Optimization and Controlled Systems: A Case Study on Thermal Aware Workload Dispatching

Andrea Bartolini and **Michele Lombardi** and **Michela Milano** and **Luca Benini**
DEIS, University of Bologna
{a.bartolini,michele.lombardi2,michela.milano,luca.benini}@unibo.it

Abstract

Although successfully employed on many industrial problems, Combinatorial Optimization still has limited applicability on several real-world domains, often due to modeling difficulties. This is typically the case for systems under the control of an on-line policy: even when the policy itself is well known, capturing its effect on the system in a declarative model is often impossible by conventional means. Such a difficulty is at the root of the classical, sharp separation between off-line and on-line approaches. In this paper, we investigate a general method to model controlled systems, based on the integration of Machine Learning and Constraint Programming (CP). Specifically, we use an Artificial Neural Network (ANN) to learn the behavior of a controlled system (a multicore CPU with thermal controllers) and plug it into a CP model by means of Neuron Constraints. The method obtains significantly better results compared to an approach with no ANN guidance. Neuron Constraints were first introduced in [Bartolini et al., 2011b] as a mean to model complex systems: providing evidence of their applicability to controlled systems is a significant step forward, broadening the application field of combinatorial methods and disclosing opportunities for hybrid off-line/on-line optimization.

Introduction

Advances in Combinatorial Optimization in the last decades have enabled their successful application to a large number of industrial problems. Nevertheless, many real-world domains are still impervious to such approaches. This is mainly due to difficulties in formulating an accurate declarative model of the system to be optimized.

This is typically the case for systems under the control of an on-line policy: even when the controller behavior is known in detail, capturing the effect of its actions in a declarative way is often out of the reach of conventional modeling techniques. Such a difficulty is at the root of the classical, sharp separation between off-line and on-line approaches. In this paper, we investigate a method to optimize decisions on controlled systems, based on the integration of Machine Learning and Constraint Programming. We propose to extract an approximate system representation, via Artificial Neural Networks (ANN), and plug such an approximation into a declarative model. In this way, the off-line optimizer has information about the behavior of the controlled system. The resulting approach is inherently off-line/on-line hybrid.

The use of ANNs for learning modeling components by means of the so-called Neuron Constraints was pioneered in [Bartolini et al., 2011b], where the authors tackle a thermal aware scheduling problem over a quad-core CPU. Thermal models are complex and multi-dimensional, usually formulated as large set of difference equations [Pedram and Nazarian, 2006; Huang, Ghosh, and Velusamy, 2006]. Their embedding in combinatorial solvers for optimal workload allocation is far from trivial. The method proposed in [Bartolini et al., 2011b] achieves such a demanding goal and is remarkably easy to implement. However, a limitation of the approach is that it assumes the system is thermally in *open-loop*: there is no feedback control on the system operation which continuously samples the operating temperature and forces safe operation by reducing performance when temperature emergencies arise. In practice, most of state-of-the-art multi-core platforms do implement closed-loop thermal controllers to prevent thermal runaway. In second place, the paper targets a small scale system with 4 cores only, providing little evidence of the method scalability to larger models.

The target platform considered in this paper is the SCC prototype CPU by Intel (Single-chip Cloud Computer, [Howard and Dighe et al., 2010]). This platform envisions future High Performance Computing (HPC) and datacenters processors. It features 48 cores connected by a Network on Chip *and is subject to the action of temperature regulation mechanisms*, with non-negligible impact on the performance. In particular, we consider: 1) a threshold-activated thermal controller and 2) a thermal aware Round Robin scheduling policy. We show how a reasonable model can be built and exploited for off-line workload allocation. The result is far from trivial, due to the large non-linearities introduced by the control policies. Optimization is carried on via Large Neighborhood Search, reporting consistently better results compared to an ANN free approach.

Problem Description

We tackle thermal aware workload dispatching on the experimental SCC CPU by Intel. Specifically, we want to map a set of heterogeneous tasks on the platform cores so as to maximize the overall efficiency, which is workload dependent due to the action of low-level thermal control strategies. In detail, we assume the presence of two specific temperature reduction mechanisms: since those are not yet fully implemented in the Intel prototype, we developed a powerful simulation framework for the platform and the controller.

The Target Platform

The SCC platform has 24 dual-core tiles arranged in a 6×4 mesh (overall, 48 cores in a 6×8 mesh). Each core runs and independent instance of the Linux kernel. Inter-tile communication occurs via message passing on a network interface, so that tasks cannot easily migrate between cores. The SCC API [Mattson et al., 2010] is designed to dispatch "batches" of jobs rather than single ones. Each tile contains two thermal sensors and each core contains a set of performance counters, suitable to track various architectural events (such as the number of retired instructions) at periodic intervals. SCC allows fine grain power management via Dynamic Voltage and Frequency Scaling (DVFS).

