Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 403086, 11 pages
doi:10.1155/2008/403086

Research Article

A Game-Theoretic Approach for Run-Time Distributed Optimization on MP-SoC

Diego Puschini,[1] Fabien Clermidy,[1] Pascal Benoit,[2] Gilles Sassatelli,[2] and Lionel Torres[2]

[1] *CEA /Leti-Minatec, 17 rue des Martyrs, 38054 Grenoble Cedex 9, France*
[2] *LIRMM, University of Montpellier 2/CNRS UMR 5506, 161 rue Ada, 34392 Montpellier Cedex 5, France*

Correspondence should be addressed to Fabien Clermidy, fabien.clermidy@cea.fr

Received 29 February 2008; Accepted 12 August 2008

Recommended by Michael Hubner

With forecasted hundreds of processing elements (PEs), future embedded systems will be able to handle multiple applications with very diverse running constraints. Systems will integrate distributed decision capabilities. In order to control the power and temperature, dynamic voltage frequency scalings (DVFSs) are applied at PE level. At system level, it implies to dynamically manage the different voltage/frequency couples of each tile to obtain a global optimization. This paper introduces a scalable multiobjective approach based on game theory, which adjusts at run-time the frequency of each PE. It aims at reducing the tile temperature while maintaining the synchronization between application tasks. Results show that the proposed run-time algorithm requires an average of 20 calculation cycles to find the solution for a 100-processor platform and reaches equivalent performances when comparing with an offline method. Temperature reductions of about 23% were achieved on a demonstrative test-case.

1. Introduction

"MP-SoC is not just coming: it has arrived" [1]. Intel's Tera Scale computing research program [2] is one illustration of today's reality: a prototype is composed of 80 processors interconnected with a network-on-a-chip (NoC) [3, 4]. In a near future, an increasing number reconfigurable processing elements on a single chip is forecasted, leading to new challenges for system-level designers. From technological side, the variability between dies requires chip-by-chip optimization. Offline methods are no more possible when hundred of processors are integrated on advanced technologies. Thus, on-chip distributed techniques are required to adjust the parameters of each chip. On the applicative side, MP-SoC platforms will support several applications, including someone unknown at design time. Then, run-time adaptability is a requirement to optimize applicative parameters.

In this paper, we consider MP-SoC platforms integrating hundreds of reconfigurable processing elements (PEs) as shown in Figure 1, each includes processors, memories, and peripherals. It is assumed that each PE embeds the required components to locally adjust its parameters. For instance, dynamic voltage/frequency scaling (DVFS) tunes the voltage/frequency couple of each tile. In this context, an important issue is how to manage the tradeoff between the performance achieved and the die temperature which is an indirect indicator of the power consumption. It is a multiobjective optimization problem [5]: how to solve the power and temperature management for hot-spot reduction [6] taking into account the performance control through task synchronization for a given application with functional dependencies [7].

The main contribution of this paper is the use of the game theory [8] as a model to dynamically optimize MP-SoC platforms in a distributed way. A run time game-theoretic method to choose the frequency for each PE in MP-SoC platforms in order to optimize the circuit temperature taking into account the task synchronization has been presented in [9]. In this article, this technique is reviewed. A statistical analysis regarding the algorithm convergence and scalability is studied and a demonstrative test-case is presented.

1.1. Problem Formulation

Consider an MP-SoC architecture composed of several reconfigurable processing elements (PEs). It is assumed that future MP-SoC will integrate a large number of PEs leading to the choice of a distributed control architecture. Each PE integrates devices such as processors, memories, peripherals, and dynamic voltage/frequency scaling (DVFS). The choice of distributed DVFS is justified in [10, 11] while in [12, 13], a whole demonstrator is presented.

It is assumed that synchronous PEs are connected by an asynchronous 2D-mesh NoC as in [14]. The interconnect system offers the required bandwidth and latency for the targeted applications, as well as the ability of individual frequency selection.

Consider an application composed of n tasks T_i and connections as illustrated by the example described in Figure 2(a). It is assumed that the application is supported by the MP-SoC and it is mapped on the platform by a given mechanism. At application level, the functional dependencies between tasks lead to adjust the frequency of each PE in order to guarantee the task synchronization [7].

At physical level, the frequency choice is influenced by the temperature metric. In this paper, we consider a first-order approximation that the temperature of a given PE is affected by its neighbors [6, 15], as shown in Figure 2(b). The two constraints meet when the application is mapped on the MP-SoC (Figure 2(c)). A tradeoff between the synchronization and the temperature metrics has to be solved. It is a multiobjective optimization problem we solve in this paper.

1.2. Paper Organization

The paper is organized as follows. Section 2 presents some related works regarding dynamic and static optimizations for embedded reconfigurable systems. The basic formalisms, definitions, and notations of game theory are presented in Section 3. In Section 4, we present the multiobjective optimization notation and we formulate our task synchronization and temperature models. Based on discussed models and using the preliminary presented theory, a game-theoretic optimization algorithm is proposed. Simulations were performed on several scenarios to study the feasibility of this approach in the MP-SoC context. Section 5 analyzes the dynamic behavior and its optimization quality. Finally, a demonstrative test case is evaluated in Section 6.

2. Related Works

Several optimization methods are used to get the best tradeoff between system metrics for a given architecture. For instance, designers try to obtain the best performance on power consumption ratio (MIPS/mW). Optimization may be applied at different stages, either at design time (static optimization) or at run time (dynamic optimization), or both, as described in the following paragraphs.

Several works propose some static optimization techniques for given metrics. In [16], the authors developed a new framework based on integer linear programming (ILP) solvers and constraint programming to solve at design time the task allocation/scheduling problem. In this domain, there are also several significant contributions such as [17, 18] for workload optimization and [19] for energy savings.

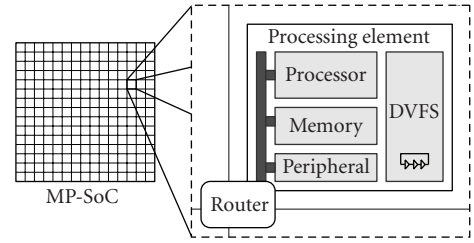

Figure 1: MP-SoC with hundreds of processing elements connected by an NoC. Each PE integrates a DVFS.

