Hardware/Software Partitioning of Combination of Clustering Algorithm and Genetic Algorithm

Weijia Li, Lanying Li*, Jianda Sun, Zhiqiang Lv and Fei Guan
The College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China

Abstract

As to the embedded system of multi-tasks, this paper proposes a hardware/software partitioning algorithm based on Clustering algorithm using reference and density and Genetic algorithm. The algorithm uses reference task to reflect attribute of system task node, and firstly finish classification of task nodes to decrease the scale of the system, and then partitions hardware/software based on genetic algorithm. The experiments results have showed that this algorithm can accelerate converge to appropriate solution of complex system with more tasks.

Keywords: hardware/software partitioning; genetic algorithm; clustering algorithm; reference and density

1. Introduction

One key step of Hardware/Software Co-design for embedded system is hardware/software partitioning [1], which is NP-complete problem. In 1992, Hardware/software partitioning algorithm was invented which was aimed at the exploration of automatically space design. Then people invented all sorts of partitioning algorithms [2], among them, the heuristic algorithm is classic for its better application effect, which contains Hill Climbing, Ant Algorithm, Tabu searching, simulated annealing, genetic algorithm, etc. Yet, when facing massive complex problems, these algorithms all have limits, and the rate of convergence is slow in the late stage which eventually make the algorithm’s time too long and more easily get stuck into local optimum.

In this paper, we combine cluster method with hardware/software partitioning. Firstly, we divide the task nodes into several big notes according to all the attributes of the task node in order to decrease the scale of the system. Then we use the classic genetic algorithm to do the hardware/software partitioning of these big nodes.

2. Model of Hardware/Software Partitioning

2.1 Features of model

Definition 1.(K-way partitioning problem) for module set \( M = \{m_1, m_2, ..., m_n\} \), K-way partitioning problem is finding the Cluster’s set \( P = \{p_1, p_2, ..., p_k\} \) which satisfies formula (1).

The solution of K-way partitioning problem is normally to optimize object function under
some constraints. When \( k = 2 \), it is called as double-way partitioning problem. When \( k > 2 \), it is called as multi-way partitioning problem [3].

\[
\begin{align*}
p_i & \subseteq M, 1 \leq i \leq k, \\
\bigcup_{i=1}^{k} p_i = M, p_i \cap p_j = \emptyset \\
1 \leq i, j \leq k, i \neq j
\end{align*}
\]

(1)

In this paper, we use double-way partitioning in our research on hardware/software partitioning, its model has following features:

1. Specified system has a embedded processor (software execution unit) and a ASIC、 a FPGA or other hardware (hardware execution unit), the system’s function unit could only be reflected on one of them.

2. Because the speed of hardware’s execution is much faster than software’s, the execution time of functional element which accomplished by hardware is regarded as 0.

3. If two functional elements which communicate with each other are mapped to different realization, then the cost of the communication is calculated.

The biggest advantage of this model is that schedule doesn’t need to be given obviously. The hardware doesn’t need to be scheduled is because the execution time of functional element is regarded as zero. For the software, the schedule can be ignored because there is only one processor, and the sum of every part of software’s execution time can be used as the whole execution time. So the partitioning problem can be separated from schedule problem.

2.2 Description of Hardware/Software Partitioning

In Hardware/Software Co-design, firstly the function of embedded system is described by C or other procedural language. Then the system-call graph can be extracted according to different granularity demand. The call graph usually adopts DAG. For example, Figure 1 is a 7-node DAG, \( \text{DAG} = (T, E) \) and \( T = \{t_0, t_1, \ldots, t_n\} \) is task-node set. The node stands for task, and \( E \) is directed-edge set. The directed edge stands for the cost of data communication among task nodes.

![Figure 1. 7-node DAG](image)

Task-node \( t_i \) is defined as the following triad: \( \{ts_i, cs_i, ch_i\} \). In that, \( ts_i \) and \( cs_i \) stand for the software time of task nodes and software cost of task nodes. \( ch_i \) stands for the hardware cost when task nodes use hardware. Moreover, each side of the DAG has an attribute: communication cost between node \( i \) and node \( j \). In this paper we regard communication cost as hardware cost. When \( P \) is a partitioning, and \( P \) is made of two parts of \( T \), which is
\[ P = (T_H, T_S), \quad T_H \cup T_S = T \text{ and } T_H \cap T_S = \phi, \quad E_p \text{ is a side set of } P, \text{ which is } E_p = \{(t_i, t_j) \} \{t_i \text{ and } t_j \text{ are tasks which have different realization ways}\}. \]

