NON-PREEMPTIVE MULTI-CONSTRAIN SCHEDULING FOR MULTIPROCESSOR WITH HOPFIELD NEURAL NETWORK

Abdellatif H. ALI
ECE Department Modern Academy, Cairo, Egypt

ABSTRACT
In this paper, task scheduling for non-preemptive multi-constrained multi-processor systems was presented. The proposed model based on discrete Hopfield Neural network augmented with a methodology for weighting constrains to form overall network energy function. The network augmented with a layer to handle network re-initialization, based on min-max algorithm, case of local minima trapped without an acceptable solution. The proposed neural network solution does not require a predetermined scheduling length. Constrains included in the study are: task time, precedence, resources conflict, task dead time, and favoring tasks of the same setup to run on the same processor to suit reconfigurable hardware.

KEYWORDS: Preemptive, Multi-constrains, Multi-processors, Scheduling, Hopfield, Neural Network.

I. INTRODUCTION
Multiprocessor have become powerful computing mean for many applications such as information processing, database systems, weapon systems, weather forecasting, image processing, and real-time high speed simulators [17]. The multiprocessor hardware both fixed configuration and reconfigurable is the trend of the new developed hardware to cope with the high demands to processing power. The utilization of this powerful hardware highly dependent on proper scheduling of the problem tasks over the processing resources available. It is well known that scheduling is the most challenging problem of parallel computing [16].

Scheduling a problem tasks over multiprocessors, normally, is subject to constrain(s). Scheduling tasks of a multi-constrain problem has wide range of applications such as computer operating systems, digital communications, industrial control, weapons to targets assignment, Unmanned Aerial Vehicles, fighter aircrafts, and operation research. Such a problem is well known to be NP hard type. A tremendous effort required to find out an acceptable solution due to the growth of the problem complexity exponentially with problem size. Many approaches explored by researchers in fields such as AI, operation research, and neural networks to solve such problems. The aim was to find out a pseudo optimal solution in an efficient manner. The use of genetic algorithms recently have taken a great attentions of researchers [17][18][19].

The neural network approach is considered superior giving the potential of hardware realization, owing to their parallel architecture. Moreover, the mapping of algorithms based on neural network to run on parallel platforms is more straightforward than others.


The Hopfield neural network is well known to have the potential of being trapped in local minima’s [5]. Such a problem is well known to exist in search schemes for NP complete type of problems [11]. To solve an optimization problem by Hopfield network one has to follow the following steps:-

• Finding a network topology that represents the problem. That is, the final state of the network neurons could be interpreted easily as a solution to the problem.
• Finding energy function that if optimized, minimized, it will correspond to the best solution to the problem. The targeted function, combines both constrains and cost functions. In multi-constrain type of a problem arises the problem of finding out the proper weight for each constrain to combine in a single energy function.
• Computation of network synaptic neurons weights and neurons thresholds based on the energy function.

In this study, a methodology for task scheduling over multiprocessor to non-preemptive multi-constrained problem is proposed. The proposed methodology includes a simple analysis step to compute constrains weights, re-initialization algorithm based on Min_Max algorithm [15] case of local minima trap or looking for better solution, and a neuron firing sequence biased towards neurons with higher momentum to change. The constrains used in the study are task time, precedence, resources conflict, task dead time, and favoring tasks of the same setup to run on the same processor, beside the cost function of the schedule length.

This paper is organized as following: Section 2 introduces the Hopfield network. Section 3 contains the problem description and associated energy function.
Section 4 presents the network synaptic weights and neurons threshold. Section 5 explains neurons firing sequence. Section 6 presents the network re-initialization. Section 7 includes tests and results followed by the conclusion.

2. HOPFIELD NEURAL NETWORK

The Hopfield network is a fully connected single layered feedback type neural network. The Laipunov function, or the energy function, Eq. (1), which is subject to optimization by the network was introduced in [12].

$$E = -\frac{1}{2} \sum_i \sum_j V_i W_{ij} V_j + \sum_j \theta_j V_j$$

Where: $V_i$ is neuron state and $W_{ij}$ is the synaptic weights connecting neuron $V_i$ to $V_j$. and $\theta_j$ is the threshold of neurons $j$. The neuron states, in discrete Hopfield, are binary values 0, or 1, which complies with problem requirement. The network, normally, starts with neuron states distributed uniformly randomly between 0, and 1. The update rule of the neurons in iteration $n + 1$ is as follows:-

$$V_{i}^{n+1} = \begin{cases} 1 & \text{if } net_i > 0 \\ V_i^n & \text{if } net_i = 0 \\ 0 & \text{if } net_i < 0 \end{cases}$$

where

$$net_i = \sum_j W_{ij} V_j^n - \theta_i$$

The network found to evolve to a stable state giving the following conditions [13]:-

- There is no self-feedback, in other words, the diagonal elements of the weight matrix are all equal zero.
- The synaptic weight connecting neuron $i$ to neuron $j$ is the same as the one connecting neuron $j$ to neuron $i$, in other words, the connection matrix is symmetric.

