Hardware/Software Partitioning Algorithm under Multi-Constraints for the Optimization of Power Consumption

Abstract: An improved particle swarm algorithm with multiple neighborhood optimizations was proposed for hardware/software partitioning of system on chip under multiple constraints. A method for fitness calculation was designed with the unity of multiple constraint conditions and objective function. The particle position was updated by the average of the best position for all particles, the individual optimal location and the global optimal position. The effect of the average of position on particle position was adjusted adaptively according to the number of iterations. The multiple neighborhood of the optimal solution was searched for a better solution. The variation information of the optimal position was randomly produced by the Gaussian function. An arbitrary particle in population would be replaced by the variation particle whose fitness was better than the optimal fitness. The experimental results show that the proposed algorithm achieves hardware/software partitioning with lower power in a shorter searching time under the same constraints.

Keywords: Hardware/Software Partitioning; System on Chip; Particle Swarm Optimization; Multi-neighborhood Optimization

1 Introduction

The complexity of system functions is increased by the diverse application requirements, while the operation efficiency of the reconfigurable system on chip (SoC) is improved by the combination of the processor on chip and reconfigurable FPGA resources. Hardware/software partitioning of system has a significant impact on system performance. How to obtain an optimal performance of the hardware/software partitioning solution under system constraints has become a research hotspot [1,2].

A variety of hardware and software partitioning algorithms have been proposed, such as dynamic programming, integer programming, the genetic algorithm, simulated annealing, tabu search, and the particle swarm optimization algorithm.
A hardware/software partitioning algorithm based on genetic algorithm (GA) is presented for partial dynamic reconfiguration of system-on-chip in [3]. A partitioning algorithm of embedded system based on genetic particle swarm optimization is proposed in [4], which introduces crossover and mutation of the genetic algorithm into the particle velocity updating and improves the global search ability of the algorithm. However, the execution time of the proposed algorithm becomes longer because of the complexity of the particle update operation. In [5] a greedy algorithm is introduced into the simulated annealing to accelerate the convergence of the algorithm, reducing the search time for partitioning. An adaptive partitioning algorithm based on chaos genetic annealing (ACGSA) is presented in [6] under multi-performance index constraints. An objective function is designed with different proportions of punishment based on the multiple constraints to reduce the system power effectively. A hardware/software partitioning based on mixed integer linear programming is proposed for the region-based partial dynamic reconfigurable FPGA in [7]. A graph reduction technique is proposed to reduce the design space for HW/SW partitioning without sacrificing the partition quality [8].

Some scholars resolve the partitioning problem by transforming it into other issues. A heuristic approach which treats the HW/SW partitioning problem as an extended 0–1 knapsack problem is presented in [9] to minimize the hardware cost under software and communication constraints, and tabu search is used to further improve the solution obtained by the proposed heuristic algorithm. In [10], three algorithms for the multiple-choice hardware-software partitioning with the objectives of minimizing execution time and power consumption under the area constraint are discussed: the heuristic and tabu search algorithms for proximate solutions as well as the dynamic programming algorithm for exact solutions.

The algorithms mentioned above have achieved some optimization effect, but some of them require a long time to search for optimization [3,4, ], some do not consider the performance constraints [8], while others only consider one constraint [5,9,10], thus a poor efficiency and quality of partitioning. We present a hardware/software partitioning algorithm for power optimization of hardware/software partitioning under multiple constraints, design the fitness calculation with the unity of objective function and multiple constraints, and introduce an improved particle swarm optimization into partitioning algorithm to save the system power and reduce the searching time effectively under multiple constraints.

2 Hardware/Software Partitioning System Models

For reconfigurable SoC system hardware and software partitioning, the software functions are realized by a processor (soft or hard core) on a chip, and the processor area is a certain fixed value. The hardware part is implemented by the field programmable gate array (FPGA) logic modules, whose area is represented by the
number of logical units for the module occupied. For FPGA-based SoC systems, the system function modules which are implemented in hardware or software result in corresponding power consumption, occupied area, development cost and execution time.