More recently, there is a growing interest for run-time optimization. When dynamic methods are considered, most of the proposed approaches are global techniques: the optimization decisions are taken considering the whole system. In [20], heuristics for optimal task placement are discussed in an NoC-based heterogeneous MP-SoC. There are several approaches where tasks are moved in order to balance computation workload and homogeneously dissipate the power, for example, in [6, 21]. The slow reaction time, the requirement of unused tiles, and the memory or the important number of data transfer between tiles can limit the use of these techniques for certain applications. Moreover, the implementation of these techniques is limited to functionally homogeneous arrays.

In [22, 23], authors propose a design time Pareto exploration and characterization combined with run-time management [24]. In [15], a convex optimization method is used in a 2-phase algorithm that allows frequency assignment on an MP-SoC controlling hot-spots. These approaches become prohibitive as the applications should be completely characterized at design time for an efficient implementation. In [25, 26], voltage and frequency are chosen at run time by centralized mechanisms. These solutions do not scale with the number of PEs and then do not well match for future multiprocessor platforms [1].

Our contribution is based on the use of game theory for dynamic optimization. Game theory has been widely used in other domains such as economy, sociology, and biology. In the VLSI context, in [27], game theory has been firstly used for circuit synthesis. To the best of our knowledge, it has never been applied to run-time optimization of embedded systems.

3. Game Theory

The objective of this section is to introduce the notations used in the game theory and the common solution known as the Nash equilibrium (NE) [28]. Game theory was introduced by Von Neumann and Morgenstern [8]; it can be viewed as a branch of applied mathematics to study interactions among rational individuals or decision makers.

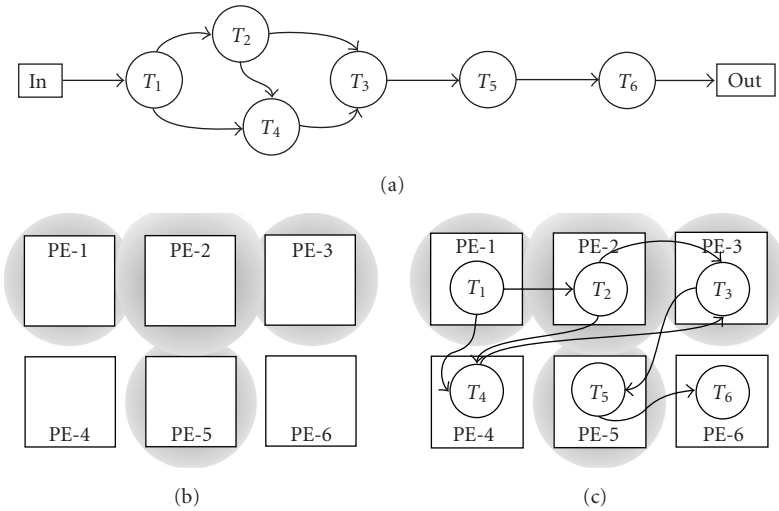

Figure 2: (a) An application example with 6 tasks: each task has to be processed at a given frequency to guarantee the synchronization between logical neighbors. (b) The MP-SoC composed of 6 PEs: the temperature of each PE is affected by physical neighbors. (c) The application is mapped on the MP-SoC: applicative and physical requirements are merged on a two-metric optimization problem.

3.1. Noncooperative Games

A game is a scenario with several players interacting by actions and consequences [29]. Basically, the players, or decision makers, individually choose how they act, resulting in consequences or outcomes. Each decision maker tries to maximize its own outcome according to its preferences, leading to a global optimization. Since the decision is made without need to the cooperation of other players, the game is called *noncooperative*.

Mathematically, such a game in *normal form* (normal form is the way in which the game is described) with n players is described as

$$\Gamma = \{N, S_i, u_i\}, \quad \forall i \in N, \tag{1}$$

where $N = \{1, \ldots, n\}$ is the set of players; S_i is the set of actions for players i; and u_i is a function describing its outcomes. The discrete set of actions or *strategies* of player i is defined as

$$S_i = \{s_{i1}, s_{i2}, \ldots, s_{im_i}\}, \tag{2}$$

where m_i is the number of possible actions for this player. The outcome of player i is represented by a score: the higher the score, the nearer the optimal point. Because of dependencies with other players, the score or *utility* is a function of choices from current player and choices from other players:

$$u_i : S_1 \times S_2 \times \cdots \times S_n \longrightarrow \Re. \tag{3}$$

For MP-SoC platforms, the choice of the game type to be applied is driven by the complexity, for low-cost implementation, and the number of steps of the game for the run-time feature. For these reasons, the *noncooperative normal-form simultaneous repeated game* model has been chosen.

3.2. The Nash Equilibrium

Nash proved that each n-player noncooperative game has at least one equilibrium point, known as Nash equilibrium (NE). It can be defined by *pure strategies* or by *mixed strategies*. In pure strategies, the solutions are obtained by allowing only one action per player. On the other side, in mixed strategies, the solution is chosen in a set of actions, each played with a given probability [28]. For pure strategies, an equilibrium point is defined as follows.

Definition 1. For a given game $\Gamma = \{N, S_i, u_i\}$, a solution

$$S^* = \{s_1^* \in S_1,\ s_2^* \in S_2, \ldots, s_n^* \in S_n\} \tag{4}$$

is a Nash equilibrium if for all $i \in N$,

$$u_i\{s_1^*, \ldots, s_i^*, \ldots, s_n^*\} \geq u_i\{s_1^*, \ldots, s_i, \ldots, s_n^*\}, \tag{5}$$

where s_i^* is the Nash equilibrium strategy for player i.

Note that in an NE, players cannot improve their outcomes by unilaterally changing their strategies, indicating then that it is at least a local maximum.

4. Game-Theoretic Multiobjective Optimization

In this section, a new algorithm based on game theory is proposed to solve the multiobjective optimization issue at run time. The mathematical description of multiobjective optimization problems is formulated. Then, the system-level metrics are discussed and finally, the game-theoretic model and the distributed optimization algorithm are proposed.

