Tabu Search with Intensification Strategy for Functional Partitioning in Hardware-Software Codesign

Theerayod Wiangtong, Peter Y.K. Cheung, Wayne Luk
	tw1@ic.ac.uk, p.cheung@ic.ac.uk, wl@doc.ic.ac.uk : Imperial College, London, UK

Abstract

This paper presents tabu search (TS) method with intensification strategy for hardware-software partitioning. The algorithm operates on functional blocks for designs represented as directed acyclic graphs (DAG), with the objective of minimising processing time under various hardware area constraints. Results are compared to two other heuristic search algorithms: genetic algorithm (GA) and simulated annealing (SA). The comparison involves a scheduling model based on list scheduling for calculating processing time used as a system cost, assuming that shared resource conflicts do not occur. The results show that TS, which rarely appears for solving this kind of problem, is superior to SA and GA in terms of both search time and the quality of solutions. In addition, we have implemented intensification strategy in TS called penalty reward, which can further improve the quality of results.

Tabu Search and Intensification Strategy

The major distinction of TS compared to other heuristic search methods is that it exploits data structures of the search history as a condition of the next moves [1]. Rather than randomly exploring the search space like the others, TS uses a short-term memory, named recency-based memory, to store recent search areas and uses a long-term memory, named frequency-based memory, to store frequency of searching in each area. Data in frequency-based memory are scaled by a positive constant $Q$ which is then used as penalty values.

The purpose of such penalty is to diversify future search into regions which are rarely visited. But here, we improve upon the idea of using penalty for both diversification and intensification in the following way: the region of the best solution in the last $K_q$ iterations is selected. For all neighbours in this region, instead of using constant $Q$ to compute the penalty factor, a new reward constant $Q' < Q$ is used. The region that earns the reward ($Q'$) is called the promising region and it will be updated in every $K_q$ iterations. As a result, neighbours belonging to the promising region in the next $K_q$ iterations have a higher chance to be selected because of the reduced penalty value. As will be seen later, TS with this intensification strategy, called penalty reward, provides better solutions compared to those using the standard TS algorithm.

Our Approach

We establish an approach that uses heuristic algorithms as partitioner. The partitioning results will be evaluated by a cost estimation model to obtain processing time (makespan). Estimating processing time is based on list scheduling, in which hardware tasks are scheduled without resource conflicts. In other words, our approach is a combination of partitioning and scheduling as shown in figure 1.

![Program's structure](image1)

**Figure 1.** Program’s structure

The processing time includes communication time, reconfiguration time, execution time on hardware or software, and waiting time for available shared resources or input data. Task allocations are obtained as a result of scheduling, and no resource conflict is guaranteed. Such an approach is applicable to coarse grain or functional partitioning and scheduling.

![Reference codesign architecture](image2)

**Figure 2.** Reference codesign architecture

The reference architecture used is shown in figure 2. We employ a shared memory to fully exploit tasks parallelism in our codesign system.
Comparisons with GA and SA

TS is compared with two other heuristic algorithms: GA and SA. The programs are written in C for a PC with a 750 MHz Duron processor. Randomly generated task graphs (RG) as well as graphs that are usually found in real applications such as in-tree (IT), out-tree (OT), fork-joint (FJ), mean value (MV) and FFT (FT), are used as inputs to the programs.

To obtain a fair comparison, standard structures of the algorithms are used, so at this stage the penalty reward scheme is not included in TS. Input parameters for each algorithm are carefully selected by several pre-simulations in order to ensure fair comparison because the performance of heuristic searches depends strongly on these parameters.

Table 1 compares the three algorithms applied to the various task graphs. The mean values are obtained over 20 experiments. As can be seen, TS produces solutions that are better than GA and SA in all cases and in the least amount of computation time. TS also scales better with the complexity of the problem.

Table 1. Processing time and search time from the algorithms in various types of graphs

<table>
<thead>
<tr>
<th>Genetic Algorithm</th>
<th>Mean Values Proc. Time</th>
<th>Best of Proc. Time</th>
<th>Mean Values Srch Time (sec)</th>
<th>Best of Srch Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RG 20/40</td>
<td>7916.9 5.67</td>
<td>7450 7757.7 5.90</td>
<td>7450 7470.4 3.36</td>
<td>7325</td>
</tr>
<tr>
<td>RG 50/100</td>
<td>17694.6 91.83</td>
<td>16947 16484.7 33.61</td>
<td>15875 16173.3 22.69</td>
<td>15484</td>
</tr>
<tr>
<td>RG 100/200</td>
<td>33339.6 613.6</td>
<td>32234 31477.8 220.6</td>
<td>30378 29226.8 140.5</td>
<td>28529</td>
</tr>
<tr>
<td>IT 31/30</td>
<td>6713.5 13.27</td>
<td>6577 6829.9 7.91</td>
<td>6279 6446.9 6.51</td>
<td>6327</td>
</tr>
<tr>
<td>OT 31/30</td>
<td>6388.1 12.09</td>
<td>6181 6453.2 8.15</td>
<td>6137 6328.4 6.95</td>
<td>6063</td>
</tr>
<tr>
<td>RJ 31/50</td>
<td>8567.1 20.11</td>
<td>8379 8638.4 10.56</td>
<td>8483 8653.2 10.48</td>
<td>8468</td>
</tr>
<tr>
<td>MV 36/60</td>
<td>9515.0 32.74</td>
<td>9355 9699.5 18.09</td>
<td>9123 9198.3 14.70</td>
<td>8922</td>
</tr>
<tr>
<td>FT 31/46</td>
<td>7888.3 22.25</td>
<td>7681 7926.8 13.83</td>
<td>7523 7666.5 12.54</td>
<td>7415</td>
</tr>
</tbody>
</table>

N.B. 20400 = 20 nodes, 40 Edges

TS with Penalty Reward

We implement TS with the penalty reward modification described earlier with different values of neighbourhood size (Nsize). In each experiment the random graphs of 50 nodes 100 edges is used. The results are averaged over 20 runs to improve reliability. In each run, 400 iterations are performed to select the best solution that offers the minimum processing time. In case of adding penalty reward strategy in TS, we define the value of $K_p=30$, $Q=80$, $Q'=10$.

As seen from figure 3, TS with the penalty reward strategy (PR) always produces designs with a shorter processing time than those from TS. Although the improvement is only about 2%, the ability of finding a new solution better than the best from conventional TS is shown. Results from the other sizes of task graphs follow similar trend.

Conclusions

This work examined the partitioning and scheduling problem using simulated inputs in some of in the forms of randomly generated and other task graphs that are often encountered in real applications. The results from our heuristic partitioner are used successfully in providing a schedule that minimizes the processing time. Although using processing time as a cost function may increase the search time, more accurate results can be obtained.

Three popular combinatorial optimisation algorithms, genetic, simulated annealing and tabu search, have been compared using a reference system architecture on various kinds of problems. In partitioning a design into hardware and software components, we find that tabu search provides higher quality results in a shorter time than both simulated annealing and genetic algorithm. Furthermore genetic algorithm demands more memory to store information about a large number of solutions, while tabu search and simulated annealing are also more memory efficient. We have also implemented the penalty reward scheme in an intensification strategy for tabu search, which can further improve the quality of solutions.