We developed for the SCC architecture a trace-driven simulator based on Matlab. The simulation framework is designed to take into account realistic temperature evolution and the effect of the operating frequency on the execution time (and power consumption) of each task. To simulate the SCC thermal evolution, depending on the chip workload and the room temperature, we use the Hotspot [Huang, Ghosh, and Velusamy, 2006] thermal model developed in [Sadri, Bartolini, and Benini, 2011]. The position of each core and the workload executed by its immediate neighbors have a strong impact on its steady state temperature.

Modern multicore platforms feature a hardware shutdown mechanism, switching-down the entire chip when critical temperatures are reached. To avoid this situation the operating system preventively slows down one or more cores when the temperature is higher than a given threshold. In an attempt to mitigate the resulting performance loss, thermal aware scheduling policies have been proposed [Coskun, Rosing, and Gross, 2009; Cuesta et al., 2010]. The basic idea is to exploit the different task attitude by spatially and temporally alternating the execution of *hot*-tasks and *cold*-tasks, leading to a lower and more stable temperature map. We implemented in our framework both these mechanisms.

Threshold Controller: At the bottom of the stack we implemented a threshold controller. The controller probes the core temperature every $2ms$: if it is higher than a threshold T_{max} ($80°C$) the frequency is switched to f_{min}; if it is lower than T_{min} ($78°C$) the core frequency is switched to f_{max}, with $f_{max} \gg f_{min}$. Reducing the operating frequency leads to increased task durations and reduced platform efficiency. Such a variation is smaller for tasks making frequent memory accesses (i.e. memory-bound), since memory waiting time is independent of the core frequency. Similarly to [Bartolini et al., 2011b], we characterize tasks via their average number of Clocks Per Instruction at maximum frequency (CPI). Small CPI values correspond to computation intensive tasks, while higher values are symptomatic of frequent memory accesses.

Thermal Aware Scheduler: On the top of the stack, for each core we implemented a thermal-aware, fair scheduler. This relies on separate *hot* and *cold* task queues and is triggered every $10ms$ (scheduling quantum):

1. The current temperature t_{cur} is probed and compared with that of the last quantum. Based on the result of the comparison, the expired task is classified as hot or cold and inserted in the corresponding queue.
2. The scheduler updates a moving average t_{avg} of the core temperature I.e. $t_{avg} = \alpha \cdot t_{avg} + (1 - \alpha) \cdot t_{cur}$, where α is a parameter in $]0, 1[$.
3. If $t_{cur} > t_{avg}$, the scheduler selects for execution the first task in the *cold* queue; conversely, the first task in the *hot* queue is selected. This mechanism mitigates thermal cycles keeping the temperature close to its average value.

In order to ensure fair execution, the scheduler keeps a counter for each task, incremented every time it is scheduled. At step 3, once the scheduler has selected the queue from which the next task has to be drawn, a check is performed to see if the associated counter is equal to a maximum value β. If this is the case, the task is skipped and the next one in the queue is selected. Once all the counters have reached the β value, they are reset. This mechanism ensures a Round Robin like behavior on a $\beta \cdot 10ms \cdot \#tasks$ time window. The implementation is compatible with the standard Linux scheduler.

Optimization Problem Formulation

Let T (with $|T| = n$) be the set of tasks t_i to be scheduled and C (with $|C| = m$) be the set of platform cores c_k. As in many realistic settings, we assume the duration of each task is unknown. The computation demand of t_i is measured by its CPI value at maximum frequency, i.e. CPI_i. Each task must be executed by a platform core. Tasks are preemptive, but no migration is possible. We assume the system runs at constant room temperature $temp$. The objective is to make the execution as fair as possible: since durations are not available, this is achieved by mapping roughly the same number of tasks to each core and by minimizing the penalties due to the threshold controller. On this purpose, we chose to enforce:

$$\lfloor {}^{n}/_{m} \rfloor \leq |T(c_k)| \leq \lceil {}^{n}/_{m} \rceil \tag{1}$$

where $T(c_k)$ is the set of tasks assigned to core c_k. Hard balancing constraints such as (1) are common in mapping policies for embedded systems. We decided to maximize the worst case platform efficiency[1]. The average efficiency eff_k of a core c_k over a time interval is defined as:

$$\mathit{eff}_k = \underbrace{\frac{1}{|T(c_k)|}}_{(A)} \cdot \underbrace{\sum_{t_i \in T(c_k)} \left(\frac{CPI_i^{-1}(f_{min}) \cdot f_{min}}{CPI_i^{-1}(f_{max}) \cdot f_{max}} \cdot tc + (1 - tc) \right)}_{(B)}$$

[1]Maximizing the cumulated efficiency of all cores has a high chance of hiding platform hot-spots.

where $CPI_i^{-1}(f) \cdot f$ denotes the instructions of task t_i processed per second at clock frequency f and tc is the ratio of the interval when the threshold controller is active. Both values can be measured at run-time by accessing the CPU counters. In practice, term (A) in the formula is the (inverse, normalized) number of instructions processed in the interval if T_{max} is never exceeded, while term (B) is the reduced value due to the controlled activation.