4.1. Multiobjective Optimization

The multiobjective optimization problem is defined as how to find a vector $X = (x_1, x_2, \ldots, x_q)$ that optimizes a vector

function $\vec{O}$ whose elements represent l-normalized objective functions $O_j(X)$. These functions are mathematical descriptions of performance criteria, usually in conflict. That is,

$$\vec{O}(X) = (O_1(X), O_2(X), \ldots, O_l(X)), \tag{6}$$

where $\vec{O}(X)$ produces solutions in an l-dimension space. The problem is converted in a single objective optimization problem thanks to the linear combination of the metrics [5]:

$$O'(X) = w_1 O_1 + w_2 O_2 + \cdots + w_l O_l, \tag{7}$$

where $\sum_{j=1}^{l} w_j = 1$ and $w_j \geq 0$ for $j = 1, \ldots, l$. The weights w_j represent the importance of each metric in the combination $O'(X)$.

This principle combined with the game theory can be used to optimize the task synchronization when minimizing the temperature of the blocks by the run-time selection of the PE frequency. Thus, (7) can be written for each PE of the MP-SoC as

$$O_i = w_{syn,i} O_{syn,i}(F) + w_{temp,i} O_{temp,i}(F), \tag{8}$$

where $F = (f_1, f_2, \ldots, f_n)$ is the frequency of each PE, $O_{syn,i}(F)$ represents the synchronization function, and $O_{temp,i}(F)$ is the temperature metric for each PE. Finally, $w_{syn,i}$ and $w_{temp,i}$ are the relative importance of each metric in the optimization process.

4.2. System-Level Metrics

4.2.1. Task Synchronization Model

The application synchronization problem is defined as the choice of the best working frequency for each processor. In [30], authors try to equalize the input and outputs rates of application nodes in order to maintain the processor rate. Based on this principle, we propose a simple task synchronization model based on the load assigned to each PE and its frequency.

Assume that block i supports the task T_i and transfers the generated data to block j, which is running the task T_j. They are synchronized when they are working at the same mean performance, that is, the data produced by block i are entirely consumed by block j. Otherwise, if block i is faster than block j, some data produced by the first block are not immediately consumed by the second one. On the contrary, if block i is slower than j, the second one will have undesirable idle cycles. In both cases, they are not synchronized.

Following this principle, we define the synchronism between two blocks by the following equation:

$$A_{syn_{i,j}} = -\left| \frac{\alpha_i f_i}{L_i} - \frac{\alpha_j f_j}{L_j} \right|, \tag{9}$$

where f_i and f_j are the frequencies of blocks i and j, L_i and L_j are their respective computation loads. Parameters α_i and α_j are considered to normalize heterogeneous PEs. In the rest of the paper, they are assumed to be $\alpha_i = \alpha_j = 1$ for homogeneous MP-SoC. The result of this equation is zero when the blocks are synchronized, and a negative value otherwise.

Equation (9) can now be used to build the first metric for each PE as follows:

$$O_{syn,i} = -\frac{1}{k_i}\left[A_{syn_{i,1}} \;\; A_{syn_{i,2}} \;\; \ldots \;\; A_{syn_{i,n}}\right] C_i, \tag{10}$$

where k_i is the number of connections and C_i is the connectivity vector of task T_i for the application:

$$C_i = \left[c_{i1} \;\; c_{i2} \;\; \ldots \;\; c_{in}\right]^T, \tag{11}$$

where $c_{ij} = 1$ if task i is connected to task j and $c_{ij} = 0$ otherwise.

Finally, the problem of global task synchronization results in the choice of the frequency f_i that maximizes $O_{syn,i}$ for each PE.

4.2.2. Temperature Model

Considering a fixed voltage, the temperature of a block depends on the power consumption and the effect induced by other blocks. On a simplified model, we neglect the static consumption. We consider the dynamic power consumption of the PE as $P_i = \beta_i f_i V^2$ with f_i the clock frequency, V the supply voltage and β_i a given circuit constant.

Following [31], the transfer thermal resistance R_{ij} of PE_i with respect to PE_j is

$$R_{ij} = \frac{\Delta T_{ij}}{\Delta P_j}, \tag{12}$$

where ΔT_{ij} is PE_i temperature rise due to the power ΔP_j dissipated by PE_j.

The temperature of each PE in an n-processor array is calculated by

$$T_{\mathrm{PE}_i} = R_{i1} P_1 + R_{i2} P_2 + \cdots + R_{in} P_n, \tag{13}$$

where $P_1, \ldots, P_n$ are the power consumptions of the n-PEs. For simplification purposes, it is assumed in this work that the PEs are only affected by the power dissipated by the nearest neighbors. In a regular 2-dimension mesh array, it means the PEs located at the north, south, east, and west positions.

Finally, expression (13) is reformulated for each PE in order to express the reduction of temperature as an optimization metric:

$$O_{temp,i} = -R_i\left[P_1 \;\; P_2 \;\; \ldots \;\; P_n\right]^T \tag{14}$$

with R_i a vector containing the transfer thermal resistances R_{ij} of PEs affecting the temperature of PE_i.

4.3. Game-Theoretic Method

The tradeoff problem discussed in Section 1.1 is modeled as a game (as explained in Section 3) with multiple objectives (Section 4.1). The components of such a game are identified and an algorithm is proposed to solve the formulated problem.

```
Require: u_i, MyStgy, OtherStgies
Ensure: NewStgy
    NewStgy ⇐ MyStgy
    for all Stgy do
        if u_i(Stgy, OtherStgies) > u_i(NewStgy, OtherStgies)
        then
            NewStgy ⇐ Stgy
        else
            NewStgy ⇐ NewStgy
        end if
    end for
```

Algorithm 1: UnilaterallyMax(player-i).

4.3.1. Game Model

As stated in Section 3.1, a game Γ is composed of players (N), a set of actions per player (S_i), and the outcomes (u_i). For the given tradeoff problem, the PEs will be considered as the player set. The tradeoff variable is the clock frequency of each PE. Thus, the set of actions of each player is each possible frequency f_i. Setting the frequency step allows to deduce the strategy space S_i. For example, with a 5 MHz step into the bounds of 100 to 300 MHz, the strategy space will be as follows:

$$S_i = \{s_{i1} = 100,\ s_{i2} = 105, \ldots, s_{i41} = 300\}. \quad (15)$$

The outcome or utility function u_i is described by the linearized version of the multiobjective optimization problem, that is, by using (8):

$$u_i = w_{syn,i}O_{syn,i}(s_1, \ldots, s_n) + w_{temp,i}O_{temp,i}(s_1, \ldots, s_n), \quad (16)$$

where $O_{syn,i}$ and $O_{temp,i}$ are, respectively, the objective functions of (10) and (14). $w_{syn,i}$ and $w_{temp,i}$ are the synchronization and temperature weights of the optimization for block i.

