution quality and the execution time of the algorithm can be balanced by users according to their different requirements.

The goal of this thesis work is to apply heuristics for the purpose of the memory power optimization using memory partitioning method. The targeted problem set is the same which is used in [SER16]. Firstly, multiple potential heuristics are theoretical examined.

After the comparison between them, the most promising algorithm is identified and proposed for the optimization objective. Secondly, the selected heuristic is adapted to a existing formal power model which is proposed in [SER16]. Then the framework of the chosen algorithm along with its parameters are realized for the power model. Lastly, the evaluation of the implemented heuristic is performed. The optimal solutions found by the chosen algorithm are compared with the obtained results in [SER16].

The structure of this document is organized as following. Chapter 2 introduces the basic knowledge of the memory partitioning method and the formal memory power model. Besides, the discussion of potential heuristics are made. And in this chapter, the first goal of this thesis work is achieved by proposing the simulated annealing algorithm according to the comparison between the discussed heuristics. Chapter 3 discusses the related works for the memory power reduction and the simulated annealing algorithm. Chapter 4 represents the design details of the simulated annealing for the memory power optimization using memory partitioning method. The evaluation of the simulated annealing algorithm results is given in Chapter 5. And Chapter 6 concludes this thesis work.

2 Basics

This chapter represents the basic knowledge for this thesis work. Section 2.1 introduces the memory partitioning method with an existing formal power model. Section 2.2 discusses three potential heuristics. In Section 2.3, the discussed heuristics are compared with each other and the most promising one is selected to be used for the memory power optimization.

2.1 Memory partitioning and formal power model

This section is just a represent of the original work in the article [SER16] and no new material is included.

There are two central concepts for memory power optimization using the memory partitioning method. One concept is the allocation α which is a set of memory instances of certain memory types. The memory types are described by several parameters related to their physical characteristics. The other concept is the binding β of the application's code and data fragments to the selected memory instances. The code and data fragments of an application are referred as application profiles. And each application is represented by a set of profiles [SER16]. Every profile is characterized by some user-defined parameters. Table 2.1 and Table 2.2 describe the relevant parameters of the memory type and application profile respectively.

Parameter	Description
Size	Provided memory space
Area	Consumed on-chip area
Read current	Current required by the read operation
Deselect current	Current required when no operation is performed
Stand by current	Current consumed all the time
Write current	Current required by the write operation (RAM only)

Table 2.1: Memory Type Related Parameters

A configuration for the memory system is defined as a pair of an allocation of memory instances and the corresponding binding for the application profiles. Through the memory partitioning, an optimal configuration is expected to be found such that the average power consumption by the selected memory instances and the interconnect is the lowest under certain predefined constrains. To achieve this optimization objective, a formal power model

Parameter	Description
Duty cycle	The active time frame in the profile's period
Read probability	The probability to perform the read operation in profile's duty cycle
Write probability	Same with read probability except for write operation (RAM only)
Size	Required memory space by the profile

Table 2.2: Application Profile Related Parameters

with four constraints are defined by the authors of [SER16]. In the following, some concepts related to the existing power model along with the constraints are introduced.

Let M denotes the memory type set that can be used in the memory system. The allocation α is represented as a vector whose size is the number of memory types in the set, $\alpha \in \mathbb{R}_0^{|M|}$. Each element in α is the number of instance for the corresponding memory type and its value should be non-negative. Let A denotes the application set provided in the optimization problem. For each application, let P_a represent its profile set. Then the binding β to the corresponding α is in the form of a binary matrix with size $|P_a| \times |M|$, $\beta \in \{0,1\}^{|P_a| \times |M|}$. If the matrix element value β_{aij} of an application a is 1, it means that the profile i of application a is bound to the memory type j. Otherwise, the memory type j does not contain the profile i for application a.

The following are the four predefined constraints.

- Constraint 1
  Define a fixed integer number $mems_{max}$, The total number of the allocated memory instances should not exceed $mems_{max}$.

$$\sum_{i=1}^{|M|} \alpha_i \leq mems_{max} \tag{2.1}$$

- Constraint 2
  Define a vector A_M with size of $|M|$, $A_M \in \mathbb{R}^{|M|}$. Each element in A_M indicates the area consumption of the corresponding memory type. Define a function $A_F \colon \mathbb{N}_0 \to \mathbb{R}$. A_F outputs the area consumed by the interconnect according to the total number of the allocated memory instances. The area required by both memory instances and the interconnect should be limited to a maximum value, $area_{max}$.