Related Work

Temperature management in MPSoCs has been attracting considerable research attention in recent years, pushed by the growing awareness that the development of modern multi-core platforms is about to hit a “thermal wall”. Classical temperature control techniques include abrupt reduction of the operating frequency, task migration or core shutdown, typically triggered when a specified threshold temperature is reached. These reactive methods avoid chip overheating, but 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 [Coskun, Rosing, and Gross, 2009; Coskun, Rosing, and Whisnant, 2007; Bartolini et al., 2011a; Zanini et al., 2009; Hanumaiah, Vrudhula, and Chatha, 2011], based on predictive models and taking advantage of run-time temperatures read from hardware sensors; (2) off-line allocation and scheduling approaches [Puschini et al., 2008; Murali et al., 2008], usually embedding a simplified thermal model of the target platform [Paci et al., 2006] or performing chip temperature assessment via a simulator [Bao et al., 2009; Xie and Hung, 2006].

Approaches based on learning a thermal model as an ANN have been recently proposed. In [Sridhar et al., 2012] the authors use an ANN as a proxy to optimize the thermal simulation on a GPU cluster. ANNs are shown to be capable of accurately modeling the heat equation of a thermal simulation. In [Ge, Malani, and Qiu, 2010] an ANN is used to embed thermal prediction in an on-line task migration and DVFS policy. [Jayaseelan and Mitra, 2009] uses instead an ANN to try and test on-line the micro-architectural configurations generated by a binary search algorithm, with the goal of finding performance optimal, thermally safe global system settings. [Hanumaiah, Vrudhula, and Chatha, 2011] shows that the thermal control and workload balancing of a multicore platform is a cyclic, complex and non-linear problem and suggests a set of approximations to reduce the problem complexity. Unfortunately the proposed simplifications neglect lateral heat, the thermal interaction between neighboring cores and specificities of the platform.

As a common trait, all those approaches focus on modeling the thermal behavior of the multicore system, but none of them takes into account the effects of a control policy. As a consequence, the proposed methods tend to be strictly off-line or on-line. Even when some multi-level optimization is performed (e.g. [Hanumaiah, Vrudhula, and Chatha, 2011; Jayaseelan and Mitra, 2009]), the off-line (or slow pace) optimizer is not actually *aware* of the behavior of the on-line controller, thus limiting the optimization opportunities.

Solution Method

In this paper, we investigate a general method to model controlled systems, based on the integration of Machine Learning and Constraint Programming (CP). Specifically, we use an Artificial Neural Network to learn the behavior of the CPU and the thermal controllers. The network is embedded into a CP model by means of Neuron Constraints, thus providing an off-line optimizer with knowledge about the on-line controllers and effectively making the approach an off-line/on-line hybrid.

ANN Design

We chose to use an ANN to predict the platform efficiency, given a task-to-core mapping and the room temperature value. Specifically, we introduce and train an ANN for each core c_k. The network output is the core efficiency eff_k, as defined earlier in this paper.

The value eff_k of a core has a negative dependence on its average temperature, as a consequence of the slow-down introduced by the threshold controller. Due to the specific implementation of the on-line thermal aware scheduler, we expect the temperature of a core c_k to be close to the average value determined by the composition of hot and cold tasks. This composition can be estimated by the average CPI among the tasks in $T(c_k)$. In second place, the average c_k temperature is sensitive to the temperature of the room and of the surrounding cores. In particular, the sensitivity to the latter has been demonstrated to decay as the distance between the cores grows [Bartolini et al., 2011a].

The reported considerations together with empirical tests motivated the choice of the input vector for each ANN. In particular, assuming core c_k is in position (i, j) on the 6×8 mesh, the input vector is composed by the values of: 1) the room temperature $temp$; 2) the average CPI of the tasks on the core[2], i.e. $ACPI_k$; 3) the average CPI of the four cardinal neighboring c_h, formally defined as those in the set $N(c_k)$ of cores in position $(i \pm 1, j \pm 1)$. More in detail, we use normalized input values:

$$\overline{temp} = \frac{temp - \mu_{temp} - \nu_{temp}}{\nu_{temp}} \quad (2)$$

$$\overline{ACPI}_k = \frac{ACPI_k - \mu_{CPI} - \nu_{CPI}}{\nu_{CPI}} \quad (3)$$

where μ and ν values are normalization parameters. Each normalized value ranges in $[-1, 1]$. If the core is in a corner or on a chip edge the missing neighbor input is replaced by replicating the $\overline{ACPI}_k$ value. Overall, this amounts to 6 inputs for each core. The output is the predicted efficiency for core $\overline{\mathit{eff}}_k$, normalized with $\mu_{\mathit{eff}} = 0$ and $\nu_{\mathit{eff}} = 1$.