The cost of \( P \) is
\[ C_p = \sum_{i \in T_H} c_{x_i} + \sum_{i \in T_H} c_{h_i} + \sum_{(t_i, t_j) \in E_p} c_{c_{ij}}, \text{ and the time of } P \text{ is } T_p = \sum_{i \in T_H} t_{s_i}. \]

After having defined the target function of system, each increment of hardware node will lead to the decrement of the system delay. In the meantime, the cost will be increased. According to different limits, there are two basic problems: optimizing time under the limit of cost and optimizing cost under the limit of time. The Cluster Genetic Algorithm (CGA) presented in this paper uses optimizing cost under the limit of time, which is:

Minimize \( C_p \)

Subject to \( T \leq TimeReq \)

3. Partitioning Algorithm of Hardware/software

3.1 Clustering Algorithm

Clustering, as an individual tool, can find deep information of data distribution in database. It can also be a preprocessing step of other analysis algorithms. Clustering algorithm can divide the data into different clusters. Data in the same cluster is similar, while there are differences between data in different clusters. The introducing of Cluster to hardware/software double-way partitioning in the paper will reduces the complexity of system, which can also reduce algorithm’s executing time.

There are common clustering algorithms such as K-MEANS, CURE and C2P. K-MEANS now has been introduced into hardware/software partitioning, which is an algorithm based on dividing. It tries to find \( K \) clusters under one specific standard, which usually uses Square Error Standard. The time complexity of K-MEANS is \( O(ktn) \). \( N \) stands for the sum of data. \( K \) stands for the amount of clusters, and \( t \) stands for the times of the algorithm circulation. Usually it’s an efficient. However, the algorithm needs the value of \( K \), which is difficult to judge. And whatever value \( K \) takes, it will influent the optimal solution needed by the genetic algorithm in the late stage. We uses a fast clustering algorithm CURD(Clustering using reference and density) [4], which is based on reference node, and it has the same time complexity as K-MEANS algorithm, which means high efficiency.

3.2 Combination of Clustering Algorithm and Genetic Algorithm

The process of CGA is showed as Figure 2.

3.2.1 Normalization of the Parameter

The two characters value appropriating for clustering is worked out and normalized according to formula (4) and (5).

Ratio of hardware/software’s executing cost:
\[ w_i = \frac{c_{x_i} / c_{h_i}}{Max(c_{x_j} / c_{h_j})}, \quad j = 1, \ldots, N \]

Performance-price ratio of software execution: \( N \)
\[ w_{id} = \frac{c_s / t_s}{\text{Max}(c_s / t_s)}, \quad j = 1, \ldots, N \]  

\( N \) stands for quantity of the task nodes.

### 3.2.2 Pseudocode of CURD

Definition 2. (Adjacent Reference Task): Give distance Radius and threshold value \( t \), if reference task \( p \) and \( q \) satisfy \( \text{Dist}(p, q) \leq 2 \cdot \text{Radius} \), we call this two nodes as Adjacent Reference Task. \( \text{Dist}(p, q) \) stands for the distance between \( p \) and \( q \). According to the euclidean distance, \( \text{Dist}(p, q) = \sqrt{(w_{pc} - w_{qc})^2 + (w_{pd} + w_{qd})^2} \) when the task nodes is classified, the three attributes should be expanded to following \( 8 \)-tuple: \( (\text{Rpt}, \text{Rep}, \text{Size}, \text{Radi}, \text{Repant}, \text{Repr}, \text{Class}, \text{Canid}) \). Here, \( w_{pc} \) and \( w_{pc} \) represent the sum of \( \text{Rep} \) and \( \text{Radi} \) in the representative area of task-reference node, and \( \text{Size} \) represents the amount of task nodes in the representative area of task-reference node, and its initial value is 0.

![algorithm flow chart of main body of CGA](image1)

![flow chart of GA](image2)

**Figure 2. Flow Chart of CGA**

Picture (a) shows the procedure of CGA main body. We can see that after the process to the inflection of clustering and task-node by GA [5], we can finally get satisfying results.
Picture (b) shows the details of GA that new population can be received after calculating the fitness function, roulette selection, crossover operation and variation of the initial population [6].