In this paper, the system model is described with the task data flow graph (TDFG), and the communication overhead between tasks is combined into the execution time of the task node, which simplifies the system model. There are four function parameters: the system power, the system area, the development cost and the execution time for each task node. For a system with \( n \) functional modules, it is assumed that \( (b_1, b_2, \ldots, b_n) \), \( b_i \in S \) represents a feasible solution to hardware/software partitioning. \( b_i = 0 \) and \( b_i = 1 \) represent the task is implemented in hardware and software, respectively. The power optimization problem of hardware/software partitioning can be described as

\[
\begin{align*}
\text{min} & \quad \text{Power} \\
\text{s.t.} & \quad \sum_{i=1}^{n} (cs_i b_i + (1 - b_i)ch_i) \leq \text{Cost}_{\text{limit}} \\
& \quad \sum_{i=1}^{n} (1 - b_i)ah_i \leq \text{Area}_{\text{limit}} \\
& \quad \sum_{i=1}^{n} (ts_i b_i + (1 - b_i)th_i) \leq \text{Time}_{\text{limit}} \\
& \quad b_i \in S
\end{align*}
\]

(1)

For the \( i_{th} \) functional module, \( cs_i \) and \( ch_i \) represent the development cost implemented with software and hardware, respectively. \( ah_i \) stands for the hardware area, and the software area is the processor area, which is constant in the process and thus is set to zero in the area calculation. \( ts_i \) and \( th_i \) represent the execution time of software and hardware, respectively. \( \text{Cost}_{\text{limit}}, \text{Area}_{\text{limit}} \) and \( \text{Time}_{\text{limit}} \) represent the constraint values of the development cost, the system area and the execution time, respectively.

3 Particle Swarm Partitioning Algorithm With Multiple Neighborhood Optimization

3.1 Particle Position Update

The base particle swarm optimization (PSO) is a global optimization algorithm analogous to the food-searching behavior of birds, but it easily falls into local optimum. A quantum-behaved particle swarm optimization algorithm is introduced to achieve a better result in [11]. To accelerate the particle status update, the particle position is updated by removing the particle flight speed parameter and directly calculating the mean best position of all particles, the global and local best positions.
In this paper, the particle position update formula is improved for hardware/software partitioning. With the increment of the number of iterations, the impact of the mean best position of all particles on the particle is reduced, so that all particles tend to be close to the best location. The particle position is updated by

\[
p_{id}(k+1) = (\phi \times p_{best_{id}} + (1-\phi)g_{best_d}) \\
\hspace{1cm} + \beta \times |m_{best} - p_{id}(k)|
\]

s.t. \( \phi = \text{rand}(); \)
\( \beta = 1 - 0.5 \times \frac{k}{N}; \)
\( m_{best} = \frac{1}{M} \sum_{i=1}^{M} p_{i}; \)
\[
= \left( \frac{1}{M} \sum_{i=1}^{M} p_{1i}, \frac{1}{M} \sum_{i=1}^{M} p_{12} \ldots \frac{1}{M} \sum_{i=1}^{M} p_{id} \right).
\]

(2)

where \( k \) and \( N \) are the current number and the maximum number of iterations, respectively. \( \phi \) is a random value of uniformly distributed in \([0,1]\). \( \beta \) is a coefficient of expansion and contraction, which is decreased adaptively from 1 to 0.5 according to the number of iterations. \( M \) stands for the size of the particle population, and \( m_{best} \) represents the mean best position of all particles. \( p_{id}, p_{best_{id}} \) and \( g_{best_d} \) represent the current position, the individual and the global best position for the \( d_{th} \) dimension of the \( i_{th} \) particle, respectively.

### 3.2 Fitness Function Calculation

The system constraints are included in the objective function, which simplifies the decision process for constraints. The optimization under system constraints can be obtained by optimizing the objective function value directly in (6). In this paper, the method is improved to reduce some custom parameters, thus less impact of human factors on the partitioning results. The optimal power for hardware/software partitioning problem under multiple constraints in (1) can be converted into a specific objective function

\[
\min(\text{Power}) = A \times (f(x_A) + f(x_T) + f(x_C)) + B \times g(x_P)
\]

(3)

There are two parts of the objective function: the system constraint information and the system optimization performance information.