We implemented and evaluated different network topologies, input configurations and activation functions. The best trade-off between ANN complexity and efficiency approximation is obtained by using a feed-forward three-layer network with 10 sigmoid neurons for the layer-0, 5 pure linear

[2]Where $ACPI_k = \frac{\sum_{t_i \in T(c_k)} CPI_i}{|T(c_k)|}$.

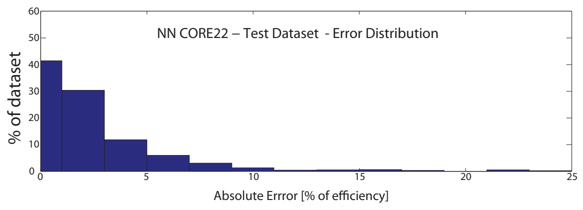

Figure 1: Neural Network Test Error histogram

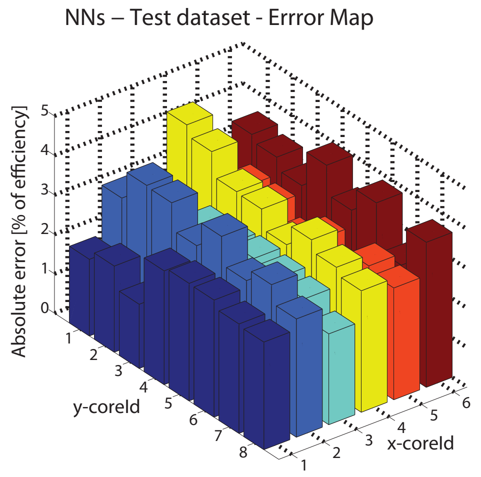

Figure 2: Neural Network Test Error histogram

neurons for the layer-1 and one sigmoid neuron for the output layer.

The training and test set for core k respectively consist of 2600 and 1000 randomly generated tuples, containing values for the inputs $\overline{temp}, \overline{ACPI}_k, \overline{ACPI}_h \forall c_h \in N(c_k)$. For the normalizations we assume $\mu_{temp} = 300°K$, $\nu_{temp} = 30°K$, $\mu_{cpi} = 0$, $\nu_{cpi} = 25$. Those values are suitable for $temp$ values in $[300°K, 360°K]$ and $CPI_i \in [0, 50]$. We then use the simulator to measure the actual efficiency corresponding to each tuple. Network training was performed via back-propagation, adjusting weights and bias according to the Levenberg-Marquardt algorithm [Hagan and Menhaj, 1994]. Figure 1 shows the prediction error on the test set for the core in position $(2, 2)$. The ANN estimation of the future efficiency has an error below 5% for more than the 80% of the test patterns. Figure 2 instead shows the average prediction error for all the cores, with their x and y coordinate respectively on the x and y axes. From the figure we can see that the average error is below the 5% for all the cores.

Embedding an ANN in a Constraint Model

An ANN can be plugged into a constraint model by means of Neuron Constraints, introduced in [Bartolini et al., 2011b]. Those are global constraints modeling a single artificial neuron. They come in floating point and integer versions: the latter has signature:

$$actfunction(\mathtt{Z}, [\mathtt{X_i}], [w_i], b, \pi)$$

where $\mathtt{Z}$ is an integer output variable, $[\mathtt{X_i}]$ is a vector of integer input variables, $[w_i]$, is a vector of real-valued weights and b is a bias; π is an integer precision value. The term "$actfunction$" denotes the neuron activity function. The constraint maintains Bound Consistency on the expression:

$$\mathtt{Z} = round\left(\pi \cdot actfunction\left(\pi \cdot b + \sum_i w_i \cdot \mathtt{X_i}\right)\right)$$

where it is assumed that the domain of $\mathtt{Z}$ and the domain of $\mathtt{X_i}$ variables is assumed to be scaled according to π. A full ANN can be integrated into a CP model by: 1) introducing a Neuron Constraint for each neuron; 2) adding an auxiliary integer variable to represent the output of each hidden neuron; 3) specifying the input/output of each Neuron Constraint according to the network structure and weights.

Model

We model the problem with Constraint Programming. The decision variables represent the core assignment:

$$\mathtt{CORE_i} \in \{0, \dots m-1\} \qquad \forall t_i \in T \tag{4}$$

where $\mathtt{CORE_i} = k$ iff task t_i is assigned for execution to core c_k. The restrictions on the minimum/maximum number of tasks are captured via Global Cardinality Constraints:

$$gcc([\mathtt{CORE_i}], \lfloor {}^n/_m \rfloor, \lceil {}^n/_m \rceil) \qquad \forall k \in C \tag{5}$$

where $[\mathtt{CORE_i}]$ denotes the vector of all $\mathtt{CORE_i}$ variables and $\lfloor {}^n/_m \rfloor$ and $\lceil {}^n/_m \rceil$ respectively are the minimum and maximum cardinalities for every domain value. The efficiency of each core is estimated using the ANNs described above. Formally, we add into the model, $\forall c_k \in C$:

$$ANN_k(\mathtt{EFF_k}, \mathtt{ACPI_k} \cup \{\mathtt{ACPI_h} : c_h \in N(c_k)\} \cup \mathtt{TEMP})$$

where the predicate ANN_k stands for the set of Neuron Constraints and auxiliary variables corresponding to the network for c_k. Variable $\mathtt{EFF_k}$ represents the normalized predicted efficiency of c_k. This is modeled adopting a fixed precision representation and takes discrete values in $\{-\pi, \dots \pi\}$, where π is the chosen precision value. Variables $\mathtt{ACPI_k} \in \{-\pi, \dots \pi\}$ denote the (normalized) average cpi value for core c_k and variable $\mathtt{TEMP} \in \{-\pi, \dots \pi\}$ is the (normalized) room temperature. The notation $N(c_k)$, which refers to the set of cores adjacent to c_k. Since the room temperature is constant, we have:

$$\mathtt{TEMP} = round(\pi \cdot \overline{temp}) \tag{6}$$

where $\overline{temp}$ is the normalized temperature and $round$ is the classical rounding operation. Average CPI values for each core are more complex to obtain. In first place, we introduce a set of auxiliary variables $\mathtt{X_{i,k}} \in \{0, 1\}$, connected to $\mathtt{CORE_i}$ via chaining constraints:

$$\mathtt{X_{i,k}} = (\mathtt{CORE_i} = k) \qquad \forall t_i \in T, c_k \in C \tag{7}$$

Then the $\mathtt{ACPI_k}$ variables are computed by means of a novel $average$ global constraint:

$$average(\mathtt{ACPI_k}, [round(\pi \cdot \overline{cpi}_i)], [\mathtt{X_{ik}}]) \quad \forall c_k \in C \tag{8}$$

where $\overline{cpi_i}$ are the normalized cpi_i. The $average$ constraint has signature $average(\mathtt{Z_i}, [w_i], [\mathtt{V_i}])$, where w_i are integer weights and $\mathtt{Z}, \mathtt{V_i}$ are integer variables. The constraint maintains Bound Consistency on the equality:

$$\mathtt{Z} = round\left(\frac{\sum_i w_i \cdot \mathtt{V_i}}{\sum_i \mathtt{V_i}}\right)$$

where $\mathtt{Z} = 0$ if all $\mathtt{V_i}$ are 0. The constraint filtering algorithm is not discussed here due to space limitations. The problem objective is to maximize the worst-case efficiency, i.e.:

$$\max \mathtt{Z} = \min_{c_k \in C} \mathtt{EFF_k} \tag{9}$$

The basic model structure is relatively simple, with the Neural Network capturing most of the problem complexity. It is worth noting that, for the considered 48 core platform, the model may grow pretty big (12,823 constraints and 12,577 variables for a 240 task instance), thus stressing the importance of the search strategy and of efficient filtering to obtain good quality solutions in reasonable time.

Search Strategy

Intuitively, low CPI tasks cause larger heat dissipation and are more likely to trigger the threshold controller than high CPI ones. Moreover, the temperature of each c_k is influenced by the workload of neighboring cores. Hence, as a general rule, higher platform efficiency is pursued by either spreading tasks uniformly according to their CPI_i or by surrounding heavy loaded cores by cooler ones. Driven by this intuition, we perform optimization via Large Neighborhood Search (LNS, [Shaw, 1998]).

Starting Solution: The starting solution is computed by spreading tasks so as to maximize the worst case average core CPI, i.e. $ACPI_k$. This is achieved as described in Algortihm 1: the basic underlying idea is to map an equal number of warm and cold tasks on each core. When mapping warm tasks, we favor c_k with high $ACPI_k$. When mapping cold tasks, we favor low $ACPI_k$ values, in an attempt to make the mapping as uniform as possible. More advanced heuristics may be considered in future work.

Algorithm 1 Initial Solution

```
1: let $\theta$ be the median $CPI_i$
2: build $\mathcal{W} = \{t_i : CPI_i < \theta\}$ and $\mathcal{C} = \{t_i : CPI_i \geq \theta\}$
3: for all $t_i$ in $\mathcal{W}$ by increasing $CPI_i$ do
4:     map $t_i$ to the $c_k$ with min $|T(c_k)|$. Break ties by max $ACPI_k$, then randomly
5: for all $t_i$ in $\mathcal{C}$ by decreasing $CPI_i$ do
6:     map $t_i$ to the $c_k$ with min $|T(c_k)|$. Break ties by min $ACPI_k$, then randomly
```

Algorithm 1 is designed to take into account Constraints (5) and hence always finds a feasible solution. Once this is available, we start the LNS iterations, trying to improve the objective value by exploiting platform heterogeneity and neighborhood interactions.