**Finding_Task_Candidate_Reference** (TaskPointSet, dRadius, Iterate)

//input: TaskPointSet, distance: dRadius , circulation times: Iterate
//output: TaskCandidateReferenceSet
{S1: TaskCandidateReferenceSet = φ, I = 1;
 S2: While (I = Iterate )
 {S3: Single_Scan (TaskPointSet,dRadius, TaskCandidateReferenceSet)
  { step1: j=1
  step2: while( j ≤ TaskPointSet.Size)
  step3: for every task point TP_j:
    The coordinate information of TP_j is sent into every TaskCandidateReferenceSet R. If the distance to point TP_j ≤ dRadius , then the information of TP_j will be put into the candidate point, and change the information into R.wcs = R.wcs + TP_j.wjc , R.wds = R.wds + TP_j.wjd , R.N_r = R.N_r +1; On the contrary, if the distance between TP_j and dRadius ≤ all reference points, then create a new candidate reference point. Its information is : R.wcm = TP_j.wjc,R.wdm = TP_j.wjd,R.wcs = TP_j.wjc,R.wds = TP_j.wjd,R.N_r = 1.
    Step4: j++}
 S4: Regenerate_Task_Candidate_References (TaskCandidateReferenceSet)
 { Select the center point of TaskCandidateReferenceSet as new candidate point R’:
   R.wcm = R.wcs / R.Ns , R.wds = R.wds / R.Ns , R.wcs = 0 , R.Ns = 0 .
 S5: I ++ }
 S6: Single_Scan (TaskPointSet,dRadius, TaskCandidateReferenceSet) }

**Regenerate_TaskReferences** (TaskCandidateReferenceSet, t)

{compare every candidate task point Ns, if R.Ns ≥ t , then add it into TaskReferenceSet, else just drop it.}

**Mapping_References_and_TaskPoints** (TaskPointSet, TaskReferenceSet, dRadius)

//Output: TaskReferenceSet
{Step 1. Each reference, RTPs = φ, I = 1
 Step 2. While(1 ≤ TaskPointSet.Size){
  Step 3. For each Taskpoint TP_i :
  If the distance between TP_i and some reference R is equal to or less than dRadius .
    RTPs = RTPs ∪ {TP_i} .
  Otherwise, the distances between TP_i and all the references are larger than dRadius .
  TP_i is considered as new task-reference.
  Step 4. I ++
Classifying the references (ReferenceSet, dRadius)

//Output: ReferencesClusterSet
{If the distance of reference points R1 and R2 \( \leq \) double Radius, then R1 and R2 are nearby reference points. If not, they are two isolated points.
Now we can use undirected graph to describe new produced reference point set, the vertex of graph is reference point. There is an edge between the adjacent reference points, and reference points which are in the same connected subgraph make up a class. We can find vertexes which are in the same connected subgraph by BFS.) \( \sum ts_i, \sum cs_i \). At last, we build mapping between cluster and corresponding task point, then the similar task’s cluster will be gotten. According to the cluster’s result, we work out the total software’s execution time, software cost and hardware cost of each kind of task, which is \( \sum ts_i, \sum cs_i \) and \( \sum ch_i \). The partitioned \( k \) big task nodes can be used by classic genetic algorithm.

3.2.3 Genetic Algorithm

Genetic algorithm includes following steps [7, 8]:
1) Population Initialization, the population’s scale is \( 3K \).
2) Selection of Fitness Function. Change the execution time into penalty term in objective function. We bring in normalizing factor \( \chi_c \) and \( \chi_s \):
\[
\chi_c = \text{CostHw} - \text{CostSw}
\]
\[
\chi_s = \max(\text{TimeSw} - \text{TimeReq}, \text{TimeReq})
\]
CostHw/ CostSw stands for the whole cost when all present nodes are implemented by hardware/software. TimeSw stands for relevant software execution time, TimeReq stands for restricted hours provided by designer. Therefore, the generalized objective function can be defined as:
\[
\text{COBJ} = \alpha \exp \left[ \frac{T - \text{TimeReq}}{ \chi_s } \right] \frac{\left( T - \text{TimeReq} \right)}{\chi_s} + \beta \frac{C}{\chi_c}
\] (6)
Here, \( C \) and \( T \) stand for cost and time of given partition, and \( M_i \) is annealing coefficient in SA algorithm. Take \( M_0 = 1 \) and \( M_{i+1} = 0.98 * M_i \) in evolutionary process, and \( \alpha, \beta \) are used to adjust the proportion of \( T \) and \( C \).
Algorithm optimization is corresponding to minimum value of generalized function, and fitness function is defined as: \( \text{Fitness} = 1/(1+\text{COBJ}) \).
3) Coding problem. The binary coding is used here. Vector \( B = (b_1, \ldots, b_k) \) stands for a kind of system design, meanwhile, it’s also operation object chromosome of genetic algorithm. Here, \( b_i \in (0,1) \), 0 stands for using software, 1 stands for using hardware, and \( k \) stands for partitioned cluster task node.
4) Genetic operator
a) Selection:
Selection is the main strategy in genetic algorithm, which usually takes proportional selection operator. This operator is a kind of playback random sampling, which is based on roulette wheel at $k$ times, and every wheel can reach into progeny population.