4.3.2. Distributed Algorithm

The method used to select the choice is defined as *unilateral maximization*: each player chooses the action maximizing its own utility function. The choices of other players are considered as given parameters. The maximization can be performed by running Algorithm 1 for each PE, where u_i is the utility function of PE i, $MyStgy$ is its last chosen strategy, and *OtherStgies* are the strategies chosen by other players in the previous cycle. Note that this code implements the utility maximization by comparing the outcomes of all possible solutions per player.

The code is embedded in each PE and is simultaneously executed at run time, allowing the parallel distributed optimization. The period between two executions is defined as *game cycle*. It is assumed that before the next game cycle, all PEs will end the current execution of the maximization process.

```
for all GameCycle do
    for all i do
        NewStgy ⇐ UnilaterallyMax(player-i)
    end for
    Stgy ⇐ NewStgy
end for
```

Algorithm 2: GameKernel.

In order to analyze the global behavior, the parallel launching of Algorithm 1 is simulated by Algorithm 2. All PEs launch the unilateral maximization at the same time and taking into account the last known choice of other players. Note that once a new application is mapped or the application load changed, the game-theoretic algorithm will adjust the PE frequencies.

5. Results

The results presented in this section include the characterization of the algorithm behavior. Two aspects are analyzed. From one side, since the objective of this work is to provide a *run-time* optimization mechanism, the dynamic response of the algorithm has major interest: the convergence speed of the system is studied. On the other side, in order to characterize the quality of the solution, it has to be compared with an existing optimization method.

In this section, these two aspects are analyzed from a statistical point of view. Firstly, the exploration methodology is discussed. Secondly, the convergence results are presented regarding the system scalability. Then, an exploration of the impact of the objective weights on the dynamic response is presented. This section ends with a comparison of the optimization quality of the game-theoretic solution with an offline method.

5.1. Methodology

The game complexity is defined by the number of PEs needed by the application (i.e., the number of players),

```
for Size = 4 to 100 do
    for Connectivity = 2 to Size −1 do
        for Scenario = 1 to 1000 do
            Generate random application
            Run reference optimization analysis
            Save results
        end for
    end for
end for
```

Algorithm 3: Statistical simulation script.

the application connectivity (i.e., the interaction between players), and the multiobjective function u_i (i.e., the utility function). In order to study the scalability of our technique, the application size is explored in a range of 4 to 100 processors. For each evaluated case, the maximum application connectivity is explored from 2 links per task to full connectivity ($n - 1$, where n is the number of tasks). The third aspect, the utility function, is explored by changing the objective weights w_{syn} between 0 and 1 and w_{temp} between 1 and 0.

A first analysis of the convergence speed is done by setting the objective weights to $w_{syn} = 1$ and $w_{temp} = 0$ and then repeating several different scenarios. In order to obtain statistical results, each parameter combination (i.e., size and application connectivity) is simulated 1000 times. For each time, a new application graph is defined. The simulation procedure is implemented by Algorithm 3.

In a second convergence analysis, the impact of the objective weights is explored. Several random 25-task applications are evaluated for each pair w_{syn}, w_{temp}.

5.2. Convergence

The dynamic response is analyzed by measuring the convergence speed in a number of game cycles that the algorithm takes to reach a solution. Statistically, the convergence speed corresponds to a gauss distribution with mean and standard deviation depending on the game parameters (number of processors and actions). For example, Figure 3 shows the distribution results over 98000 simulations of a 100-processor platform with random applications, connections chosen between 2 per PE to full connectivity. This curve indicates that typically a 100-processor MP-SoC will find a solution in around 18 game cycles but usually in less than 40 cycles, regardless of the application.

The analysis of the mean and the standard deviation of the convergence speed are repeated for the simulation results in a range of 4 to 100 processors. Figure 4 shows the results for different sizes. The graph shows 3 regions corresponding to the concentration of 68%, 95%, and 99.7%, respectively, the first, second, and third standard deviations. For instance, 99.7% of applications will converge in less than 32 game cycles in a 36-processor system. Note that the convergence speed decreases in $O(\log(n))$ when the platform

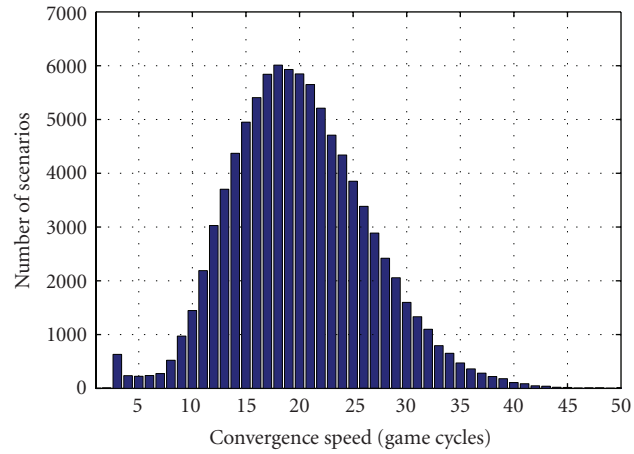

Figure 3: Convergence speed distribution for a 100-processor platform with $w_{syn} = 1$ and $w_{temp} = 0$: the average speed is 18 game cycles.

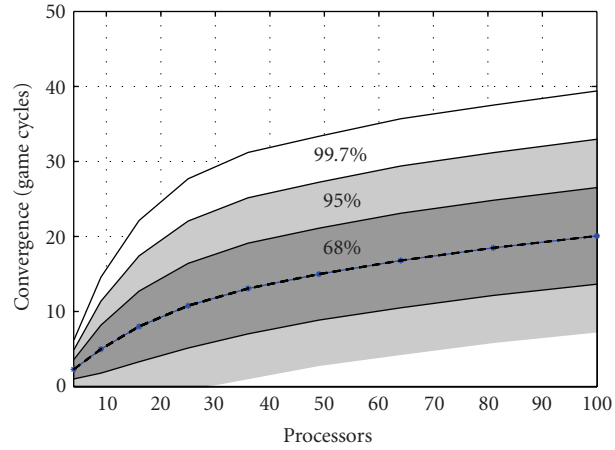

Figure 4: Convergence speed distribution from 4 to 100 processors with $w_{syn} = 1$ and $w_{temp} = 0$. The mean and standard deviations are indicated.

size augments, showing that the algorithm perfectly scales with the number of processors.