$$\sum_{i=1}^{|M|} \alpha_i \cdot A_{M,i} + A_F(\sum_{i=1}^{|M|} \alpha_i) \leq area_{max} \tag{2.2}$$

- Constraint 3
  For each application, every profile should be contained in one and only one memory type. If there are multiple applications, all of them should satisfy this constraint.

$$\forall a \in [1, |A|], \forall i \in [1, |P_a|] : \sum_{j=1}^{|M|} \beta_{aij} = 1 \tag{2.3}$$

- Constraint 4
  Define a vector $\sigma^{P_a} \in \mathbb{N}_0^{|P_a|}$ whose elements indicate the memory consumed by the profiles individually. Define another vector $\sigma^M \in \mathbb{N}_0^{|M|}$ where the total memory spaces of the memories types are recorded in the corresponding elements. For every application, each memory type should have large enough memory space to contain all the profiles that are bound to it.

$$\forall a \in [1, |A|], \forall j \in [1, |M|] : \sum_{i=1}^{|P_a|} \beta_{aij} \cdot \sigma_i^{P_a} \leq \alpha_j \cdot \sigma_j^M \tag{2.4}$$

The configurations satisfy all the introduced constraints are considered as valid for the optimization problem and their average power consumption is computed by the following model. Here, the illustration for the power model is focused on ROM where only read operations are required.

$$P_j(a) = P_{read,j}(a) + P_{desel,j}(a) + P_{stdby,j} \tag{2.5}$$

The power consumption of one single memory type is consisted of three parts. Let $P_j(a)$ denotes the power consumed by memory type j for the application a. Seen from Equation 2.5, the three power fractions are:

$P_{read,j}(a)$, consumed power when reading from the memory.

$P_{desel,j}(a)$, consumed power when the memory is deselected.

$P_{stdby,j}$, power continuously consumed by the memory.

Equations 2.6 to 2.8 show how to compute the three power fractions individually. In these equations, d_i and p_{ri} are the duty cycle and read probability of application profile i respectively. $I_{r,j}(f)$, $I_{d,j}(f)$ and $I_{s,j}$ are the read, deselect and standby currents of memory type j respectively. V is the voltage of the power supply to the memory system.

$$P_{read,j}(a) = \sum_{i=1}^{|P_a|} \beta_{aij} \cdot d_i \cdot p_{ri} \cdot I_{r,j}(f) \cdot V \tag{2.6}$$

$$P_{desel,j}(a) = \left(\alpha_j - \sum_{i=1}^{|P_a|} \beta_{aij} \cdot d_i \cdot p_{ri} \right) \cdot I_{d,j}(f) \cdot V \tag{2.7}$$

$$P_{stdby,j} = \alpha_j \cdot I_{s,j} \cdot V \tag{2.8}$$

In this model, $P_j(a)$ is regarded as the unit power to calculate the total power consumed by all memories for all applications. The average power consumption P_{avg}, also includes the contribution of the interconnect. Let P_F denotes the function that outputs the power required by the interconnect. Then, P_{avg} is computed according to Equation 2.9.

$$P_{avg} = P_F(\sum_{i=1}^{|M|} \alpha_i) + \frac{1}{|A|} \sum_{a=1}^{|A|} \sum_{j=1}^{|M|} P_j(a) \tag{2.9}$$

2.2 Heuristics

There are a lot of existing heuristics. At the early time of heuristics' usage, a certain algorithm is applied to solve one particular optimization problem. These problem-dependent heuristics can not be adapted to other optimization processes. To improve the portabilities of heuristics, some algorithms are invented as parameterized interfaces that can be widely deployed for a variety of optimization problems. Such problem-independent heuristics usually are consist of a base framework with several parameters. Only the parameters are related to the optimization problems. When using one of those heuristics for different problem sets, the algorithm framework is common while the parameters should be set up according to the problem requirements. In the recent years, there is a new trend of heuristic which is called hyper-heuristic. The hyper-heuristics provide a high-level strategy to seek one or several low-level heuristics to generate a proper algorithm for solving an optimization problem. The hyper-heuristic is a cutting-edge technique and it is beyond the knowledge of this work. For the memory power optimization, the problem-independent heuristics are mainly focused because of their extensive usage.

