Neuron Constraints to Model Complex Real-World Problems

Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini

DEIS, University of Bologna
{a.bartolini,michele.lombardi2,michela.milano,luca.benini}@unibo.it

Abstract. The benefits of combinatorial optimization techniques for the solution of real-world industrial problems are an acknowledged evidence; yet, the application of those approaches to many practical domains still encounters active resistance by practitioners, in large part due to the difficulty to come up with accurate declarative representations. We propose a simple and effective technique to bring hard-to-describe systems within the reach of Constraint Optimization methods; the goal is achieved by embedding into a combinatorial model a soft-computing paradigm, namely Neural Networks, properly trained before their insertion. The approach is flexible and easy to implement on top of available Constraint Solvers. To provide evidence for the viability of the proposed method, we tackle a thermal aware task allocation problem for a multi-core computing platform.

Keywords: Constraint Programming, Neural Network, Thermal aware allocation and scheduling

1 Introduction

The benefits of combinatorial optimization for the solution of real-world industrial problems are a widely acknowledged evidence, sitting of an ever-growing collection of success stories [11, 12, 20]. Yet, the application of optimization approaches to many practical domains still encounters active resistance by practitioners. A considerable part of the issue stems from difficulties in devising an accurate representation for the target domain. As matter of fact, many optimization approaches assume the availability of a declarative description of the system, usually obtained by introducing some degree of approximation; the resulting accuracy is critical for the optimization effectiveness: an over-simplified model may threat the successful application of the most advanced combinatorial method. Coming up with an accurate model may be very challenging whenever there are elements admitting no obvious numerical description, or the system behavior results from the interaction of a very large number of actors.

In this work, we propose a simple and effective technique to bring hard-to-describe systems within the reach of optimization methods; the goal is achieved by embedding a properly trained Neural Network into a combinatorial model. The Neural Network basically learns how to link decision variables either with

a corresponding metric or with observable variables or with other decision variables. Such a hybridization with a soft-computing paradigm allows the model to accurately represent complex interactions and to handle difficult-to-measure metrics.

As a host technology, Constraint Programming (CP) represents an ideal candidate, thanks to the ability to deal with non-linear functions and the modularity of constraint based models. Specifically, we introduce a novel class of global Neuron Constraints to capture the behavior of a single Neural Network node. The ability to incorporate soft-computing system representations marks a distinguishing advantage of CP over competitor techniques (namely those based on linear models), increasing its appeal for the solution of industrial problems.

To showcase the proposed approach, we consider a temperature aware workload allocation problem over a Multi-Processor Systems on Chip (MPSoC) with Dynamic Voltage and Frequency Scaling (DVFS) capabilities. DVFS allows the programmer to slow the pace of one or more processors, to let the system cool down and become ready to accept more demanding tasks later on. The thermal behavior of a MPSoC device is the result of the interaction of many concurrent factors (including heat conduction, processor workload, chip layout). Despite the dynamic of the single phenomena is known, the complexity of the overall system makes it very hard to devise a declarative thermal model. In such a context, a Neural Network can be designed and trained to approximate the system thermal behavior. The resulting network can then be embedded in a combinatorial model and used to produce an optimized workload allocation, avoiding resource overheating as well as over-usage. We tested the approach obtaining consistently better result compared to a load balancing strategy guided by a temperature aware heuristic; moreover, we even improve the results of a very well-performing surrogate temperature measure.

2 Neural Networks: Background and Definitions

An artificial Neural Network (NN) is a computational system emulating the operation of a *biological* neural network; NNs are capable to perform non-linear computations and can be deployed to perform different tasks by proper *training*. NNs are parallel systems, consisting of o set of many interconnected computing elements; the basic computation block is called *artificial neuron* and mimics the behavior of a neural cell, processing multiple electrical input from neighboring cells to produce a single electrical output. The first simplified neuron models date back to the 40' [13]: basically, an artificial neuron is a non-linear function with vector input $\overline{x}$ and scalar output y; in detail:

$$y = \phi\left(b + \sum_i w_i x_i\right) \tag{1}$$

where x_i denotes a single component in $\overline{x}$, the argument of ϕ is known as neuron *activity*, b is a *bias* and ϕ is called *activation function*; ϕ is a monotonic non-decreasing function, so that inhibitory/excitatory connections between biological neurons can be respectively modeled as negative/positive weights w_i. Artificial

neurons differ by the type of activation function and can be broadly classified into *threshold*, *linear/piecewise-linear* and *sigmoid* neurons; for example:

$$\phi(a) = \begin{cases} 1 \text{ if } a \geq 0 \\ 0 \text{ if } a < 0 \end{cases} \quad (2) \qquad \phi(a) = a \quad (3) \qquad \phi(a) = \frac{2}{1 + e^{-2a}} - 1 \quad (4)$$

the function in Equation (2) corresponds to a threshold neuron (the classical *perceptron* from [19]), Equation (3) corresponds to a linear neuron and Equation (4) is a sigmoid neuron (hyperbolic tangent). In many cases ϕ acts as a squashing function, restricting the output to be in the interval $[0, 1]$ or $[-1, 1]$.