In the function, \( f(x) \) and \( g(x) \) denote the system constraint penalty function and the objective performance function, which are specifically defined in (4) and (5), respectively. \( x_A, x_T, x_C \) and \( x_P \) represent the area consumed, the execution time, the development costs and the power consumption for all the system modules. \( A \)
and $B$ determine the ratio of the two parts of the objective function. The feasibility of the design is determined by the system performance demand, so $A$ should be greater than $B$.

$$f(x) = \exp\left(\alpha \times \frac{x-x_L}{x_{\text{max}}-x_{\text{min}}} \right)$$

s.t. $\alpha = \begin{cases} 1 & \text{if } x \leq x_L \\ K & \text{otherwise} \end{cases}$

$$x_L = x_{\text{min}} + \mu(x_{\text{max}} - x_{\text{min}})$$

$0 < \mu < 1$

(4)

$$g(x) = \frac{x-x_{\text{min}}}{x_{\text{max}}-x_{\text{min}}}$$

(5)

where $x, x_L, x_{\text{max}}$ and $x_{\text{min}}$ represent the performance parameters of system functions implemented (power, area, cost and time) corresponding to the current value, constraints, maximum and minimum values, respectively. $x_L$ is a value between the minimum and maximum values, which can be regulated by the constraint factor $\mu$. The different performance indexes may not be bound by the same factor. $\alpha$ is a penalty factor. $K$ is a constant greater than 1, which is set to be 100 in the experiment. All parameters should be normalized because of different performance parameters at different magnitudes. The exponential function based on $e$ and the penalty factors are used to make a larger penalty to the solution which does not meet the constraints, while the feasible solution is spared.

Among solutions that meet the constraints, the constraint value is much smaller than the value of default constraints, but the power consumption is not optimal. For example, with the same area, shorter running time comes along with higher power consumption. In addition, the module performance of the hardware modules implemented with FPGA is primarily improved through the parallel structure, which means that the parallelism is improved by increasing the system area and reducing the running time to reduce the power consumption. So the optimal partitioning is the one that exactly meets the system constraints, when the value of the objective function is the minimum.

According to the calculation value of the objective function, the calculation result that does not satisfy the constraints will be greater than the one that satisfies the constraints. The smaller the function value, the lower power the solution obtained consumes. Therefore, this paper takes the objective function as the fitness function of the particle to search for a particle with the smallest fitness, which is hardware and software partitioning solution with the lowest power.
3.3 Multi Neighborhood Optimization For Optimal Value

For better accuracy and convergence rate of the optimal solution, multiple neighborhoods of optimal value are searched for a better solution. One of the neighborhood positions is changed by the Gaussian random function each time, so a neighborhood particle of optimal value is obtained. The specific procedures are as follows.

Step 1: A randomly-generated mutated position $i$ is changed by using the Gaussian distribution function to obtain a new particle $p_{x_k}$ $(k=1,2,\cdots,R)$.

Step 2: The fitness $x_{fit_k}$ of the particle $p_{x_k}$ is calculated using (3). If $x_{fit_k}$ is smaller than the best fitness $g_{fit}$, one particle of the population will be replaced by $p_{x_k}$ randomly to increase the population diversity.

Step 3: The operations above are executed iteratively, until $k = R$.

Step 4: Assume $x_{fit}^* = \min\{x_{fit_1}, x_{fit_2}, \cdots, x_{fit_S}\}$, for $x_{fit}^* < g_{fit}$, $g_{fit} = x_{fit}^*$, $g_{best}$ will be updated as the particle position.

3.4 Algorithm Process

The proposed multi-neighborhood particle swarm optimization (MNPSO) partitioning algorithm is improved based on the quantum-behaved particle swarm optimization algorithm, searching multi-neighborhood of optimal solution to improve the local searching ability of the algorithm. The specific steps of algorithm are as follows,

Step 1: Population initialization. Each particle is composed randomly of 0 and 1, which are produced by the random function $rand()$ for 2 modulo arithmetic.

Step 2: The performance parameter information of system task is read from the task file. Each particle is combined with every task to obtain the performance parameters.

Step 3: The fitness of each particle is computed by (3), and the individual best position $p_{best}$ and the global best position $g_{best}$ are selected.

Step 4: If the end conditions (the maximum number of iterations or the number of the optimal value which hasn't changed) are met, go to Step 8. Otherwise, go to Step 5.

Step 5: The mean position of all the particles $m_{best}$ is calculated.

Step 6: The global best solution $g_{best}$ is searched for multi-neighborhood by multi-neighborhood optimization algorithm.

Step 7: The particle position is updated using (2), and the fitness of each particle is computed by (3). $g_{best}$ is updated. Return to Step 4.