There are a variety ways to classify the heuristics. One common classification is to differentiate the algorithms according to their searching mechanisms. To be simplified, the heuristics are divided as local search-based and non-local search-based in this work. The well known local search algorithm aims to seek for the optimal solution by iteratively moving to a better solution in the neighborhood. However, the local search algorithm is greedy and cannot guarantee providing the good enough solutions because it may trap in local optimums. The idea of local search-based heuristics is to avoid the local optimum trap through some criteria for the solution selection and improve the result's quality. Heuristics of this kind output only one single optimal solution. Some classical local search-based heuristics are simulated annealing, tabu search, guided local search, etc. Unlike local search-based heuristics, the non-local search-based heuristics usually seek for a set of good enough solutions. By manipulating some defined solution characteristics, it can guide the searching process to the global optimal region. Some typical non-local search-based heuristics are genetic algorithm, particle swarm optimization, ant colony optimization, etc. Normally, the frameworks of non-local search-based heuristics are more complicated than that of local search-based algorithms. And the expected result for memory power optimization is one optimal configuration not a set of configurations. Therefor, the local search-based heuristics are the main focuses in this work. In this section, the local search algorithm along with its local optimum trap is discussed first. Then, two typical local search-based heuristics, tabu search and simulated annealing, are represented. Lastly, the most promising algorithm is proposed to the memory power optimization according to the comparison between these heuristics.

2.2.1 Local search algorithm

Local search algorithm is one of the simplest heuristics. Given an optimization problem, it starts from an initial solution and searches in the current solution's neighborhood. If a better solution is found, the current solution is replaced by it. The searching process is repeated until there is no better solution in the current solution's neighborhood. Then it outputs the current solution as the algorithm result.

Algorithm 2.2.1: Local Search Algorithm

Input: an optimization problem
Output: an optimal solution

```
1 current solution = initial solution;
2 while not terminate do
3     generate a neighboring solution;
4     evaluate the neighboring solution;
5     if neighboring solution is better than current solution then
6         current solution = neighboring solution;
7 output current solution;
```

Algorithm 2.2.1 shows the pseudo-code of local search process. There are four main steps in the algorithm. First step is finding an initial solution and setting it as the current solution. The initial solution should be valid for the optimization problem. In the second step, a neighboring solution is generated by certain mechanisms. And the third step is to compare the neighboring solution with the current one through an object function. The object function is a method to indicate how good the solution is. The last step is the selection criterion for solutions. Local search algorithm selects the better one between the current and the neighboring solution, which is a naive criterion.

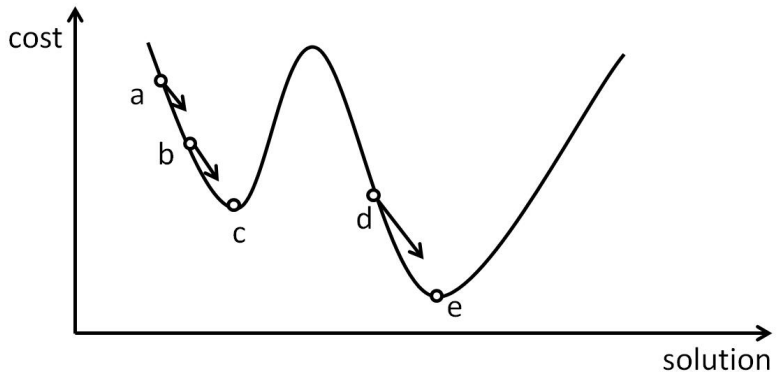

Figure 2.1: Local Optimum Trap

Though the local search algorithm is simple, the solution it provides may be the local optimal one. This is the major problem of the local search algorithm. Figure 2.1 illustrates

the local optimum trap. Suppose the optimization problem is to find the solution with minimum cost, the local search algorithm starts with the initial solution a. The cost of the neighboring solution b is lower than the cost of a, then b is selected and becomes the current solution. The same searching process is repeated until the current solution reaches c. There is no better solution in c's neighborhood, thus the algorithm outputs solution c and terminates. However, solution c is only the local optimum while the global optimum is solution e which is not in c's neighborhood. In order to reach solution e, the algorithm has to move to solution d whose cost is higher than c's cost. And this violates the solution selection criterion of the algorithm. Another drawback of the local search algorithm is that the result quality is dependent on the initial solution. If the algorithm starts with solution d, the output will be the global optimum e. These two disadvantages make the local search algorithm an improper choice when global optimal solution is required for the optimization problems.

2.2.2 Tabu search algorithm

One of the improvements to the local search is the tabu search algorithm. It is based on the local search but it avoids to be stuck at the local optimal trap through a different selection strategy for solutions. As discussed in section 2.2.1, once the local search algorithm is trapped at a local optimal solution, it can not move any further due to the naive solution selection criterion. To solve this problem, Fred Glover proposes the concepts of the tabu list and the aspiration criterion in [Glo89] and [Glo90]. The following discussion is based on Fred Glover's proposal.