A Neural Network is a system with vector input/output (say $\overline{x}$, $\overline{y}$) and composed of one or more artificial neurons; each neuron receives input from neighbors (or from the outside the network, i.e. $\overline{x}$) and computes an output signal which is propagated to other neurons; designated neurons provide the network output $\overline{y}$. A NN can be represented as a directed graph and is said *feed-forward* in case the graph is cycle-free, *recurrent* if at least a loop is present. Feed-forward networks are usually organized into *layers*; in this case neurons/nodes in level 0 accept the input $\overline{x}$, neurons in the last layer provide the output $\overline{y}$, while each neuron in the remaining layers (*hidden*) is connected to all nodes in the previous and in the next layer; there is no connection between nodes in the same layer.

Weights of a NN are usually decided in a learning stage to match input/output pairs in a *training set*; this can be done (e.g.) by means of the back-propagation process [6, 17]. Depending on the neuron types and the training set, the network acts as a classifier or performs regression analysis; the network ability to treat previously unseen input patters (i.e. generalization) depends to a large extent on the chosen training set. Single layer networks can only match linearly separable training sets [14]; conversely, multi-layer networks have no such limitation and can model any $\mathbb{R}^n \to \mathbb{R}^m$ function with finitely many discontinuities [8], provided the hidden layers have a non-linear activation function and the network is sufficiently large.

3 Neuron Constraints

The main appeal of Neural Networks stems from their ability to learn the approximate behavior of opaque or very complex systems, without requiring detailed knowledge of their components and interactions. User intervention is required in the preparation of the training set, but not in the actual definition of weights. Once the training stage is over, the network *is intrinsically declarative* and can therefore be embedded into a classical combinatorial model. In detail, we proceed by introducing a novel and simple class of (global) *Neuron Constraints*, modeling a single artificial neuron with a specific activation function. Real valued variables are associated to the output and to each component of the input vector; hence a Neuron Constraint has the following signature:

$$\text{actfunction}(\mathtt{Y}, \overline{\mathtt{X}}, \overline{w}, b)$$

where 'actfunction' denotes the activation function type — i.e. function ϕ in Equation (1) —, $\mathtt{Y}$ is the output variable, $\overline{\mathtt{X}}$ is the vector of input variables,

$\overline{w}$ is the vector of weights and b is the bias. The integration of a trained NN into a CP model is as straightforward as introducing a Neuron Constraint for each node, connecting input/outputs variables and setting arc weights. Using a global constraint for each single neuron rather than for a whole network provides a fine grained modeling approach, allowing complex networks (even recurrent ones) to be defined with a limited number of basic components, i.e. a constraint for each type of activation function. In particular, we have implemented the activation functions from Equations (2),(3) and (4), corresponding to the Neuron Constraints 'hardlim', 'purelin' and 'tansig'[1].

3.1 Filtering for Neuron Constraints

Filtering in a Neuron Constraint can be done by separately tackling the activity expression and the activation function; namely, Equation (1) can be decomposed so that we have:

$$\mathtt{A} = b + \sum_i \mathtt{X_i} w_i \qquad (5) \qquad\qquad \mathtt{Y} = \phi(\mathtt{A}) \qquad (6)$$

where $\mathtt{A}$ is an artificially introduced *activity variable*. Equation (5) is linear and poses no issue; function ϕ is monotonic non-decreasing, so that bound consistency can be enforced by means of the following rules:

$$\max(\mathtt{A}) \text{ updated } \Rightarrow \quad \max(\mathtt{Y}) \leftarrow \max\{y' \mid \phi(\max(\mathtt{A})) = y'\} \qquad (7)$$

$$\max(\mathtt{Y}) \text{ updated } \Rightarrow \quad \max(\mathtt{A}) \leftarrow \max\{a' \mid \max(\mathtt{Y}) = \phi(a')\} \qquad (8)$$

Rules for "min" are analogous. Observe that, from a mathematical standpoint, the set $\{y' \mid \phi(\max(\mathtt{A})) = y'\}$ is a singleton and only contains the value $\phi(\max(\mathtt{A}))$, similarly the set $\{a' \mid \max(\mathtt{Y}) = \phi(a')\}$ is in fact $\{\phi^{-1}(\max(\mathtt{Y}))\}$ and so on. The distinction becomes however relevant when finite computing precision is taken into account. As an example, the filtering rules for the upper bound with *tansig* function are:

$$\max(\mathtt{A}) \text{ upd. } \Rightarrow \quad \max(\mathtt{Y}) \leftarrow tansig(\max(\mathtt{A}))$$

$$\max(\mathtt{Y}) \text{ upd. } \Rightarrow \quad \max(\mathtt{A}) \leftarrow \begin{cases} tansig^{-1}(\max(\mathtt{Y})) & \text{if } \max(\mathtt{Y}) \in]-1,1[\quad (A) \\ \max\{a' \mid tansig(a) = 1\} & \text{if } \max(\mathtt{Y}) = 1 \quad (B) \\ \max\{a' \mid tansig(a) = -1\} & \text{if } \max(\mathtt{Y}) = -1 \quad (C) \end{cases}$$

where $tansig(a)$ is as from Equation (4) and $tansig^{-1}(y) = 0.5 \times \ln((1-y)/(1+y))$. The expressions from case (B) and (C) are implementation dependent *constants*. The rules for lower bound filtering are analogous. As an important consequence of precision issues, an $\mathtt{A}$ variable may be unbound even if the corresponding $\mathtt{Y}$ variable is bound; forcing $\mathtt{A}$ to be bound would result in an incorrect behavior; hence the uncertainty due to precision errors should be eventually carried on in the problem solution. As one can see, aside from precision issues the filtering rules are simple, making the implementation of the approach fairly easy on off-the-shelf available solvers.