Welling and Yi Xlong in [14] eased the later condition.

3. PROBLEM DESCRIPTION AND ENERGY FUNCTION

We consider the problem of N tasks to be scheduled over M processing units with some known upper limit of the scheduling time $T$. The scheduling time could fit, or over fit the case. Each task time is known a priori as well as the set of applied constrains. The Hopfield topology that well suits such a problem is a three dimensional architecture [7]. Each dimension represents a major factor in the problem: task, process, and time consequently. A neuron in such topology $V_{ijk}$ represents the state of assignment of a task $i$ to a processor $j$ in time slot $k$. The synaptic weight matrix is 6-dimensional $W_{ijkxyz}$ is the weight connecting neuron $V_{ijk}$ to neuron $V_{xyz}$.

The energy function includes beside the problem given constrains and cost function: preventing a task from executing simultaneously on different processors, preventing a processing unit from running multiple tasks at the same time, and migration of a task over different processing units.

The following is the proposed energy function:

$$E = \frac{1}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{sjk} + \frac{C_1}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_2}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_3}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_4}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_5}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_6}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_7}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_8}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_9}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk} + \frac{C_{10}}{2} \sum_{i,j} \sum_{k} V_{ijk} V_{jsk}$$

Where:

$S_i$ Is the time required for task $i$

$C_1, \ldots, C_{10}$ Are the weighting factors for constrains/cost function.

$g_{x} = k - d_{i} - d_{j}$ Is the dead time for task $i$.

$H(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$

$P_{1}(i,j,k) = \begin{cases} 1 & \text{if } (type(i)=type(x) \text{ and } j \neq y) \\ 0 & \text{otherwise} \end{cases}$

$P_{2}(i,x) = \begin{cases} 1 & \text{if } ((k + S_j) \leq z) \text{ or } ((z + S_i) \leq k) \\ 0 & \text{otherwise} \end{cases}$

$P_{3}(k) = \frac{k-1}{T}$

$P_{4}(i,x) = \begin{cases} 1 & \text{if } \exists \text{ resourceConflict between } i, x \\ 0 & \text{otherwise} \end{cases}$

The terms of the equation above can be explained as following consequently:

1. Processor unit $i$ cannot execute more than one job.
2. If job started on a processor unit it should continue, and complete on the same processor, prohibit task migration.
3. Tasks must have execution time equal to the required time S.
4. No assignments for a task after its dead time.
5. No more than one task shall be allowed to execute at some specific time on same processor unit.
6. Enforce the precedence constrains required.
7. Favor tasks of the same type to run on the same processor core.
8. Enforce non-preemptive task assignment.
9. Favor early schedule
10. Resolve use of exclusive resources between tasks.

4. NETWORK SYNAPTIC WEIGHTS AND NEURONS THRESHOLD

The above mentioned energy function could be mapped to setting synaptic weights \( W \) and neurons threshold \( \theta \) as following:

\[
W_{ij} = -c_i(1-\delta(x,i))\delta(y,j)\delta(z,k) - c_j(1-\delta(x,i))(1-\delta(y,j)) - c_k(1-\delta(x,i))(1-\delta(y,j))\delta(k,z)P_i(x) - c_j(1-\delta(i,x))(P_i(x,k) - c_k(1-\delta(i,x))(P_i(x,k,z)
\]

\[
\theta_{jk} = -c_j(2S_i - 1) - c_k + c_i G_{ik}H(G_{ik}) + c_k P_j(k)
\]

where,

\[
\delta(x, y) = \begin{cases} 
1 & x = y \\
0 & x \neq y 
\end{cases}
\]

In [7], [10] the same problem studied for preemptive tasks under constrains number: 1,2,3,4,5,10 and no bases explicitly mentioned for computing \( c_i \) values. In this study, we propose methodology for calculation based on equal share of the initial total synaptic energy assuming uniform random distribution initialization.

The above mentioned energy elements could be seen as hard elements that if violated the solution provided will be of no value, and optimization/cost function. The same set could be classified based on their effect on network weights to: synaptic, threshold, and both. So, let us denote the following sets: -

- \( \eta_H = \{1,2,3,5,6,8,10\} \) The set of hard elements that affects the synaptic weights.
- \( \eta_O = \{7\} \) The set of optimization elements that affects the synaptic weights.
- \( \xi_H = \{3,4\} \) The set of hard constrains that affects neurons threshold weights.
- \( \xi_O = \{9\} \) The set of optimization constrains that affects neurons threshold weights.