The key element of the tabu search algorithm is the tabu list. It imitates the memory function of human brain to guide the searching process. It is used to record the tabu objects. The tabu objects can be defined as the solutions, solution movements or values of the object function. The tabu list has a limited size which is one of the algorithm parameters. The improvement to the local search algorithm is gained from the solution selection strategy which is usually called the tabu move. There are two rules in the tabu move. The first rule is to exclude the solutions recorded in the tabu list from a set of neighboring solutions. The second rule is to select the best in the rest of the neighboring solution set. The solution chosen by the tabu move is set as the current solution. Another concept of the tabu search algorithm is the aspiration criterion During the searching process, the best-so-far solution is kept recorded in the searching history. The aspiration criterion is to examine the neighboring solution set to find out if there are solutions that are better than the current best-so-far solution. If such solutions are found, then the best of them is selected and is set as the current solution even if it is recorded in the tabu list. If no such solutions is found, the algorithm continues with the tabu move.

Algorithm 2.2.2 is the pseudo-code of the tabu search framework. At the beginning of the algorithm, it generates a valid initial solution and sets it as the current solution and the best-so-for solution. The parameters are set up according to the algorithm inputs. And a tabu list is created as empty. After the initialization, the algorithm generates a set of neighboring solutions by some certain mechanisms and evaluates it by an object function. After this step, there are two different branches. One branch is the execution of

Algorithm 2.2.2: Tabu Search Algorithm

Input: an optimization problem, algorithm parameters
Output: an optimal solution

```
set algorithm parameters;
current solution = initial solution;
best-so-far solution = initial solution;
set tabu list as empty;
while not terminate do
    generate a set of neighboring solutions;
    evaluate neighboring solutions;
    if aspiration criterion satisfied then
        execute aspiration criterion;
        update current solution;
        update tabu list;
        update best-so-far solution;
    else
        execute tabu move;
        update current solution;
        update tabu list;
output current solution;
```

aspiration criterion. If the condition of the criterion is satisfied, the solution selected by the criterion is set as the current solution and the best-so-far solution. Also, the updated current solution is added to the tabu list. The other branch, tabu move, is executed when the condition of the aspiration criterion is not satisfied. The current solution and the tabu list are updated to the solution chosen by the tabu move while the best-so-far solution is not updated because there is no solutions better than it. There are two cases for the tabu list updating. At the early stage of the algorithm, the list is not full. The solutions are added into the list sequentially. When there is no space for a new recorded solution, the oldest solution in the list is replaced by the new one. The same searching process is repeated until the termination condition is satisfied and the algorithm outputs the current or the best-so-far solution as the optimization result. Some algorithm parameters are related to the termination condition. One simple method to terminate the algorithm is setting a fixed iteration number. Thus this fixed number is one parameter of the algorithm. However, this method can not guarantee the solution quality. Another common termination mechanism is to count the appearances of a particular solution. If this solution appears for a max number of time, the algorithm can terminate. Therefor, the max number is also one algorithm parameter.

Figure 2.2 illustrates how the tabu search algorithm avoids the local optimum trap. The solid line represents the execution of the aspiration criterion and the dotted line is the tabu move. The same problem set in section 2.2.1 is used. The tabu search algorithm starts from the initial solution a with an empty tabu list (assume the list size is large enough). After the neighboring solutions are generated, it finds solution b satisfies the aspiration criterion. Then b becomes the current solution and the best-so-far solution is updated to