[1] The naming convention comes from the MATLAB Neural Network Toolbox.

4 A Use Case: Thermal Aware Workload Allocation

Providing evidence of the method effectiveness requires a problem with non-trivial modeling issues; specifically, in this paper we tackle a thermal-aware workload allocation problem on Multi Core Systems on Chip (MPSoC); due to the inherent complexity, the description of the problem and the solution approach takes an extensive portion of the paper.

4.1 Context and Motivation

Temperature management in MPSoCs is receiving growing research interest in recent years, pushed by the awareness that the development of modern multi-core platforms is about to hit a thermal wall. A larger number of cores packed on a single silicon die lead to an impressing heat generation rate; this is the source of a number of issues [5] such as (1) the cost of the cooling system; (2) reduced reliability and lifetime; (3) reduced performance.

Classical approaches include changing the operating frequency, task migration or core shutdown, triggered when a specified threshold temperature is reached. This reactive method avoids chip overheating, but may have a relevant impact on the performance. Hence several works have investigated thermal-aware workload allocation, making use of mechanisms as DVFS to prevent the activation of more drastic cooling measures. Those approaches include: (1) on-line optimization policies [4, 5, 2, 22], based on predictive models and taking advantage of run-time temperatures read from hardware sensors; (2) off-line allocation and scheduling approaches [18, 15], usually embedding a simplified thermal model of the target platform [16]; (3) off-line iterative methods [1, 21], performing chip temperature assessment via a simulator (e.g. the HotSpot system [10]).

Capturing the thermal behavior of an MPSoC platform is a tricky task; the temperature depends on the workload, the position of the heat sinks, thermal interactions between neighboring cores. This is why off-line approaches rely on simplified models and often disregard either non-homogeneities due to the floorplan or heat transfer between neighboring cores. Iterative approaches overcome the issue by performing thermal simulation after each iteration, but this prevents information on the temperature behavior to be directly used in the optimization procedure. Despite the dynamic of the single elements concurring to system temperature is known, the complexity of the overall system makes it very hard to devise a declarative model: in such a context, however, a Neural Network can still be designed and trained to approximate the system thermal behavior.

4.2 The Target Problem

Specifically, we address a workload allocation problem on a multi-core system consisting of a set P of Processing Elements p_j (PE); the operating frequency of each element can be dynamically changed between a minimum and maximum

value (say f_{min}, f_{max}) with a fixed step[2]; the workload is specified as a set T of independent [3] tasks t_i. As a target system we designed a framework (implemented in MATLAB) for accurate emulation of the temperature evolution of a multicore platform when executing a sequence of tasks. The optimization problem consists in the assignment of a PE and an operating frequency to each task, so that the full workload is executed within a specified deadline and the final peak temperature is minimized. Higher operating frequencies result in lower durations, but also higher power consumption and heat generation; PEs are non-homogeneous from a thermal point of view; custom starting temperatures (say $Tstart_j$) can be specified for each PE to take into account the case of an already running system.

4.3 Simulation Framework

The simulation framework has been developed to simulate system evolution, with specific regard for the thermal transient;, but we also take into account the dependency of execution time and power consumption on the frequency and the task properties [3].

Task Duration: We assume task execution time to be frequency dependent, with cpu-bounded tasks being more sensitive to frequency changes than memory-bounded tasks; the Clock per Instruction (CPI) metric is a simple and widely adopted [2] way to estimate the degree of memory boundedness of a task. For an in-order CPU[4], the execution time D_i of t_i can be expressed as follow:

$$D_i = \frac{1}{f_{max}} \cdot NI_i \cdot \left(\frac{f_{max}}{f_i} + CPI_i - 1 \right)$$

where NI_i is the total number of instruction composing the task; f_i is the PE frequency during the execution of t_i and CPI_i is the average task CPI when running at maximum frequency. According to this model, each task is therefore characterized by an NI_i, CPI_i pair. The use of cycle accurate simulation would provide a more detailed duration model, but the corresponding computational burden is prohibitive with the time resolution needed to identify thermal transients. We assume a constant operating frequency for each task, even if in principle it is possible to switch the frequency during execution, since the overhead induced by recording this information at run-time would be too high.

Power Consumption: We use a model to estimate the power consumption of a Processing Element, accounting for the dependency on the frequency and the CPI of the task currently in execution; this is in line with several approaches,

[2] This is in line with the real HW DVFS capabilities of today and future MPSoC [9] that allow frequency to change by steps of hundreds of MHz

[3] Independent tasks are common in many scenarios, such as real time OS, web servers, high performance computing...