Inner Search Strategy: Each LNS attempt makes use of a random variable/value selection strategy, restarted with a varying fail limit according to the Luby sequence [Luby, 1993]. The classical fail limit for each attempt is multiplied by a factor γ. Each LNS iteration ends when an improving solution is found, or when the fail limit becomes higher than a stop value Γ. An improving solution is required to be better than the current incumbent by at least $\pi/100$ units.

This setting was chosen after preliminary experimentation, including more sophisticated search strategies. The rationale for the choice is to focus search on neighborhoods containing a significant number of improving solutions. In this situation a highly randomized search strategy is likely to yield the best results and quickly identify good quality mappings. Since the ANN model is approximated, finding the actual problem optimum is of limited interest due to the possible estimation errors.

Relaxation Strategy: At each LNS iteration, in order to have any chance of improving the solution, it is necessary to relax at least a few decision variables from the cores having minimum $\mathtt{EFF_K}$. After a preliminary experimentation, we ended up with the relaxation strategy discussed in Algorithm 2, where eff^* denotes the worst case efficiency in the incumbent solution.

Algorithm 2 Relaxation Strategy

```
1: Identify the set $WC$ of cores causing the current objective value, i.e. $WC = \{c_k : \mathtt{EFF_k} \leq eff^* + \pi/100 - 1\}$. Let $n_w$ denote $|WC|$.
2: Select from $C \setminus WC$ the $2 \cdot n_w$ cores having the highest value for $\mathtt{EFF_k} + \mathtt{ACPI_k}$. Then build a set $BC$ by randomly choosing $n_w$ cores from the previous set.
3: Build a set $NC$ by selecting $n_w$ random cores from $\bigcup_{c_k \in WC} N(c_k) \setminus BC$
4: Relax all $\mathtt{CORE_i}$ variables such that $c_{\mathtt{CORE_i}} \in WC \cup BC \cup NC$
```

As an intuition, we completely free the cores c_k having the worst case efficiency (taking into account the optimization step $\pi/100$). For each such c_k, we additionally free a neighbor core and a random "good" core. Good cores are those having either high efficiency or high average CPI. The underlying rationale is trying to improve the efficiency of the critical c_k by moving tasks from less constrained cores and by exploiting neighborhood effects.

Experimentation

In order to test the proposed method we generated two groups of synthetic workloads, each group consisting of 10 sets of 240 tasks to be mapped on the target platform. Tasks in each set are either computation-bound (CPI_i normally distributed with mean 0.75 and standard deviation 0.25) or memory-bound (CPI_i normally distributed with mean 35 and standard deviation 5)[3]. Low CPI values are capped at 0.5, high values at 50. Benchmark Group A contains 75% computation-bound and 25% memory-bound tasks, while the mix is 85%/15% for Group B. The room temperature for all test was fixed to $310K$ and the parameters for each

[3]This values represent real corner cases of SCC workloads [Sadri, Bartolini, and Benini, 2011]

	#	FIRST	ANN	CPI	RND
Group A	0	78.8% (2.6)	87.9% (0.5)	78.1% (2.0)	24.4% (15.8)
	1	78.4% (2.1)	87.2% (0.8)	78.3% (1.9)	33.3% (14.7)
	2	78.5% (0.6)	87.9% (0.5)	78.5% (0.6)	24.7% (15.1)
	3	77.8% (2.0)	87.7% (0.5)	77.6% (2.4)	27.3% (14.4)
	4	77.9% (1.7)	87.3% (0.9)	78.4% (1.8)	32.1% (16.6)
	5	78.0% (2.1)	87.6% (0.6)	77.8% (1.8)	27.2% (12.9)
	6	77.3% (2.4)	87.9% (0.7)	78.4% (2.1)	26.3% (13.3)
	7	78.3% (1.9)	87.8% (0.6)	78.3% (1.9)	29.3% (13.6)
	8	78.0% (2.7)	88.1% (0.4)	78.0% (2.7)	31.0% (15.5)
	9	78.4% (1.6)	87.4% (0.5)	78.4% (1.6)	26.9% (17.7)

Table 1: Solution Quality after 60 sec., on Group A

	#	FIRST	ANN	CPI	RND
Group B	0	42.2% (7.5)	77.7% (0.7)	42.2% (7.5)	24.5% (12.9)
	1	48.3% (7.9)	77.7% (0.5)	48.4% (7.9)	26.1% (11.9)
	2	37.9% (5.2)	78.1% (0.9)	38.0% (5.2)	27.9% (9.6)
	3	41.4% (15.2)	77.6% (0.7)	41.4% (15.3)	25.7% (10.7)
	4	39.5% (7.1)	79.3% (0.5)	39.5% (7.1)	23.2% (8.8)
	5	42.3% (11.6)	78.6% (0.7)	42.3% (11.7)	25.0% (9.3)
	6	39.9% (10.0)	78.0% (0.7)	39.6% (9.7)	20.5% (11.4)
	7	41.6% (7.6)	77.5% (0.9)	41.6% (7.6)	25.3% (12.6)
	8	39.1% (7.4)	77.4% (0.5)	39.1% (7.5)	23.1% (7.7)
	9	40.7% (11.0)	77.3% (1.1)	40.9% (11.3)	21.6% (9.2)