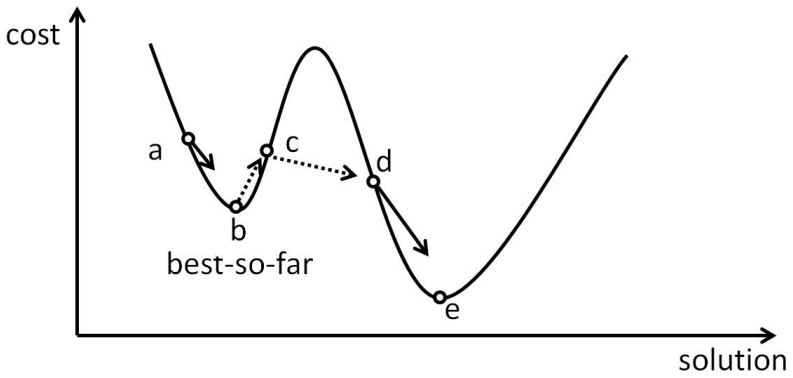

Figure 2.2: Tabu Search's Avoidance to Local Optimum Trap

b as well. Also, b is added to the tabu list. Known from the figure, solution b is a local optimum and there is no neighboring solution satisfies the aspiration criterion. Thus, the algorithm continues with the tabu move. During the tabu move, solution b is excluded from the generated neighboring solution set because it is stored in the tabu list. And solution c is found to be the best among the rests of the set. Thus, c becomes the current solution and it is added to the tabu list. In the next iteration, solution b and d both are c's neighboring solutions. However, b is still in the tabu list and the aspiration criterion is not satisfied. Thus, the solution d is selected by the tabu move as it is the best among the rest of neighboring solutions. The same searching process is repeated until the algorithm reaches solution e which is the global optimum. By selecting a worser solution in the tabu move, the tabu search algorithm can move to a new searching region and avoid the local optimum trap.

2.2.3 Simulated annealing algorithm

Another enhancement to the local search is the simulated annealing algorithm. The idea of this algorithm is to imitate the metal annealing process. Three steps are performed in the annealing process. Firstly, the metal is melted at a very high temperature. The second step is to give a small disturbance to the metal and wait until the metal reaches its equilibrium state at the current temperature level. Then in the third step, the metal is cooled down slowly. The last two steps are repeated until a low temperature limit is reached at which the metal is in the ground state. Inspired by this process, the simulated annealing algorithm is proposed in [S K83]. And in the algorithm, the metropolis criterion is introduced to avoid the local optimum trap. The following discussion is based on the proposal of S. Kirkpatrick et al. in [S K83].

The metropolis criterion is a probabilistic technique to choose the solutions. In the local search algorithm, the better neighboring solutions are always selected and it is not possible to accept a worser one. In the simulated annealing algorithm, the metropolis criterion accepts the neighboring solution according to a probability which is related to a control parameter. The control parameter is the imitation of the temperature in the metal annealing precess. To be clarified, the control parameter is referred as temperature in the rest of this

document.

$$p = \begin{cases} 1 & \text{, if } s_{next} \text{ is better than } s_{current} \\ \exp\left(-\frac{f(s_{next})-f(s_{current})}{t}\right) & \text{, otherwise} \end{cases} \tag{2.10}$$

Equation 2.10 represents the computation of the acceptance probability. In the equation, s_{next} is the next neighboring solution of the current solution $s_{current}$. p is the probability to accept s_{next}. And $f()$ is the object function. The value of $f()$ is referred as the solution cost. t is the temperature. It is can be seen from the equation that if s_{next} is better than $s_{current}$, the metropolis criterion behaves like the local search algorithm. And if s_{next} is worser, it can still be accepted depending on the computed probability. One thing is noticed in [S K83] is that the difference between costs of s_{next} and $s_{current}$ should be a positive value. Thus in the second case of the equation, the acceptance probability becomes smaller with the decrease of the temperature level.

The basic idea of simulated annealing algorithm is to combine the local search with the imitation of the metal annealing process. And the metropolis criterion is used to improve the solution quality. The algorithm searches better solutions in the current solution's neighborhood at different temperature levels. The searching process at the same temperature is repeated until certain termination condition is satisfied. The temperature is controlled to be reduced step by step as the same cooling procedure executed in the metal annealing process. At the early stage of the algorithm, the current solution is updated randomly because a lot of worser solutions are accepted due to the high temperature. However, with the decrease of the temperature, the probability to select worser solutions becomes smaller. This makes the searching space closer to the optimal region.

Algorithm 2.2.3: Simulated Annealing Algorithm

Data: an optimization problem, algorithm parameters
Result: an optimal solution

```
1  set algorithm parameters;
2  current solution = initial solution;
3  t = initial tempreture;
4  while not terminate do
5      while not terminate do
6          generate a neighboring solution;
7          evaluate the neighboring solution;
8          if neighboring solution is better then
9              accept neighboring solution;
10         else
11             comput the accept probability;
12             accept neighboring solution according to the accept probability;
13         update current solution;
14     decrease t;
15 output current solution;
```

Algorithm 2.2.3 shows the pseudo-code of the simulated annealing process. In the preparing stage, the algorithm sets up the parameters according to the algorithm inputs. And the initial solution is set as the current solution. The temperature t is set up to a high enough value so that the random solutions can be selected by the metropolis criterion. The main framework of the simulated annealing can be divided into two nested loops. The outer loop is just the reduction of the temperature. The inner loop is the execution of the local search with the metropolis criterion. Firstly, the neighboring solution is generated by certain mechanisms and evaluated by the object function. Then the current solution is updated according to the result from the metropolis criterion. The inner and outer loop terminate when some predefined conditions are satisfied.