[4] Recent trends in many-core often witness the use of a simple, in-order cores as basic blocks for the parallel architecture [9]

showing how to extract a power model directly from an MPSoC by combining power and performance measurements with data regression techniques [7]. In detail, our power model has been empirically extracted from measurements performed on an Intel® server system S7000FC4UR based on the quad-core Xeon® X7350 processor, with a maximal frequency of 2.93GHz (see [3]). The resulting model for a task t_i is reported in the following equation, together with the value of each constant; the static power consumption $Wstat$ is 3 Watt:

$$Wdyn = k_A \cdot {f_{PE}}^{k_B} + k_C + (k_D + k_E \cdot f_{PE}) \cdot {CPI_i}^{k_F} + Wstat \tag{9}$$

with:

$$k_A = 3.87 \cdot 10^{-8} \qquad k_B = 2.41 \qquad k_C = 1.10$$
$$k_D = -4.14 \qquad k_E = 5.1 \cdot 10^{-3} \qquad k_f = -3.02 \cdot 10^{-1}$$

Thermal Behavior: We use a state-of-the-art thermal simulator to emulate the system temperature evolution in time and space under different power stimuli [16, 10]. State-of-the-art simulators start from a representation of the platform and allocate the input PE power, dissipated in each thermal simulation time interval over the floorplan. Then the entire surface of the die is spatially discretized in a two dimensional grid. Each spatial block models a heat source and is characterized by an intrinsic temperature. This models the bottom surface and the injection of heat in the target multicore package. In addition, the package volume is partitioned in cells. Each cell is modeled with the equivalent thermal capacitance and resistance and connected with the adjacent ones. At each simulation step the R, C thermal-equivalent differential problem is solved providing the new temperature value for each cell as output. We embed in our set-up the HotSpot simulator [10], since it is a de-facto standard in MPSoC thermal simulation. Each time a new task is scheduled the power consumption of each core is estimated by using the power model and fed to the simulator.

5 Workload Allocation as an Optimization Problem

5.1 Modeling the Thermal Behavior via a Neural Network

The use of a thermal simulator to model temperature dynamics allows our framework to accurately emulate the behavior of a real-world MPSoC system; as a main drawback, the resulting thermal model is not declarative and cannot be directly handled via CP. Hence, *devising a declarative thermal model is a necessary step if Constraint Optimization is to be applied* and Neuron Constraints provide us an effective tool to deal with the issue.

In detailed we are interested in predicting the temperature after the system has been running some workload for a specific time span Δ; this depends non-linearly on the the initial temperature $Tstart_j$ and the power consumption P_j of every PE in the platform, plus the environment temperature $Tenv$:

$$T_j = f(\overline{Tstart}, \overline{P}, Tenv, \Delta)$$

where $\overline{Tstart}$ is the vector of initial temperatures and $\overline{P}$ is the vector with the average power consumption of each core. The network used to learn such a non-linear relationship must ideally be as simple as possible to reduce the computational burden of the optimization problem. We evaluated different topologies and input configurations: the best trade-off between NN complexity and accuracy is obtained by using a feed-forward two-layer neural network for each PE, with 'tansig' neurons in the hidden layer and a single 'purelin' one in the output layer. In detail, the network for PE p_j models the function:

$$\|T_j\| = g(\|\overline{Tstart}\|, \|\overline{P}\|, \|\overline{P} \cdot \Delta\|, \|\Delta\|)$$

all network inputs are normalized (see the $\|\cdot\|$ notation) and $\overline{P} \cdot \Delta$ represents the average consumed energy (i.e. the product between the consumption vector and the interval duration). Overall, each network has 13 inputs for a 4 core platform; the hidden layer size is 3/4 of the input number, i.e. 10 neurons in this case. The network output is the (normalized) predicted temperature for PE p_j.

The training and test set consist each of N randomly generated tuples, containing values for the inputs $\overline{Tstart}, \overline{P}, \Delta, T_{env}$. We then use HotSpot to simulate the final temperatures T_j corresponding to each tuple. Network training is performed via back-propagation, adjusting weights and bias according to Levenberg-Marquardt. Figure 1 shows the prediction error for a training and test set of $N = 5000$ random elements. One can observe from the plot that the selected Neural Network provides an estimation error below 0.1^oC for more than the 90% of the validation patterns; moreover, the error prediction is always within $\pm 1^oC$.