Step 8: The hardware/software partitioning result with optimal power is obtained, and the position information of $g_{best}$ is output.
4 Experimental Analysis

In this section, the experimental results of the proposed approach are presented. The MNPSO algorithm is implemented in C language to evaluate the algorithm performance. Our approach is compared with the approaches GA, PSO and ACGSA by several experiments. There are 8 randomly generated TDFGs (using the TGFF tool) in which the number of nodes varies from 60 to 2000. For each graph, there are 4 different values which represent the power, area, cost and time for each task node. The parameters of objective functions A and B are set to be 0.6 and 0.4, respectively. The experiments are conducted on a PC with an Intel i7-2640 2.8GHz CPU and 4GB RAM main memory. Our simulator is developed in Microsoft VS2010 on the Windows 7 32bit operating system.

Random tasks of 200 nodes are tested by the MNPSO algorithm for 100 times to show the effect of different constraints on the partitioning results. The constraint factor of the development cost is set to be 0.8. The constraint factor of the system area grows from 0.125 to 1.0, and the constraint factor of the execution time increases from 0.2 to 0.8. The experimental results are shown in Table I. The symbol “√” indicates that the solution that satisfies constraints can be found each time, and the symbol “×” indicates that there is no solution that meets the constraints. The numerals in the table represent the number of found solutions that satisfy the constraints among 100 searches, as shown in Table 1.

Table 1. Effect of different constraints on the partitioning results

<table>
<thead>
<tr>
<th>Constraints</th>
<th>$\mu_T (1/5)$</th>
<th>$\mu_T (2/5)$</th>
<th>$\mu_T (3/5)$</th>
<th>$\mu_T (4/5)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_A (1/8)$</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$\mu_A (2/8)$</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$\mu_A (3/8)$</td>
<td>×</td>
<td>×</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>$\mu_A (4/8)$</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>$\mu_A (5/8)$</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>$\mu_A (6/8)$</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>$\mu_A (7/8)$</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>$\mu_A (1)$</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

The more stringent the constraints, the lower probability for solutions that can meet constraint conditions to be found. For the same time constraint factor $\mu_T (4/5)$, there is a probability of 59% to obtain a solution under the area constraint factor $\mu_A (3/8)$, but there is no solution that satisfies the constraints under some smaller area.
constraints (indicated by "×"), and under a larger area constraints, the probability is 100% to find a solution that meets the constraints (indicated by "√").

The GA, PSO, ACGSA and MNPSO algorithms are implemented with several tests for the power optimization of hardware/software partitioning under the time constraint factor $\mu_T (4/5)$ and the cost constraints factor $\mu_C (4/5)$. Each test runs 100 times to calculate the average power consumption of optimal partitioning solution, as shown in Table 2.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Max Power</th>
<th>Area constraint factor $\mu_A (1/2)$</th>
<th>Area constraint factor $\mu_A (3/4)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>PSO</td>
<td>ACGSA</td>
<td>MNPSO</td>
</tr>
<tr>
<td>120</td>
<td>41.63</td>
<td>41.33</td>
<td>41.33</td>
</tr>
<tr>
<td>200</td>
<td>74.85</td>
<td>73.12</td>
<td>72.80</td>
</tr>
<tr>
<td>300</td>
<td>123.32</td>
<td>119.34</td>
<td>118.57</td>
</tr>
<tr>
<td>500</td>
<td>213.45</td>
<td>207.15</td>
<td>205.33</td>
</tr>
<tr>
<td>800</td>
<td>452.68</td>
<td>354.29</td>
<td>351.08</td>
</tr>
<tr>
<td>1000</td>
<td>610.37</td>
<td>462.60</td>
<td>458.54</td>
</tr>
<tr>
<td>2000</td>
<td>1516.68</td>
<td>1072.56</td>
<td>1062.65</td>
</tr>
</tbody>
</table>

When the task system is small-scale (less than 300 nodes), the power of partitioning results are obtained by the four algorithms are close. With the increasing size of the system, the difference between the algorithms is gradually displayed. When the area constraint factor is stringent ($\mu_A (1/2)$), the GA falls into the local optimal value more easily than the other three algorithms. When the area constraint factor is loose ($\mu_A (3/4)$), the GA algorithm is converged prematurely due to the lack of local searching ability, while the PSO algorithm obtains a solution with lower power consumption due to the larger flying space of the particle. The local search ability of ACGSA is enhanced by introducing the chaos, annealing, and other optimization strategies, thus better results by the ACGSA. But the proposed MNPSO algorithm achieves a even better solution with lower power consumption than the other three algorithms under different sizes and different constraints of system, for searching multi neighborhood of the optimal solution. With the increase of the system size, more power consumption is reduced. Further, since the power consumption of FPGA is reduced by increasing the system area, a smaller area constraint leads to a higher power consumption, a fact verified by the results shown in Table 2.