A game cycle is composed by two phases. The first one consists of the communication of the decisions of all PEs. The second phase is the maximization of the utility function. A generic 8051-microcontroller has been used to illustrate a game cycle duration. It takes around 400 clock cycles to process the algorithm or 800 nanoseconds considering a 500 MHz frequency. On the other side, the communication latency in a 2D mesh NoC is estimated as follows. The longest path in an n-processor system is $2\sqrt{n} - 1$ hops. It is assumed that there is no conflict in the interconnect. Considering that in the asynchronous NoC the equivalent node latency is around one cycle of the given 500 MHz clock, the maximum estimated communication delay is 38 nanoseconds per game cycle for a 100-processor MP-SoC. Table 1 summarizes the speed results measured in a number of game cycles and in the equivalent estimated times.

Table 1: Convergence speeds.

	Average*		Worst*	
Platform size	Game cycles	Time [μs]	Game cycles	Time [μs]
9 PEs	5	4.10	17	12.10
16 PEs	8	6.51	23	18.72
25 PEs	10	9.00	28	22.90
36 PEs	13	10.69	32	26.30
49 PEs	15	12.39	34	28.08
64 PEs	16	13.28	36	29.88
81 PEs	18	15.01	37	30.86
100 PEs	20	16.76	40	33.52

* From probabilistic analysis.

Table 2: Test case solutions.

	Configuration 1 ($w_{syn} = 1,\ w_{temp} = 0$)		Configuration 2 ($w_{syn} = 0.75,\ w_{temp} = 0.25$)		Configuration 3 ($w_{syn} = 0.5,\ w_{temp} = 0.5$)		Configuration 4 ($w_{syn} = 0.25,\ w_{temp} = 0.75$)	
PE	GT Algo. [MHz]	Matlab [MHz]	GT Algo. [MHz]	Matlab [MHz]	GT Algo. [MHz]	Matlab [MHz]	GT Algo. [MHz]	Matlab [MHz]
1	160	164	160	156	160	152	155	152
2	110	114	110	107	110	100	105	100
3	160	164	155	149	155	132	150	105
4	100	103	100	100	100	100	100	100
5	225	231	190	210	140	185	140	147
6	160	164	135	149	100	132	100	105
Conver.	17 game cycles	—	17 game cycles	—	4 game cycles	—	4 game cycles	—

From the statistical simulations, it is observed that 94% of evaluated scenarios converge to a solution. The other 6% cases do not converge to a unique solution but they present oscillations between two or more frequencies for each PE. It is assumed that in these conflictive cases an external mechanism such as task migration [32, 33] is used to solve the problem.

5.3. Impact of Weights

In order to study the impact of the objective weights on the convergence speed, 50 000 simulations are performed over a 5×5-processor array where each PE drives its frequency between 100 and 200 MHz with a step of 10 MHz. Applications are defined with random loads and connections as in the previous experience. For each simulation, a new application is randomly generated. The results are shown in Figure 5. The y-axis represents the number of scenarios found for each convergence speed from x-axis and for a given metric weight. The z-axis explores the synchronization weight w_{syn} from 0 to 1, corresponding to w_{temp} from 1 down to 0.

The results show that the convergence speed augments with w_{syn}, meaning that it depends on the metric complexity. For instance, for the trivial case of $w_{syn} = 0$, all the frequencies are driven to the minimum value in the first game cycle. It is due to the improvement of the temperature alone, without taking care of the task synchronization. On the other side, for $w_{syn} = 1$ the system presents the slowest convergence speed, with an average of 10 game cycles.

5.4. Optimization Quality

In the game-theoretic model, each player tries to optimize its outcome by maximizing the utility u_i. From a global point of view, the outcome is described by

$$U : \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \longmapsto \begin{bmatrix} u_1(s_1, s_2, \ldots, s_n) \\ u_2(s_1, s_2, \ldots, s_n) \\ \vdots \\ u_n(s_1, s_2, \ldots, s_n) \end{bmatrix}. \tag{17}$$

The global optimization problem is then formulated as

$$\max_{S} \left\{ \min_{u_i} \{U(S)\} \right\}, \quad s_{li} \le s_i \le s_{ui} \tag{18}$$

where s_{li} and s_{ui} are the lower and upper bounds of the strategy space of player i, that is, the minimum and maximum frequencies. This formulation is known as the *minimax problem* [34].

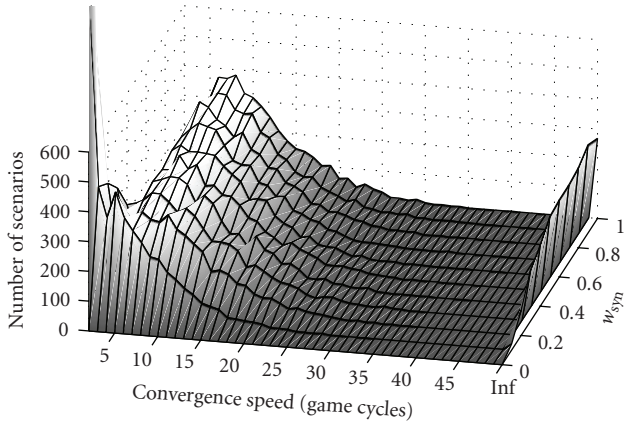

Figure 5: Weights exploration for a 5×5-processor array. The x-axis represents the number of scenarios over 50 000 simulations. The y-axis shows the convergence speed in game cycles, while the z-axis explores the weight of the synchronization metric w_{syn}. The inf label on the y-axis denotes those cases which do not converge.