Figure 2.3 illustrates how the simulated annealing algorithm can avoid the local optimum trap. The solid line represents the acceptance of better solution while the dotted line is the acceptance of worser solution. The same problem set in Section 2.2.1 is used. Suppose the current solution is a at a proper temperature level. The algorithm finds out that the neighboring solution b is better. Through the metropolis criterion, the current solution is updated to b which is the local optimum. In the following iteration, the neighboring solution c with higher cost is generated. Because of the proper temperature, it is possible to accept c with the computation of the acceptance probability. And the same searching process is repeated until the global optimal solution e is reached. It is still possible that the algorithm moves away from e at the current temperature. However, the current temperature is assumed to be proper. And it applies that even if the global solution e is discarded by the metropolis criterion, the selected solution can not be much worser than e. If the algorithm terminates at this current temperature, a near optimal solution can be obtained. And if the algorithm continues with the decrease of the temperature and a low enough temperature is reached, it can move back to e agian and a worser solution can not be selected.

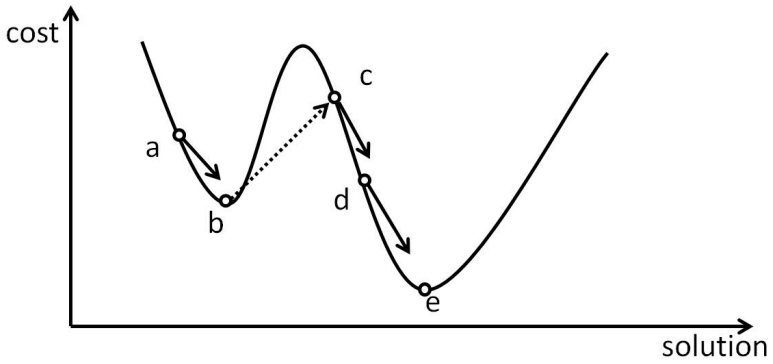

Figure 2.3: Simulated Annealing's Avoidance to Local Optimum Trap

2.3 Heuristics selection

Though the discussed heuristics in Section 2.2 seem to be the potential approaches for the memory power optimization problem. However, each of them has its own advantages and disadvantages.

The local search algorithm is simple and it can be easily adapted to the optimization process. But it can not guarantee the solution quality even for the near optimal one because of the local optimum trap. Thus, the local search algorithm is not taken into consideration according to the optimization goal.

The tabu search algorithm uses the tabu list as the memory structure to guide the searching process. By forbidding the recorded solutions, it can exclude the previously searched space to avoid the searching repetitions. As discussed in Section 2.2.2, once the algorithm traps in the local optimum, worser solutions can be selected by the tabu move. And the aspiration criterion helps the algorithm move to the new searching spaces. However, the tabu list size is the essential parameter and it can affect the solution quality and the algorithm performance. If the list size is too small, the algorithm may be trapped in a searching loop. In this case, the algorithm will stick to the local optimum trap. Because of the small tabu list size, the local optimal solution is recorded for only a short period and it is released before the algorithm can move to a better searching space. If the list size is too large, the searching time will be quite long and it may be not acceptable in practice. Besides the tabu list size, the solution quality of the algorithm is also dependent on the initial solution. And selecting a proper initial solution is a non-trivial work. Another drawback of the tabu search algorithm is the dead lock problem. In some extreme cases, there is no solution satisfies the aspiration criterion in the current solution's neighborhood. And all the generated neighboring solutions are recorded in the tabu list. Then no solution will be selected and the algorithm can not continue.

As discussed in Section 2.2.3, the simulated annealing algorithm can deal with the local optimum trap properly. And it is not sensitive to the initial solution. Because at the early stage of the algorithm, the metropolis criterion selects the solution randomly due to the high temperature. Even if the initial solution is already good enough, the algorithm may move far away from it. With the slow decrease of the temperature, the algorithm moves to the optimal searching region step by step. Nevertheless, the initial temperature is one significant parameter of the simulated annealing. It is supposed to be high enough to make the metropolis criterion accept solutions randomly. If it is too low, the algorithm behaves similarly to the local search algorithm and it may be trapped in local optimal solutions. If it is too high, an amount of time is wasted for the random searches. Another factor affects the solution quality of the simulated annealing is the cooling schedule which is reduction of the temperature. If temperature decreases too fast, the behavior of the local search occurs much earlier before the global optimal region is reached. If it decreases too slowly, the searching time will be quite long. Also, the termination conditions related to the nested loops play an important role in the simulated annealing algorithm. If the inner loop terminates too soon, the searching space is not fully explored. And if the outer loop ends too early, the acceptance probability is rather high. The final solution may be not

good enough. If both loops terminate too late, the algorithm may not be efficient enough for the optimization problem.