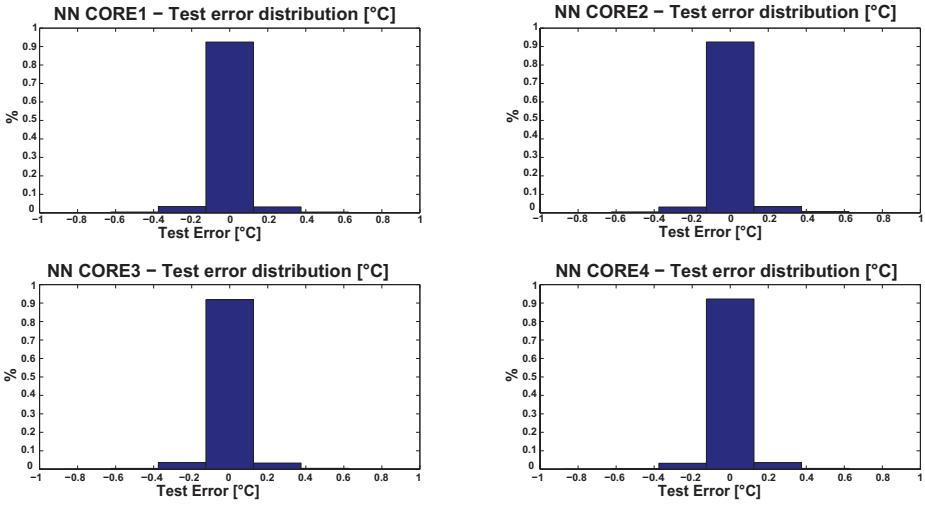

Fig. 1. Neural Network Test Error histogram

5.2 Combinatorial Model and Solution process

Now, the workload allocation over a single time interval can modeled as a Constraint Optimization Problem, featuring two decision variable arrays $\mathtt{P}$ and $\mathtt{F}$ (respectively representing the chosen frequency and processing element for each task). In detail, let D_{max} be the global deadline value, W_{max} the maximum power consumption, let $Tenv$ be the environmental temperature and T_{max} the maximum allowed temperature; let $Tstart_j$ denote the initial temperature of p_j and T_j be the final one; then the problem can be formulated as:

$$\min \quad \max_{p_j \in P} \mathtt{T_j}$$

$$\text{s.t.:} \quad \mathtt{D_i} = \frac{1}{f_{max}} NI_i \left(\frac{f_{max}}{\mathtt{F_i}} + CPI_i - 1 \right) \qquad \forall t_i \in T \qquad (10)$$

$$\sum_{t_i \in T} \mathtt{D_i} \cdot (\mathtt{P_i} = j) \leq D_{max} \qquad \forall p_j \in P \qquad (11)$$

$$\mathtt{WT_i} = (k_A \cdot {\mathtt{F_i}}^{k_B} + k_C) + (k_D + k_E \cdot \mathtt{F_i}) \cdot {cpi_i}^{k_F} + Wstat \qquad \forall t_i \in T \qquad (12)$$

$$\mathtt{W_j} = \frac{1}{D_{max}} \sum_{t_i \in T} \mathtt{D_i} \cdot \mathtt{WT_i} \cdot (\mathtt{P_i} = j) \qquad \forall p_j \in P \qquad (13)$$

for the Neural Network:

$$\mathtt{NW_j} = \mathtt{W_j}/W_{max}, \mathtt{ND_j} = D_{max}, \mathtt{NWD_j} = \mathtt{NW_j} \qquad \forall p_j \in P \qquad (14)$$

$$\mathtt{NTI_j} = (Tstart_j - Tenv)/(T_{max} - Tenv) \qquad \forall p_j \in P \qquad (15)$$

$$\mathtt{NTO_j} = (\mathtt{T_j} - Tenv)/(T_{max} - Tenv) \qquad \forall p_j \in P \qquad (16)$$

$$\text{Neuron Csts between } \mathtt{NW_j}, \mathtt{ND_j}, \mathtt{NWD_j}, \mathtt{NTI_j} \text{ and } \mathtt{NTO_j} \qquad (17)$$

with:

$$\mathtt{P_i} \in \{0, |P| - 1\} \qquad \forall t_i \in T$$

$$\mathtt{F_i} \in \{f_{min}..f_{max}, \text{ multiple of 100 MHz}\} \qquad \forall t_i \in T$$

$$\mathtt{D_i} \in [0, D_{max}], \mathtt{T_i} \in [T_{env}, T_{max}], \mathtt{WT_i} \in [0, W_{max}] \qquad \forall t_i \in T$$

$$\mathtt{W_j} \in [0, D_{max}], \mathtt{NW_j}, \mathtt{ND_j}, \mathtt{NWD_j}, \mathtt{NTI_j}, \mathtt{NTO_j} \in [0, 1] \qquad \forall p_j \in P$$

Basically, real variables $\mathtt{D_i}$ model task durations; $\mathtt{WT_i}$ and $\mathtt{W_j}$ respectively represent the power consumption for each task and the average power consumption for each processor as from Section 4.3; $\mathtt{NW_j}$, $\mathtt{ND_j}$, $\mathtt{NWD_j}$, $\mathtt{NTI_j}$ are the normalized inputs to the neural network and correspond to power consumption, duration, energy and input temperature; $\mathtt{NTO_j}$ are the normalized network outputs and $\mathtt{T_j}$ are the final temperature variables. Constraints (10) and (12) respectively correspond to the duration and power model in the simulation framework; Constraints (11) and Constraints (13) are the deadline restrictions and average power computation. Constraints (14) to (16) are normalization formulas. Finally, the model contains Neuron Constraints matching the structure of the network from Section 5.1.

Observe all variable except for the decision ones (i.e. $\mathtt{P_i}$ and $\mathtt{F_i}$) are real valued. Our current implementation is based on Comet 2.1.1, which lacks real variables support in the CP module; therefore, we use integer variables with a fixed precision factor and all constraints are formulated so as to avoid rounding errors. As a consequence, there may be (bounded) imprecision on the final temperature values forecast by the networks: in this case we assume a conservative approach and pick the worst possible value given the rounding error bound.