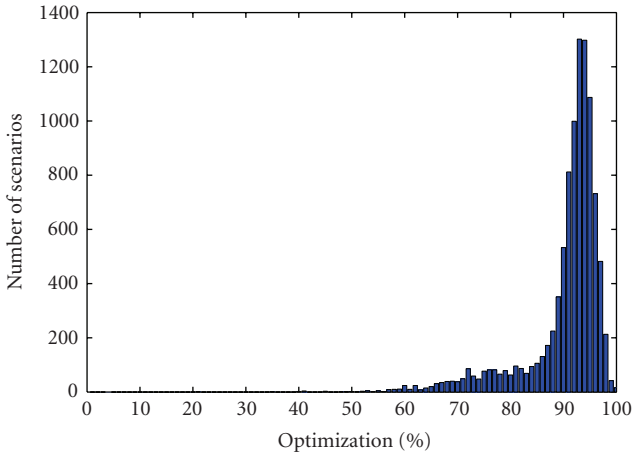

Figure 6: Optimization percentage distribution compared to Matlab minimax solution: an average of 89% and a peak of 93% of optimization are reached.

Using Matlab minimax solution, worst and best bounds are found for each simulated scenario. The game-theoretic solution is then positioned between these two references allowing the characterization of the quality of the found solution. A total of 10 000 simulations of the procedure used in Section 5.2 are analyzed, calculating the optimization percentage achieved in each case. The results are presented as a distribution curve in Figure 6. As it is shown, the distribution shows an average at 89% while it presents a peak at 93%. The results are concentrated between 58 and 98%. Note that these results are obtained in few game cycles, for instance, less than 40 for a 100-processor MP-SoC as was explained in Section 5.2. On the contrary, the Matlab minimax algorithm takes between some seconds and few minutes to calculate each solution on a nowadays desktop PC.

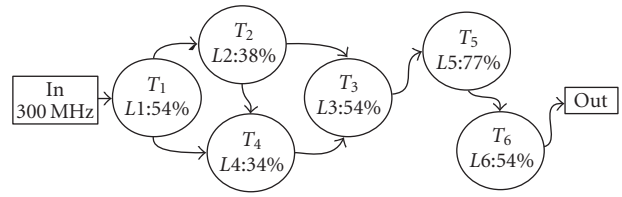

Figure 7: Application example composed of 6 tasks mapped on a 3 × 2-PE array.

6. Test Case

For clarity of the demonstration, a very simple test case composed of a 6-task application (Figure 7) mapped on a 3 × 2-PE array (Figure 2(c)) has been chosen. Each PE is able to adjust its own frequency between 100 and 300 MHz with a step of 5 MHz. The task synchronization is modeled as in Section 4.2.1 while the temperatures of PEs are calculated as in Section 4.2.2. The transfer thermal resistances are arbitrarily assumed to be $R_{ii} = 0.7$ and $R_{ij} = 0.3$ (the real values are dependent on the used technology). The two metrics are combined as in (16) by the weights w_{syn} for the task synchronism and w_{temp} for the PE temperature.

Four configurations were evaluated. The first one describes a system which is only interested in optimizing the task synchronization, that is, $w_{syn} = 1$ and $w_{temp} = 0$. In the context of this work, this configuration is used as the reference to calculate the temperature reduction achieved by the other configurations. The second one expresses a scenario where the synchronization represents 75% of the optimization importance; while only 25% is for the temperature minimization ($w_{syn} = 0.75$ and $w_{temp} = 0.25$). The third configuration defines a case with equal interest for each metric ($w_{syn} = 0.5$ and $w_{temp} = 0.5$), while the last case gives only 25% of importance for the synchronization ($w_{syn} = 0.25$ and $w_{temp} = 0.75$). For simplicity purpose, these values are arbitrary chosen to be the same for all PEs. Nevertheless, PEs may have different constraints. For example, a central PE may have more interest in temperature reduction than a border one to avoid hot-spots.

In order to measure the quality of the found solution, the same optimization issue is modeled as the minimax problem presented in Section 5.4 and solved using Matlab. The results are compared to the game-theoretic solution.

Figure 8 shows the evolution of the game-theoretic algorithm for the second configuration ($w_{syn} = 0.75$ and $w_{temp} = 0.25$). Each graph of the figure shows in solid lines the evolution of the chosen frequency. In this example, processor 1 takes only two game cycles to reach a stable solution, while processor 2 needs 3 cycles, and processor 3 needs 4 game cycles. Processor 4 has chosen the lowest frequency from the beginning; finally, processors 5 and 6 are the slowest that need 17 and 16 game cycles, respectively, to reach the solution. In this example, the game reaches the NE in 17 cycles. In dashed lines, Figure 8 shows the optimal solutions found with Matlab. After few game cycles, the game-theoretic solution converges to an NE close to the Matlab solution.