b) Crossing:

Crossing is a main method to create new individual. We mainly adopt multi-point crossover, and the amount of intersection is in direct proportion to the amount of system node. We assume the node amount is $K$, and the number of intersection is $\lceil K/l \rceil$, i.e., selecting $\lceil K/l \rceil$ randomly points in two father individual, and then exchange the information which is corresponding to these two new individuals to create new individuals. Here $l$ is a parameter whose value needs to be given.

c) Variation:

The aim of variation is to improve inheritance’s searching ability, maintain diversity of group, and prevent from precocious phenomenon. We add a disturbance variable to mutation probability of each position of chromosome. That is if $b_i = 0$, then change it into 1 at probability $p_m$, If $b_i = 1$, then change it into 0 at probability $p_m$.

4. Experiment and Analysis

We compare Genetic Algorithm with CGA using the similar method of [9].

4.1 Experimental Samples and Experimental Environment

At present, there isn’t a test set accepted by in embedded system’s hardware/software partitioning worldwide. Randomly generated DAG is a common method, and attributes were given to the point and edge. The experiment taken uses TGFF to create 6 DAG with different nodes and relevant software execution time and software/hardware cost. The whole program is designed in Visual C. The experiment environment is processor Intel P4 2.8Hz, 512M memory. Considering the rationality of parameter, hardware execution time is 2-5 times faster than software execution time under random creating, and hardware limit occupies 2/3 of the whole hardware cost.

4.2 Parameter Setting

The algorithm has following parameters: distance dRadius, cycle time Iterate, $\alpha, \beta$ crossover rate $p_c$ and mutation rate $p_m$.

When dRadius is taken between $1/50-1/100$ of maximum range, the clustering result is better. When Iterate = 4 is allowed in this experiment, the bigger value it takes, the better it is, but the time cost will increase, too. In genetic algorithm, $\alpha = 0.6, \beta = 0.8$, crossover rate $p_c = 0.6$, mutation rate $p_m = 0.6$. 
4.3 Experiment Comparison

Table 1. Execution Time Comparison of Ga and CGA

<table>
<thead>
<tr>
<th>Number of node</th>
<th>Time Constraint</th>
<th>Cost</th>
<th>Execution Time/ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>9846</td>
<td>GA</td>
<td>2860</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGA</td>
<td>2945</td>
</tr>
<tr>
<td>80</td>
<td>15799</td>
<td>GA</td>
<td>4939</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGA</td>
<td>5003</td>
</tr>
<tr>
<td>120</td>
<td>23826</td>
<td>GA</td>
<td>7878</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGA</td>
<td>7939</td>
</tr>
<tr>
<td>200</td>
<td>41061</td>
<td>GA</td>
<td>12724</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGA</td>
<td>12688</td>
</tr>
<tr>
<td>400</td>
<td>81302</td>
<td>GA</td>
<td>26293</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGA</td>
<td>26420</td>
</tr>
<tr>
<td>800</td>
<td>164534</td>
<td>GA</td>
<td>52975</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGA</td>
<td>52784</td>
</tr>
</tbody>
</table>

Figure 3. Comparison of Execution Time of Two Algorithms

We can see from Table 1 and the Figure 3 that compared with genetic algorithm, the algorithm proposed in this paper doesn’t have too much advantage in searching value of objective function. However, it truly has advantages in the field of 4 algorithm’s execution time. It can be inferred that genetic algorithm’s execution time rise perpendicularly with the increase of number of task-node. However, algorithm in this paper is a procedure rising with small slope. Therefore, with the increase of system scale, the traditional heuristic algorithm such as genetic algorithm can’t work out a satisfying division, but algorithm in this paper has strong practicability in this aspect.

5. Conclusion

CGA algorithm absorbs the ideas of clustering to make genetic algorithm better in bigger-scale embedded system. It overcomes the shortcoming that algorithm’s execution time will be too long with the increase of the number of task-node, which achieves good results in
embedded system and software/hardware partitioning. Though it only have a little advantage in objective function, it’s speed has been significantly improved comparing to GA.

In target architecture, this experiment can minimize the system cost within a certain period of time. In fact, the constraints and optimization of other performance indexes (such as power dissipation) are also very important in embedded system and SoC software/hardware partitioning, and algorithm in this paper is also appropriate for this kind of partition problem.

We will research on the infection of different values of controls parameter over CGA’s algorithm performance to find out the right value of controls parameter which fits CGA software/hardware partitioning algorithm, and apply CGA to multiprocessor-partitioning problem.

Acknowledgements

Supported by National Collegiate Innovation and Entrepreneurship Training Program (No: 201310214013).

References