5.3 Solution Process

The constraint model has been implemented in Comet 2.1.1 using the CP (rather than the local search) module and solved by alternating restarts and Large Neighborhood Search (LNS). In both cases, the base approach is tree search, with a relatively simple two-stage strategy; in detail:

- *Stage 1, PE allocation:* search is performed on the $\mathtt{P_i}$ variables, by opening binary choice points:
  - the branching variable is selected uniformly at random among those of the 15% tasks with the smallest number of instructions NI_i;
  - the value to be assigned on the left-branch is the index of the PE p_j with smallest lower bound for the expression: $\sum_{t_i \in T} \mathtt{D_i} \cdot (\mathtt{P_i} = j)$ (see Constraints (11) in the model); on the right branch $\mathtt{P_i} \neq j$ is posted.
- *Stage 2, frequency assignment:* once all $\mathtt{P_i}$ variables are bound, search is performed on the $\mathtt{F_i}$ variables by domain splitting:
  - the branching variable is chosen with the same criterion as in Stage 1;
  - let F_i be the selected variable and f^* be the middle value in its current domain; search is performed by opening a binary choice point and respectively posting $\mathtt{F_i} \leq f^*$ and $\mathtt{F_i} > f^*$ on the left/right branch.

The main underlying idea is to assign a PE and a frequency to tasks with low NI_i value early in search. The solution initially performs tree search with restarts; each attempt is capped at 800 fails and the limit grows by 7.5% if no solution is found. Whenever a solution is reached the LNS loop begins; at each LNS iteration the incumbent solution is partially relaxed; in detail, tasks are ranked by decreasing value of the expression $NI_i \cdot r$ (with r a random number in $[0, 1]$), the first 60% tasks in the ranking are selected and the corresponding $\mathtt{P_i}, \mathtt{F_i}$ variables de-assigned. The the problem is re-optimized with the described tree search method. Each LNS iteration is capped at 800 fails and the value is increased by 7.5% in case the limit is reached (same as for restarts). Every 3 iterations with no solution improvement, the process switches back to restarts and so on.

6 Experimental Results

In principle, the embedded neural network should provide the solver with a powerful model of the system behavior, taking into account the diversity of

the thermal dynamics of each core and the effect of non-homogeneous starting temperatures; on the other side, the network complexity may lead to poor propagation and slow down the solution process. To assess the effectiveness of the proposed approach, we performed an experimental evaluation: our method was compared to two different variants, making use of simpler (arguably less accurate) thermal cost functions.

Considered Problem Variants: Due to the tight connection between temperature and power consumption, in the first considered variant we replaced the temperature minimization from Section 5.2 with a *power balancing* objective; namely, we minimize:

$$\max_{p_j \in P} \mathtt{W_j} \tag{18}$$

in the following, we refer to the original approach as `NN` and as `PP` to this first variant. The resulting combinatorial model is much simpler, as it contains no neural network; moreover, this surrogate thermal objective performs usually very well, due to the strong dependency of temperature on power consumption. However, this approach does not account for non-homogeneous thermal behaviors (e.g. due to the core location) and still requires an accurate power model with a well defined structure, which may not be available in many practical situations. In this case, a Neural Network can still be trained to approximate the thermal behavior, while Equation (18) can no longer be used. Therefore, we considered a second problem variant with a *load balancing* objective; namely, we maximize the smallest cumulative duration among the processors:

$$\min_{p_j \in P} \sum_{t_i \in T} \mathtt{D_i} \cdot (\mathtt{P_i} = j) \tag{19}$$

we refer to this second variant as `HH`. In this case, the search strategy is modified to incorporate some knowledge of the thermal behavior; in particular, the left and right branches in the frequency assignment stage are *inverted* depending on the task CPI. In detail, the solver prefers high frequency values if $CPI_i \leq 10$, while low frequencies are given priority if $CPI_i > 10$. The reason is that the duration of a low CPI task has a strong dependence on the operating frequency, allowing the heat contribution from static power consumption to be minimized by reducing the execution time; conversely, high CPI tasks have less elastic behavior and are best tackled by reducing the dynamic power consumption with a low frequency assignment. This modification proved very effective for the `HH` method behavior.

Input Workload and Target System: we synthesized 40 random workload instances, counting around 50 tasks each. Task durations (in seconds) and CPI were generated according to a mixed Gaussian distribution, representative of a mostly computation intensive workload, with a minor portion of memory-bounded tasks. The NI_i values were synthesized so as to keep the system 80% busy at maximal frequency. We considered two quad core platform, with a 1x4

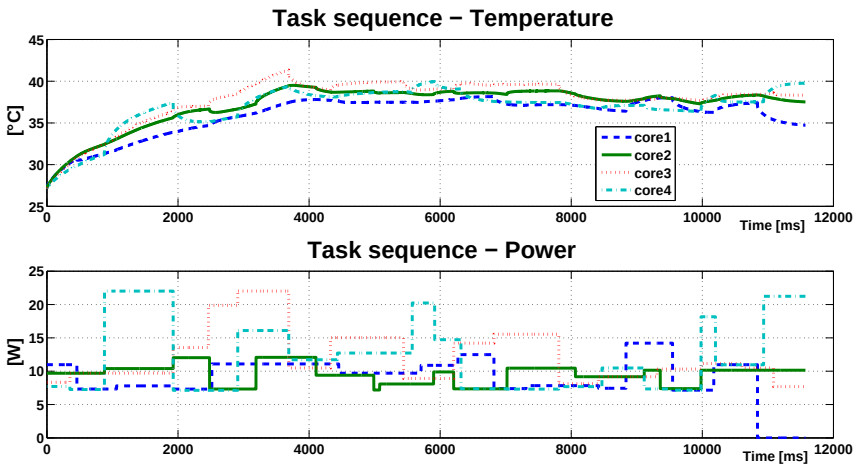

Fig. 2. Temperature and power dynamics on a single experiment

(linear) and 2x2 (square) floorplan; frequencies range between 1600 and 2900 MHz and the global deadline D_{max} is 10 seconds. The choices are representative of a server system, regularly accepting a typical workload to be dispatched before the next arrival.