Compared with the tabu search algorithm, there are more parameters should be set up very carefully in the simulated annealing algorithm. But there is a variety of existing mechanisms to adjust them properly. Finding a suitable initial solution for the tabu search algorithm will increase the complexity of the optimization process. And it may require the pre-analysis of the solution space. Furthermore, the dead lock problem of the tabu search algorithm may result in the algorithm failure, which is risky for the users of this algorithm. After taking all the aspects into consideration, the simulated annealing algorithm is more promising than the tabu search algorithm. And it is proposed for the memory power optimization with memory partitioning method. At this stage, the fist goal of this thesis work is achieved.

3 Related Work

The memory partitioning method is widely used for the memory power optimization problem. Many researchers have obtained good results and benefits through the usage of this approach [SER16; BMP00; Mai+07]. The detail development of the mathematic power model discussed in Section 2.1 is illustrated in [SER16]. In this article, M. Strobel et al. propose the the ILP approach to be used in the optimization of the memory partitioning. And the results from their experiments show that the usage of ILP can yield an optimal configuration for their problem set. Since the memory partitioning is one kind of the combinatorial optimization, heuristics can also be a potential approach for it.

The combinatorial optimization is one of the hottest topics which aims to search for an optimal solution in a finite solution space. Most pf the conventional methods such as exhaustive search is not a proper approach for it. S. Kirkpatrick et al. find that there existing a close relationship between the statistical mechanics and the combinatorial optimization [S K83]. Based on this finding, they propose the simulated annealing algorithm as a promising approach for the combinatorial optimization in [S K83]. They introduce the metropolis criterion and explain how it can be applied to improve the solution quality provided by local search algorithm. In the article, the simulated annealing process is developed to be consisted of a parameterized framework which is already discussed in Section 2.2.3. To illustrate the usability of the simulated annealing algorithm, S. Kirkpatrick et al. deploy the algorithm in the physical design of computers to optimize the circuits partitioning, placement and wiring processes. In addition, they also apply the simulated annealing algorithm to the classical traveling salesmen problem. Multiple experiments for above optimization problems are conducted in this article and the experiment results are used to show that good solutions can be obtained by the usage of the simulated annealing algorithm. It is also pointed out by S. Kirkpatrick et al. that the accuracy and efficiency of the algorithm are much dependent on its parameters. Their suggestions for the algorithm setting are discussed together with other's work later in this chapter.

In [Joh+89], the author implement the simulated annealing algorithm for the graph partitioning problem. And a deep evaluation for the algorithm is made through a compact series of experiments in the article. They propose an alternative for the design of the cooling schedule and compared it with the original method using in [S K83]. The cooling schedule implemented by S. Kirkpatrick et al. is to reduce the temperature linearly by a colling ration. In [Joh+89], the author develop an adaptive cooling schedule which slows down the temperature reduction when the solution cost is changing fast in the current searching region. However, no improvement to the linear cooling schedule is found in their conducted experiments. For the terminations of the nested loops, they terminate the inner loop when the maximum number of iterations are achieved. And a low threshold of the

acceptance probability is used for the termination of the outer loop. Unfortunately, there is no method proposed to the determination of the initial temperature in [Joh+89].

In [S K83], S. Kirkpatrick et al. suggest two methods to determine the initial temperation T_0. The first one is to use the maximum difference between two neighboring solution costs as the initial temperature. The other one is described as following. An initial acceptance probability P_0 is defined which is the expected acceptance probability at T_0. Its value should be a real number close but less than 1, typically in the range of 0.8 to 0.95. Then a random guess of T_0 is made and the inner loop of simulated annealing is performed at this temperature. The solution acceptance ration of the inner loop performance is measured. And if the acceptance ration is lower than P_0, the value of T_0 is doubled. The same process is repeated until the measured acceptance ration is higher than P_0. Based on the second suggest of S. Kirkpatrick et al., Johnson et al. provide Equation 3.1 in [Joh+91] for the estimation of the initial temperature.