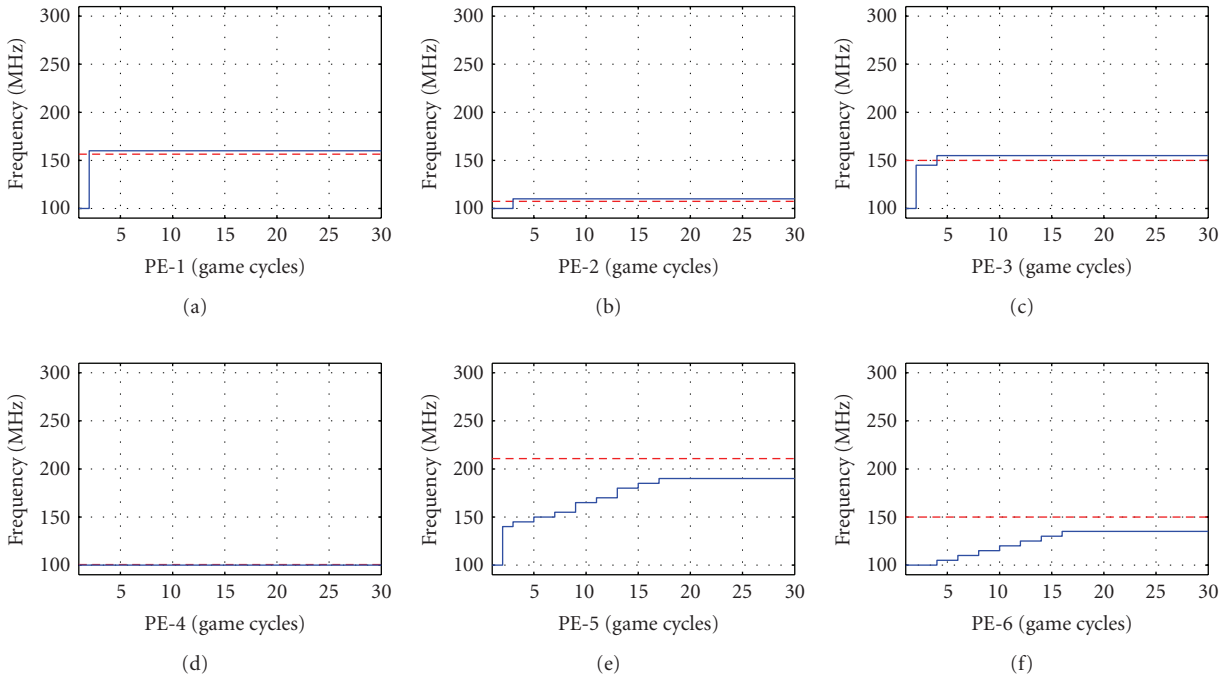

Figure 8: Dynamic response of a 3×2 processor array for a given 6-task application with $w_{syn} = 0.75$ and $w_{temp} = 0.25$. The solid line shows the evolution of the chosen frequency for each PE. The NE is defined by $f_1 = 160$, $f_2 = 110$, $f_3 = 155$, $f_4 = 100$, $f_5 = 190$, and $f_6 = 135$ MHz. The dashed line is the reference solution calculated by Matlab: $f_1 = 156$, $f_2 = 107$, $f_3 = 149$, $f_4 = 100$, $f_5 = 210$, and $f_6 = 149$ MHz.

Table 3: Test case temperature profiles.

	Configuration 1 ($w_{syn} = 1$, $w_{temp} = 0$)		Configuration 2 ($w_{syn} = 0.75$, $w_{temp} = 0.25$)		Configuration 3 ($w_{syn} = 0.5$, $w_{temp} = 0.5$)		Configuration 4 ($w_{syn} = 0.25$, $w_{temp} = 0.75$)	
PE	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab
1	35.0°C	35.9°C	35.0°C	34.2°C	35.0°C	33.2°C	34.0°C	33.2°C
2	48.1°C	49.5°C	45.7°C	45.8°C	42.7°C	42.1°C	41.4°C	38.2°C
3	38.6°C	39.6°C	36.4°C	36.2°C	34.3°C	32.4°C	33.3°C	7.0°C
4	37.1°C	38.1°C	35.0°C	35.9°C	32.0°C	34.2°C	31.7°C	31.9°C
5	53.7°C	55.2°C	47.3°C	50.7°C	38.2°C	45.8°C	37.9°C	38.8°C
6	45.5°C	46.6°C	39.6°C	42.4°C	31.7°C	37.5°C	31.4°C	29.8°C
T. Avg	43.0°C	44.1°C	39.8°C	40.9°C	35.6°C	37.5°C	34.9°C	33.1°C
T. Gain	—	—	7.95%	7.99%	23.94%	17.63%	23.02%	33.11%

Table 4: Test case synchronization profiles.

	Configuration 1 ($w_{syn} = 1$, $w_{temp} = 0$)		Configuration 2 ($w_{syn} = 0.75$, $w_{temp} = 0.25$)		Configuration 3 ($w_{syn} = 0.5$, $w_{temp} = 0.5$)		Configuration 4 ($w_{syn} = 0.25$, $w_{temp} = 0.75$)	
PE	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab
1	97.14%	98.16%	97.14%	94.68%	97.14%	88.86%	94.85%	88.86%
2	95.11%	97.23%	96.28%	93.18%	96.28%	81.81%	94.87%	68.43%
3	97.30%	98.31%	89.72%	94.42%	76.32%	85.02%	75.49%	64.51%
4	98.19%	99.10%	97.21%	92.77%	97.21%	81.26%	93.30%	71.21%
5	96.30%	96.65%	80.33%	97.11%	50.92%	96.22%	55.11%	96.80%
6	96.30%	96.65%	97.01%	97.11%	96.96%	96.22%	96.96%	96.80%
Avg	96.72%	97.69%	92.96%	94.88%	85.80%	88.23%	85.10%	81.10%

Tables 2, 3, and 4 summarize the results of the four evaluated configurations. Table 2 lists the frequencies found by the game-theoretic algorithm and by Matlab. Note that in all cases, the solutions found by the game-theoretic algorithm are close to those found with Matlab. The convergence speed of the game-theoretic solution for each configuration is also highlighted. Table 3 presents the resulting temperature of each PE, the average temperature of the entire system, and the gain achieved by configurations 2, 3, and 4 compared to configuration 1. These results show up to 23% of average temperature gain, depending on w_{temp} value. Note that these reductions are obtained in few game cycles, making this technique able to manage the parameters at run time.

In addition, Table 4 lists the improvement percentage of task synchronization. The task synchronization of each PE has been calculated by using expression (10) for a nominal case where all PEs work at 200 MHz. The synchronization improvements for the game-theoretic and Matlab optimizations are calculated for each configuration with respect to the nominal case. The results are listed in Table 4, showing for configuration 1 ($w_{syn} = 1$, $w_{temp} = 0$) an average improvement of 96.72% for the game-theoretic optimization and 97.69% for the Matlab one compared to the nominal frequency set. Note that for configurations 1, 2, and 3 Matab obtains better synchronizations at higher temperatures than the game-theoretic algorithm. On the contrary, for configuration 4, Matlab obtains worse synchronizations at lower temperatures compared to our algorithm.

Finally, the maximum, average, and minimum PE temperatures for the game-theoretic algorithm are represented in Figure 9. The results show more uniform temperature distribution when w_{temp} rises. Configuration 1 presents 19°C of difference between maximum and minimum (44% of the average), while configuration 4 only shows 10°C (28% of the average temperature). The run-time game-theoretic method has not only reduced the average temperature but also the peaks or hot spots.

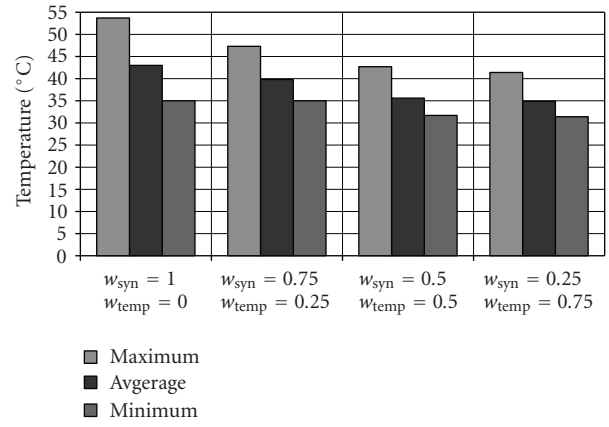

Figure 9: Maximum (Max), average (Avg), and minimum (Min) temperatures for the game-theoretic solution for 4 weight configurations.