We computed optimized workload allocation and frequency assignments for both the target platforms, by running each approach for 90 seconds on an Intel Core 2 T7200, 2GHz; the resulting solutions were *executed on the simulation framework*, with all cores starting from a room temperature of 26.5°C. Since all considered variants make use of approximated thermal model, there is no theoretical guarantee for the dominance of one approach over another: the use of simulation to evaluate the results ensures a fair comparison and a reliable effectiveness assessment. Moreover, since the optimized solution are evaluated via simulation, the results are unaffected by any numerical issue in the models.

The typical thermal behavior exhibited by the `NN` approach solutions is depicted in Figure 2, showing both temperature and power dynamics for a single experiment; each line in the graphs corresponds to a PE: as one can see, after an initial transient behavior the temperature becomes pretty stable, thanks to the thermal aware allocation.

Next, we compared the final (simulated) peak temperature obtained by each of the considered approaches; the results are shown in Figure 3, depicting for the 40 instances the distribution (histogram) of the difference $\mathtt{T}_{\mathtt{HH}} - \mathtt{T}_{\mathtt{NN}}$ (in dark grey) and $\mathtt{T}_{\mathtt{PP}} - \mathtt{T}_{\mathtt{NN}}$ (in light grey). For the considered configuration, discrepancies of around 1-2°C were found to be already significant; as one can see, the network based approach is consistently better than the `HH` one and even improves (on average) the `PP` approach, which is known to use a very good temperature proxy measure.

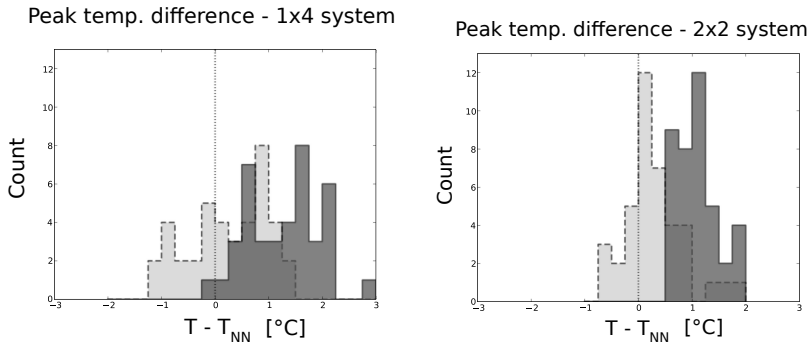

Fig. 3. Difference from `NN` in final peak temperature for the `HH` (dark grey) and the `PP` (light grey) approach

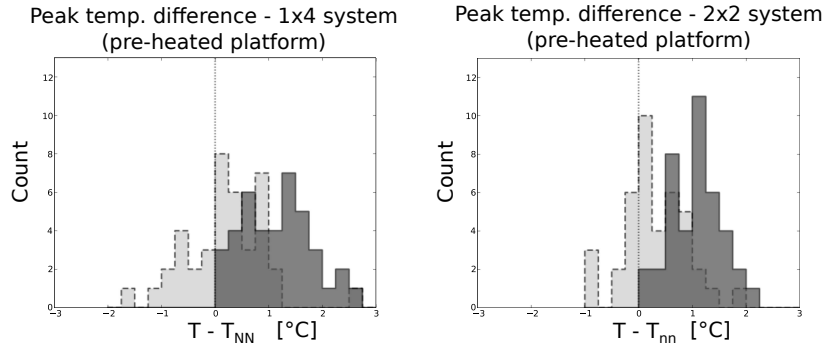

Fig. 4. Difference from `NN` in final peak temperature for the `HH` (dark grey) and the `PP` (light grey) approach – pre-heated platform

In order to investigate the effect of non-homogeneity, we performed a second evaluation after having asymmetrically pre-heated the target platforms; in this case the starting temperature for each core are 31.05°C, 33.55°C, 35.75°C, 36.48°C. The resulting differences in the final peak temperature are shown in Figure 4; as one can see the advantage of the `NN` approach becomes more relevant, due to the inability of the surrogate objective functions to capture the initial asymmetry.

Finally, Figure 5 is a scatter plot representing, for a sample workload instance, the assigned operating frequency and the CPI of each task in the `NN` and the `HH` solution; as one can see, the two plot are very similar, with low CPI tasks receiving high operating frequencies and high CPI ones usually running at 1600 MHz: as discussed earlier, this is a reasonable choice. However, while such information was *fed* to the `HH` approach by customization of the search strategy, the same relation has been *learned* by the Neural Network and *enforced via propagation*; by generalization of this reasoning, we conjecture a properly designed network has the chance of greatly reducing the effort in search strategy tuning.