Our primary focus is to find weights for constrains in \( \eta_H \) such that their shares in the initial synaptic energy are likely equal. So, let us assume \( C_i^r \) is the count of links set by constrain \( i \), \( i \in \eta_H \). A constrain share in energy is proportional to the number of weights set by that constrain. Therefore, the weight factor will be

\[
C_i = \alpha \frac{1}{C_i^r}
\]

the energy, and \( C_i^r > 0 \). The probability of a set link to contribute to the initial energy is the equal to 0.25 which is the probability of its two terminal neurons being ON. Then, the mean over large number of trails \( t \) is

\[
\frac{1}{t} \sum_{i=1}^{t} Y_i \equiv 0.25\alpha.
\]

Consequently, the total initial synaptic energy for hard constrains will be

\[
\Xi_H C_i \left\{ \begin{array}{ll}
\Xi_C C_i & \text{if } i \in \eta_C, \\
0 & \text{otherwise}
\end{array} \right.
\]

5. NETWORK NEURONS FIRING SEQUENCE

After initialization, the network runs according to eq. 2. At any typical state, the network neurons could be divided into three sets: neurons have the tendency to change state from zero to one, we call them the hot set \( H \), neurons have the tendency to change from one to zero, we call them the cold set \( C \) and others keep their state.

The most widely used network firing sequence [1-2] is uniform random choices to prevent network from looping in a sequence of states. In this study, we adopt a biased random firing methodology. The choice is random biased towards the neurons of the highest momentum to the change both in hot and cold sets.

The choice of the next neuron to change is as following: let us assume \( H^S \) is the set of hot neurons sorted descending on the value of \( net_j \) and \( C^S \) is the set of cold neurons sorted ascending on the values \( net_j \) then, firing of neurons alternated between the cold and hot with firing index as:

For hot:

\[
b_H = \min(L(1+r^2[H^S]),[H^S]).
\]

For cold: \( b_C = \min(L(1+r^2[C^S]),[C^S]) \).

Where:

\[
\begin{align*}
& r \quad \text{Random variable uniformly distributed between 0, and 1.}
\end{align*}
\]
The network comes to a stable state when both sets are null. The stable state of the network provides a solution which is a local minima in the energy surface. Looking for more solutions to choose from a network re-initialization required.

6. NETWORK Re-initialization.

Network re-initialization required to look more satisfactory solutions. The most popular re-initialization is another random draw. The random re-initialization may produce a state close to a former initialization which maximizes the probability to end in same local minima. The initialization adopted in this study based on min-max [15] to assure start in farthest point in the hyperspace from all previous initializations, to maximize the probability of ending up with different solution. The initialization adopted in this study based on min-max [15] to assure start in farthest point in the hyperspace from all previous initializations, to maximize the probability of ending up with different solution. The procedure gives an indicative measure to the coverage of the space by the least distance between initializations. To summarize the re-initialization procedure let us assume we had n number of trials with unsatisfactory solutions and a set of historical neurons initialization vectors set: \( \{h_1, h_2, ..., h_n\} \) then we follow the following steps:

\[
\begin{align*}
    & h_{n+1}(i) = \overline{h}_n(i), \text{ (complement of)} \\
    & \forall \ k = 2 \rightarrow NMT \quad \text{Find the} \quad d \text{ distance set ,and the} \quad \text{minimal distance set as following:} \\
    & D = \left\{ d_i : \sum_{j=1}^{i-1} |h_i(j) - h_{n+1}(j)| \forall i = 1 \rightarrow n \right\}, \\
    & H_m = \left\{ h_i : d_i \leq d_j \forall d_j \in D \right\}, \text{ then set current} \\
    & \text{bit according to the following rule:} \\
    & \begin{cases} 
        0 & S > 0 \\
        S = 0 & \text{where} \\
        1 & S < 0 \\
    \end{cases} \\
    & S = \sum_{i \in H_m} h_i(k) - \sum_{i \in \overline{H}_m} |h_i(k) - 1|
\end{align*}
\]

Figure 1 shows the over all process of the proposed network working in a multi-constrains scheduling problem.

7. TESTS AND RESULTS

In this section, the network model tests on variety of task scheduling problems. One of these problems presented next followed by summary over others.

Figure 2 shows a precedence diagram for sixteen-task problem taken from [11]. The tasks are marked with “task number/ required time units/task type group”. The tasks are to be scheduled on two processing elements. The network formed and set to initial state according to the former model. The result of network run is presented in fig. 3. Fig. 4 shows the number of violations, for hard constrains, per network iteration. From the figure we can see that the network came to a valid solution, after 24 iterations.