Table 2: Solution Quality after 60 sec., on Group B

#	1s	2s	5s	20s
0	86.1%, 74.3%	86.4%, 76.9%	87.3%, 77.2%	87.6%, 77.6%
1	85.9%, 75.0%	86.3%, 76.0%	86.3%, 77.0%	87.1%, 77.2%
2	86.3%, 75.9%	86.6%, 77.1%	87.2%, 77.5%	87.6%, 77.8%
3	85.7%, 75.5%	86.1%, 76.5%	86.5%, 77.3%	87.3%, 77.5%
4	85.8%, 76.5%	86.3%, 77.8%	86.6%, 78.3%	87.2%, 79.1%
5	85.9%, 76.3%	86.8%, 77.3%	87.0%, 77.8%	87.5%, 78.5%
6	86.1%, 75.1%	86.7%, 75.7%	87.1%, 77.2%	87.4%, 77.8%
7	85.2%, 74.7%	86.1%, 76.6%	86.8%, 77.0%	87.5%, 77.4%
8	86.0%, 72.4%	86.8%, 76.3%	87.5%, 76.6%	87.9%, 77.2%
9	86.0%, 74.8%	86.7%, 76.2%	86.8%, 76.4%	87.1%, 77.1%

Table 3: Solution Quality after N sec., on Group A and B

LNS iterations are $\gamma = 5, \Gamma = 1000$. The precision value π is 1000. The approach is implemented using the Google or-tools CP solver [Perron, 2011]. The Neuron Constraints are written in C++, while the search method is in Python.

Solution Quality

We performed a first experimentation to assess the expected quality of the mapping provided by our approach. On this purpose, our method was compared with a pure random allocation (serving as baseline) and with a simplified approach making use of no ANN. The latter was obtained by removing the network representation from the model and by using the $\mathtt{ACPI_k}$ variables as a proxy for the core efficiencies. The problem objective becomes $\max \mathtt{Z} = \min_{c_k \in C} \mathtt{ACPI_k}$ and high quality cores in the LNS relaxation strategy are identified based solely on their average CPI.

The search being largely randomized, each workload instance was solved 10 times, with different seeds for the random number generator. All tests were run on an Intel Core i7 2.8GHz, with a time limit of 60 seconds (the solution quality tends to stabilize within this limit). Table 1 reports for each workload the results of this first experimentation, showing for each approach the mean quality of the final solution and the corresponding standard deviation (between round brackets). Specifically, column ANN refers to our approach, CPI is the method with the simplified model and RND is the baseline random mapping. Column FIRST reports the quality of the first solution (the same for ANN and CPI).

Exploiting the ANN guidance to take advantage of platform non-homogeneity and neighborhood effects allows an average 6-8% improvement. Since this (predicted) value is about twice the value of the average prediction error, it seems reasonable to think that the obtained solutions achieve an actual efficiency gain. By increasing the ratio of computation intensive tasks in the mix (i.e. moving to Group B), platform peculiarities not captured by the average CPI model become more relevant and the ANN advantage grows to more than 30%. As discussed in the Conclusion section, we expect the actual (simulated) gain to be more limited, but those are nevertheless very relevant improvements.

Convergence Rate

Workload dispatching routines as the one described in this work are likely to be run relatively often at execution time. It is therefore interesting to assess the solver ability to quickly provide high quality solutions. On this purpose we measured the solution value evolution over time. Table 3 reports this information for Group A and B. Each cell reports the mean solution quality obtained by the ANN approach (for the two groups) after the specified number of seconds of search time. As one can see, values close to the final solution are reached already after $\sim 5s$. The convergence rate is very fast, given the size of the search space, and sufficient for dispatching batches of medium-long jobs at run-time. Higher solver efficiency should be achievable with further research effort.

Concluding Remarks

We have discussed a method to model controlled systems in Combinatorial Optimization. The main idea is to use machine learning (here, an ANN) to extract an approximate description of the system and plug it into a conventional combinatorial model (here, using CP). The method provides a general and novel way to combine off-line and on-line optimization. We have shown the approach feasibility on a workload dispatching problem for a multi-core CPU with multiple on-line controllers. As a final stage, the obtained solutions should be validated on the target systems. According to a preliminary experimentation, this will likely provide an interesting scientific topic. In fact, the optimizer apparently tends to direct search to solutions for which the ANN provides a poor prediction. We plan to address this issue in a number of ways, including the use of the optimizer to augment the training set until sufficient stability is reached. As a natural extension, we plan to apply this methodology on the real SCC platform. We also plan to tackle thermal aware dispatching on HPC and Data-centers, since the problem has many similarities to the one we have addressed.