$$T_0 = -\frac{\overline{\Delta E}}{\ln P_0} \tag{3.1}$$

The $\overline{\Delta E}$ in the equation is the average difference between two neighboring solution costs. The value of $\overline{\Delta E}$ can be measured by generating a set of positive solution acceptances. Another determination is proposed by the authors of [Whi84]. They estimate the initial temperature T_0 by using Equation 3.2, where "K is a fixed number in the range of 5 to 10 and σ_∞^2 is the second moment of the cost distribution when the temperature is infinite "[Ben04]. However, this approach requires the analysis base on the pre-knowledge of the solution cost distribution.

$$T_0 = K\sigma_\infty^2 \tag{3.2}$$

4 Simulated Annealing for Memory Power Optimization

5 Evaluation

6 Conclusion

A Appendix

Appendix goes here...

B List of Figures

C List of Algorithms

D List of Tables

E List of Abbreviations

MPSoC Multiprocessor System-on-Chip

Bibliography

[Ben04] Walid Ben-Ameur. “Computing the Initial Temperature of Simulated Annealing”. In: *Computational Optimization and Applications* 29.3 (2004), pp. 369–385. ISSN: 1573-2894. DOI: 10.1023/B:COAP.0000044187.23143.bd. URL: http://dx.doi.org/10.1023/B:COAP.0000044187.23143.bd.

[BMP00] L. Benini, A. Macii, and M. Poncino. “A recursive algorithm for low-power memory partitioning”. In: *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*. July 2000, pp. 78–83. DOI: 10.1145/344166.344518.

[Glo89] Fred Glover. “Tabu Search—Part I”. In: *ORSA Journal on Computing* 1.3 (1989), pp. 190–206. DOI: 10.1287/ijoc.1.3.190. eprint: http://dx.doi.org/10.1287/ijoc.1.3.190. URL: http://dx.doi.org/10.1287/ijoc.1.3.190.

[Glo90] Fred Glover. “Tabu Search—Part II”. In: *ORSA Journal on Computing* 2.1 (1990), pp. 4–32. DOI: 10.1287/ijoc.2.1.4. eprint: http://dx.doi.org/10.1287/ijoc.2.1.4. URL: http://dx.doi.org/10.1287/ijoc.2.1.4.

[His05] Jason D. Hiser. “Effective Algorithms for Partitioned Memory Hierarchies in Embedded Systems”. AAI3169653. PhD thesis. Charlottesville, VA, USA, 2005. ISBN: 0-542-05748-4.

[Joh+89] David S. Johnson et al. “Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning”. In: *Operations Research* 37.6 (1989), pp. 865–892. DOI: 10.1287/opre.37.6.865. eprint: http://dx.doi.org/10.1287/opre.37.6.865. URL: http://dx.doi.org/10.1287/opre.37.6.865.

[Joh+91] David S. Johnson et al. “Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning”. In: *Oper. Res.* 39.3 (May 1991), pp. 378–406. ISSN: 0030-364X. DOI: 10.1287/opre.39.3.378. URL: http://dx.doi.org/10.1287/opre.39.3.378.

[Mai+07] Songping Mai1 et al. “An application-specific memory partitioning method for low power”. In: *2007 7th International Conference on ASIC*. Oct. 2007, pp. 221–224. DOI: 10.1109/ICASIC.2007.4415607.

[MBP02] Alberto Macii, Luca Benini, and Massimo Poncino. *Memory Design Techniques for Low Energy Embedded Systems*. Springer Science & Business Media, 2002.

[S K83] M. P. Vecchi S. Kirkpatrick C. D. Gelatt. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680. ISSN: 00368075, 10959203. URL: http://www.jstor.org/stable/1690046.

[SER16] Manuel Strobel, Marcus Eggenberger, and Martin Radetzki. “Low power memory allocation and mapping for area-constrained systems-on-chips”. In: *EURASIP Journal on Embedded Systems* 2017.1 (2016), p. 2. ISSN: 1687-3963. DOI: "10.1186/s13639-016-0039-5". URL: http://dx.doi.org/10.1186/s13639-016-0039-5.

[Whi84] Steve R White. “Concepts of scale in simulated annealing”. In: *The Physics of VLSI*. Vol. 122. 1. AIP Publishing. 1984, pp. 261–270.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.

Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.

Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.

Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 02.12.2016

Declaration

I hereby declare that the work presented in this thesis is entirely my own.

I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.

Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.

The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart, 02.12.2016