In order to analyze power consumption saving, the power consumptions in Table 2 are compared with the maximum power of the same system size under the
area constraints factor $\mu_A (3/4)$, as shown in Figure 1. With the increase of the system size, the power consumption of hardware/software partitioning solution obtained by each algorithm is continuously decreased. For the task system with 60 task nodes, the GA and PA achieve power reductions of 8.8% and 9.2%, respectively, and the other two algorithms reduce power consumption by 9.9%. For the task system with 500 task nodes, compared with the maximum power, the power obtained by the GA, PSO, ACGSA and MNPSO algorithm are much lower, saving 22.3%, 24.9%, 28.6%, and 29.1% power, respectively. For the task system with 1000 task nodes, the power by all the four algorithms is reduced more than 28%, with the MNPSO achieves the greatest reduction of power (36.9%). When the task nodes are increased to 2000, the power by the four algorithms is reduced more than 32%, with the MNPSO algorithm reducing by up to 44.2%. Compared with the ACGSA, GA and PA, the MNPSO achieves power reduction increases of 0.5%, 9.4% and 11.7%, respectively.

According to the algorithm time complexity analysis, the computational complexity of the GA, PSO, ACGSA and MNPSO are $O(N \times n \times m)$, where $N$ is the number of loop iterations, $n$ is the population size, and $m$ represents the number of task nodes. Since the crossover and mutation operation of genetic algorithm are executed for the entire population, the searching time of the GA is approximately twice the time of the PSO. The chaos and the annealing optimization algorithms are introduced into the ACGSA based on the genetic algorithm which takes the longest time for searching. A better partitioning solution is obtained at the expense of the search time. Although the multi-neighborhood searching of the optimal solution is introduced into the MNPSO algorithm in this paper, its computational complexity is much smaller than the population size, so that the searching for the optimal solution is significantly accelerated. The searching time of the proposed algorithm is less than the time of the other three algorithms.

![Figure 1. Ratio of saving power consumption for the partitioning algorithm results.](image-url)
This paper compares the searching time of four algorithms under the area constraint factor $\mu_A(3/4)$, as shown in Figure 2. With the growth of the system size, the algorithm searching time increases, where the searching time of the ACGSA grows the fastest, and that of the MNPSO grows the slowest.

When the task system is small-scale (less than 300 nodes), except the ACGSA, the time of the other three algorithms is very closer, i.e., less than 1s. For 300 task nodes, the searching time of the ACGSA is the longest, i.e., more than 1.7s, and the time of GA is close to 1s, while those of the PSO and the MNPSO are less than 0.5s and 0.3s, respectively. Although the ACGSA has better optimum performance, it takes a long time to search. Less searching time is needed by the proposed MNPSO algorithm due to the strong astringent properties, with a longest searching time of less than 2s.

According to the obtained experimental results, the MNPSO achieves the optimal partitioning solution most quickly. For the system with 2000 task nodes, when the factors of the time constraints and the cost constraints are 0.8, and the area constraint factor is $\mu_A(3/4)$, the MNPSO achieves a 44.2% power reduction in 1.9s, while by GA it takes three times as long to achieve a 32.5% power reduction, by PSO it takes 2.3 times as long to achieve a 34.8% power reduction, and by ACGSA it takes 4.1 times as long to achieve a similar power reduction, which is still 0.5% lower than that obtained by the MNPSO. Therefore, in terms of the optimal power for hardware/software partitioning search algorithms, the MNPSO algorithm is a time-saving partitioning algorithm.
5 Conclusion

This paper proposes a multi neighborhood particle swarm optimization algorithm for the optimal power of hardware/software partitioning under multiple constraints. The mean best position of all particles is introduced into the particle position status updating, and the optimal solution is searched for better solution in multi-neighborhood to expand the local search capacity. The Gaussian distribution is used to generate mutated position information and increase the population diversity. A multi-constraint penalty function is integrated into the calculation of the objective function, speeding up the searching of partitioning solution under the system constraints. Experimental results show that the algorithm effectively achieves hardware and software partitioning for power optimization. Compared with the GA, PA and ACGSA, the proposed algorithm achieves a better partitioning solution with lower power consumption in shorter time under multiple constraints.

Acknowledgment: This work is supported in part by the national natural science foundation of China under Grant No. 61474087, and the Fundamental Research Funds for the Central Universities under Grant No. JB150315.

References