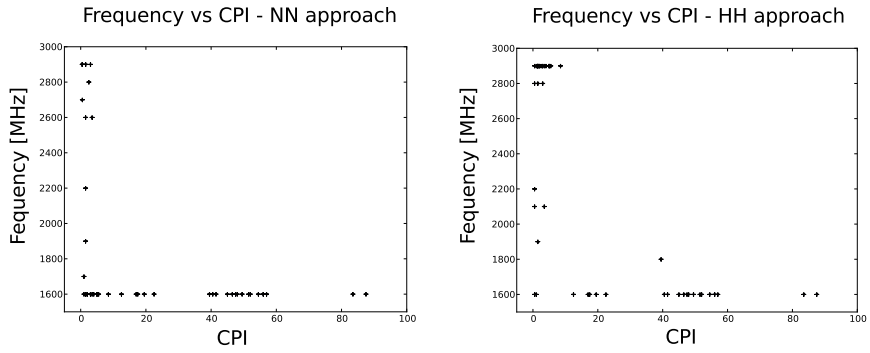

Fig. 5. (Frequency, CPI) values for each task in sample solution, for the `NN` and the `HH` approach.

7 Conclusions

We have introduced the idea of hybridizing Constraint Programming with a soft-computing paradigm, namely Neural Networks, to model complex real world problems; the novel Neuron (global) Constraint class provide a simple and yet effective tool to incorporate a trained network into a declarative CP model. As an important consequence training and designing the network becomes part of the modeling process; this involves deciding the parts of the target system to be represented via soft-computing and those to be tackled by more traditional means. To provide some evidence of the approach viability, we tackled a realistic thermal-aware workload allocation problem, with promising results.

Future research directions include experimentation with different real world problems, to investigate the applicability and effectiveness of the Neural Network integration approach to a broader set of target domains. Moreover, we are interested in improving the use of the Network provided information, e.g. search heuristics based on weight and connection structures could be designed. Finally, we plan to investigate the generalization of the approach to different soft-computing paradigms.

References

1. M. Bao, A. Andrei, P. Eles, and Z. Peng. On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration. In *Proc. of DAC2009*, pages 490–495. IEEE, 2009.
2. A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. A Distributed and Self-Calibrating Model-Predictive Controller for Energy and Thermal management of High-Performance Multicores. *Accepted for publication at DATE2011*, 2011.
3. A. Bartolini, M. Cacciari, A. Tilli, L. Benini, and M. Gries. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *Proc. of the 20th Great lakes symposium on VLSI*, pages 311–316. ACM, 2010.

4. A.K. Coskun, T.S. Rosing, and K.C. Gross. Utilizing predictors for efficient thermal management in multiprocessor SoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1503–1516, 2009.
5. A.K. Coskun, T.S. Rosing, and K. Whisnant. Temperature aware task scheduling in MPSoCs. In *Proc. of DATE2007*, pages 1659–1664. EDA Consortium, 2007.
6. L.V. Fausett. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall Englewood Cliffs, NJ, New Jersey, 1994.
7. Bhavishya Goel et. al. *Portable, scalable, per-core power estimation for intelligent resource management*. IEEE, August 2010.
8. R Hecht-Nielsen. Theory of the backpropagation neural network. *Neural Networks*, 1988.
9. Jason Howard et. al. *A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS*. IEEE, February 2010.
10. W Huang, S Ghosh, and S Velusamy. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on VLSI*, 14(5):501–513, 2006.
11. IBM Press Release. Netherlands Railways Realizes Savings of 20 Million Euros a Year With ILOG Optimization Technology. *Available at: http://www-03.ibm.com/press/us/en/pressrelease/27076.wss#release*, 2009.
12. INFORMS. Operations Research Success Stories. *Available at: http://www.scienceofbetter.org/can_do/success_alpha.php*, 2011.
13. W.S. McCulloch and W. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
14. M.L. Minsky and S. Papert. *Perceptrons: An introduction to computational geometry*, volume 1988. MIT press Cambridge, MA, Cambridge, MA, 1969.
15. S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli. Temperature Control of High-Performance Multi-core Platforms Using Convex Optimization. In *Proc. of DATE2008*, pages 110–115. IEEE, March 2008.
16. G. Paci, P. Marchal, F. Poletti, and L. Benini. Exploring temperature-aware design in low-power MPSoCs. *Proc. of DATE2006*, 3(1/2):836–841, 2006.
17. Dan Patterson. *Artificial Neural Networks. Theory and Applications*. Prentice Hall, Singapore, 1996.
18. D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres. Temperature-aware distributed run-time optimization on MP-SoC using game theory. In *Proc. of ISVLSI2008*, pages 375–380. IEEE, 2008.
19. F Rosenblatt. The perceptron: a perceiving and recognizing automaton (Technical Report 85-460-1), 1957.
20. Helmut Simonis. Constraint Application Blog. *Available at: http://hsimonis.wordpress.com/*, 2011.
21. Y. Xie and W.L. Hung. Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design. *The Journal of VLSI Signal Processing*, 45(3):177–189, 2006.
22. F. Zanini, D. Atienza, L. Benini, and G. De Micheli. Multicore thermal management with model predictive control. In *Proc. of ECCTD2009*, pages 711–714. IEEE, 2009.