Acknowledgements This project is partially funded by a Google Focused Grant on Mathematical Optimization and Combinatorial Optimization in Europe, with title "Model Learning in Combinatorial Optimization: a Case Study on Thermal Aware Dispatching", the EU FP7 ERC Project MULTITHERMAN (GA n. 291125) and the EU FP7 Project Therminator (GA n. 248603).

References

Bao, M.; Andrei, A.; Eles, P.; and Peng, Z. 2009. On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration. In *Proc. of DAC*, 490–495. IEEE.

Bartolini, A.; Cacciari, M.; Tilli, A.; and Benini, L. 2011a. A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores. In *Proc. of DATE*, 830–835.

Bartolini, A.; Lombardi, M.; Milano, M.; and Benini, L. 2011b. Neuron Constraints to Model Complex Real-World Problems. *Proc. of CP* 115–129.

Coskun, A.; Rosing, T.; and Gross, K. 2009. Utilizing predictors for efficient thermal management in multiprocessor SoCs. *IEEE Trans. on CAD* 28(10):1503–1516.

Coskun, A.; Rosing, T.; and Whisnant, K. 2007. Temperature aware task scheduling in MPSoCs. In *Proc. of DATE*, 1659–1664. EDA Consortium.

Cuesta, D.; Ayala, J.; Hidalgo, J.; Atienza, D.; Acquaviva, A.; and Macii, E. 2010. Adaptive task migration policies for thermal control in mpsocs. *Proc. of ISVLSI* 110–115.

Ge, Y.; Malani, P.; and Qiu, Q. 2010. Distributed task migration for thermal management in many-core systems. In *Proc. of DAC*, 579–584. IEEE.

Hagan, M., and Menhaj, M. 1994. Training feedforward networks with the marquardt algorithm. *Neural Networks, IEEE Transactions on* 5(6):989–993.

Hanumaiah, V.; Vrudhula, S.; and Chatha, K. S. 2011. Performance Optimal Online DVFS and Task Migration Techniques for Thermally Constrained Multi-Core Processors. *IEEE Trans. on CAD* 30(11):1677–1690.

Howard, J., and Dighe et al., S. 2010. A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS. *Proc. of ISSCC* 108–109.

Huang, W.; Ghosh, S.; and Velusamy, S. 2006. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Trans. on VLSI* 14(5):501–513.

Jayaseelan, R., and Mitra, T. 2009. A hybrid local-global approach for multi-core thermal management. In *Proc. of ICCAD*, 314. New York, New York, USA: ACM Press.

Luby, M. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47(4):173–180.

Mattson, T.; Van der Wijngaart, R.; Riepen, M.; Lehnig, T.; Brett, P.; Haas, W.; Kennedy, P.; Howard, J.; Vangal, S.; Borkar, N.; Ruhl, G.; and Dighe, S. 2010. The 48-core scc processor: the programmer's view. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, 1 –11.

Murali, S.; Mutapcic, A.; Atienza, D.; Gupta, R.; Boyd, S.; Benini, L.; and De Micheli, G. 2008. Temperature Control of High-Performance Multi-core Platforms Using Convex Optimization. In *Proc. of DATE*, 110–115. IEEE.

Paci, G.; Marchal, P.; Poletti, F.; and Benini, L. 2006. Exploring temperature-aware design in low-power MPSoCs. *Proc. of DATE* 3(1/2):836–841.

Pedram, M., and Nazarian, S. 2006. Thermal Modeling, Analysis, and Management in VLSI Circuits: Principles and Methods. *Proceedings of the IEEE* 94(8):1487–1501.

Perron, L. 2011. Operations Research and Constraint Programming at Google. In *Proc. of CP*, 2.

Puschini, D.; Clermidy, F.; Benoit, P.; Sassatelli, G.; and Torres, L. 2008. Temperature-aware distributed run-time optimization on MP-SoC using game theory. In *Proc. of ISVLSI*, 375–380. IEEE.

Sadri, M.; Bartolini, A.; and Benini, L. 2011. Single-chip cloud computer thermal model. In *Thermal Investigations of ICs and Systems (THERMINIC), 2011 17th International Workshop on*, 1 –6.

Shaw, P. 1998. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proc. of CP*, 417–431.

Sridhar, A.; Vincenzi, A.; Ruggiero, M.; and Atienza, D. 2012. Neural Network-Based Thermal Simulation of Integrated Circuits on GPUs. *IEEE Trans. on CAD* 31(1):23–36.

Xie, Y., and Hung, W. 2006. Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design. *The Journal of VLSI Signal Processing* 45(3):177–189.

Zanini, F.; Atienza, D.; Benini, L.; and De Micheli, G. 2009. Multicore thermal management with model predictive control. In *Proc. of ECCTD*, 711–714. IEEE.