Fig. 2. The precedence diagram for the 16 task problem
In the following table we summarize the results of running the network on 10 problems. The problems parameters (the number of tasks, number of processing elements, and the task processing time) taken from random normal distribution draws with mean and standard deviation as follow: (10, 2), (4, 1), and (5, 2.5). The precedence constrains generated using random generator as sequence of pairs (task1 proceeds task2) with rejection to invalid choices. The given scheduling time span computed by distributing the tasks equally on the processors and assuming that the lengthiest jobs will run on the same processing element then adding more 25%. Table 1 presents the results with only precedence constrains compared to the heavy brute search.

Table 1 Results of ten problems with only precedence constrains

<table>
<thead>
<tr>
<th>Index</th>
<th>No. of Tasks</th>
<th>No. of Processors</th>
<th>Given Scheduling Time</th>
<th>Schedule length from first valid Solution</th>
<th>Iterations to first valid Solution</th>
<th>Trials count to global minima proposed: random</th>
<th>Minimum schedule length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>4</td>
<td>24</td>
<td>14</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
<td>15</td>
<td>13</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>4</td>
<td>30</td>
<td>20</td>
<td>1</td>
<td>1:3</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>4</td>
<td>17</td>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>4</td>
<td>22</td>
<td>15</td>
<td>2</td>
<td>2:2</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>4</td>
<td>26</td>
<td>16</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>5</td>
<td>25</td>
<td>18</td>
<td>3</td>
<td>3:5</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>4</td>
<td>31</td>
<td>22</td>
<td>1</td>
<td>2:5</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>4</td>
<td>24</td>
<td>14</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>5</td>
<td>23</td>
<td>17</td>
<td>1</td>
<td>1:1</td>
<td>14</td>
</tr>
</tbody>
</table>

From the table we easily infer that the networks converged to a solution within few numbers of iterations. In 50 percent of the time the network came to a deep local minimum. In 50 percent of the time the network came to the global minima. The cases in which the network did not converge to the global minima or deep local minima the solution is found to be within few re-initialization trials of the network.

Table 2 shows the results of running another 10 problems with constrains. Constrains generated, also, based on random uniform distribution, rejecting invalid values. The number of constrains is a random number drawn from normal distribution of mean equal to 1.5 times the number of tasks and standard deviation of 50% of that mean. The proposed firing sequence is faster overall 25 percent compared to random firing.

The network took more iteration to reach the first valid solution compared to precedence only cases. The probability of reaching global minima in the constrained problems appears to be higher than the unconstrained problems which points to the fact that constrains eliminate many of the valid unconstrained solutions.

Table 2. Results of constrained Problems

<table>
<thead>
<tr>
<th>Index</th>
<th>No. of Tasks</th>
<th>No. of Processors</th>
<th>No of constrains</th>
<th>Given Scheduling Time</th>
<th>Schedule length of first valid Solution</th>
<th>Iterations of first valid Solution</th>
<th>Trials to global minima Proposed: Rand</th>
<th>Minimum schedule length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
<td>13</td>
<td>27</td>
<td>23</td>
<td>16</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>4</td>
<td>15</td>
<td>25</td>
<td>19</td>
<td>32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>23</td>
<td>18</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>2</td>
<td>8</td>
<td>36</td>
<td>26</td>
<td>70</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>2</td>
<td>8</td>
<td>30</td>
<td>18</td>
<td>70</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>2</td>
<td>10</td>
<td>23</td>
<td>15</td>
<td>3</td>
<td>1:3</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>3</td>
<td>8</td>
<td>27</td>
<td>22</td>
<td>18</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>3</td>
<td>14</td>
<td>26</td>
<td>21</td>
<td>91</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>4</td>
<td>10</td>
<td>14</td>
<td>9</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>2</td>
<td>23</td>
<td>31</td>
<td>22</td>
<td>31</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

8. CONCLUSION

In this paper a task scheduling model for non-preemptive multi-constrained multi-processor problems based on Hopfield Architecture presented. Methodologies for computing constrains weighting factor, and network re-initialization proposed. The proposed neural network solution dose not requires a predetermined scheduling length. The constrains used in the study are task time, precedence, resources conflict, task dead time, and favoring tasks of the same setup to run on the same processor beside the cost of schedule length.

The results of running the network on different problems show convergence to a valid solution within a fewer number of iterations. A selection of the shortest scheduling length from the first few solutions will be very close to the global minima or the global minima itself.

The major problem of this approach is network size and the simulation computations time. The hardware realization with simple and dense processing elements resolves this problem.

REFERENCES