7. Conclusion

In this paper, we have presented a novel run-time technique based on the game theory. We have discussed the optimization of multiple objectives on embedded reconfigurable systems. We have proposed an algorithm that optimizes the temperature profile while maintaining the task synchronization. Compared to other approaches, our technique assumes a complete distributed multiprocessor system able to take decisions at run time.

The results have shown that our method scales with the number of processor without excessive convergence times. For a 100-processor platform, our technique has required an average of 20 calculation cycles to reach the solution, that is, about 16 μs when using the 8051 microcontroller at 500 MHz. The few calculation cycles needed to converge make this technique able to optimize metrics at run time on massively parallel embedded systems.

We have measured that the achieved optimization is about 89% in average compared to a global offline method. The evaluated test case has showed that our algorithm can achieve reductions of up to 23% in the temperature profile.

References

[1] G. Martin, "Overview of the MPSoC design challenge," in *Proceedings of the 43rd Annual Conference on Design Automation (DAC '06)*, pp. 274–279, ACM, San Francisco, Calif, USA, July 2006.

[2] http://techresearch.intel.com/articles/Tera-Scale/1421.htm.

[3] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Annual Conference on Design Automation (DAC '01)*, pp. 684–689, Las Vegas, Nev, USA, June 2001.

[4] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[5] J. L. Cohon, *Multiobjective Programming and Planning*, Academic Press, New York, NY, USA, 1978.

[6] G. M. Link and N. Vijaykrishnan, "Hotspot prevention through runtime reconfiguration in network-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*, vol. 1, pp. 648–649, IEEE Computer Society, Munich, Germany, March 2005.

[7] M. Ruggiero, A. Acquaviva, D. Bertozzi, and L. Benini, "Application-specific power-aware workload allocation for voltage scalable MPSoC platforms," in *Proceedings of the IEEE International Conference on Computer Design (ICCD '05)*, pp. 87–93, IEEE Computer Society, San Jose, Calif, USA, October 2005.

[8] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, NJ, USA, 1944.

[9] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, "Temperature-aware distributed run-time optimization on MP-SoC using game theory," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '08)*, pp. 375–380, IEEE Computer Society, Montpellier, France, April 2008.

[10] J. Donald and M. Martonosi, "Techniques for multicore thermal management: classification and new exploration," in *Proceedings of the 33rd International Symposium on Computer*

Architecture (ISCA '06), pp. 78–88, Boston, Mass, USA, June 2006.

[11] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, "Voltage-frequency island partitioning for GALS-based networks-on-chip," in *Proceedings of the 44th Annual Conference on Design Automation (DAC '07)*, pp. 110–115, ACM, San Diego, Calif, USA, June 2007.

[12] E. Beigné, F. Clermidy, S. Miermont, and P. Vivet, "Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC," in *Proceedings of the 2nd ACM/IEEE International Symposium on Networks-on-Chip (NOCS '08)*, pp. 129–138, Newcastle upon Tyne, UK, April 2008.

[13] E. Beigné, F. Clermidy, S. Miermont, et al., "A fully integrated power supply unit for fine grain dvfs and leakage control validated on low-voltage srams," in *Proceeding of the 34th European Solid-State Circuits Conference (ESSCIRC '08)*, Edinburg, UK, September 2008.

[14] D. Lattard, E. Beigné, C. Bernard, et al., "A telecom baseband circuit based on an asynchronous network-on-chip," in *Proceedings of the 54th IEEE International Solid-State Circuits Conference (ISSCC '07)*, pp. 258–601, San Francisco, Calif, USA, February 2007.

[15] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '07)*, pp. 111–116, ACM, Salzburg, Austria, September-October 2007.

[16] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 1, pp. 3–8, Munich, Germany, March 2006.

[17] M. Kandemir and G. Chen, "Locality-aware process scheduling for embedded MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*, vol. 2, pp. 870–875, IEEE Computer Society, Munich, Germany, March 2005.

[18] F. Li and M. Kandemir, "Locality-conscious workload assignment for array-based computations in MPSOC architectures," in *Proceedings of the 42nd Annual Conference on Design Automation (DAC '05)*, pp. 95–100, ACM, Anaheim, Calif, USA, June 2005.

[19] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '04)*, vol. 1, pp. 234–239, IEEE Computer Society, Paris, France, February 2004.

[20] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," in *Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP '07)*, pp. 34–40, IEEE Computer Society, Porto alegre, Brazil, May 2007.

[21] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*, pp. 1659–1664, EDA Consortium, Nice, France, April 2007.

[22] Ch. Ykman-Couvreur, E. Brockmeyer, V. Nollet, T. Marescaux, F. Catthoor, and H. Corporaal, "Design-time application exploration for MP-SoC customized run-time management," in *Proceedings of the International Symposium on System-on-Chip (SoC '05)*, pp. 66–69, Tampere, Finland, November 2005.

[23] Ch. Ykman-Couvreur, V. Nollet, T. Marescaux, E. Brockmeyer, F. Catthoor, and H. Corporaal, "Pareto-based application specification for MP-SoC customized run-time management," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS '06)*, pp. 78–84, Samos, Greece, July 2006.

[24] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal, "Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management," in *Proceedings of the International Symposium on System-on-Chip (SOC '06)*, pp. 195–198, Tampere, Finland, November 2006.

[25] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, vol. 35, pp. 89–102, ACM, Banff, Canada, December 2001.

[26] B. Knerr, M. Holzer, and M. Rupp, "Task scheduling for power optimisation of multi frequency synchronous data flow graphs," in *Proceedings of the 18th Symposium on Integrated Circuits and Systems Design (SBCCI '05)*, pp. 50–55, ACM, Florianopolis, Brazil, September 2005.

[27] N. Hanchate and N. Ranganathan, "Simultaneous interconnect delay and crosstalk noise optimization through gate sizing using game theory," *IEEE Transactions on Computers*, vol. 55, no. 8, pp. 1011–1023, 2006.

[28] J. Nash, "Non-cooperative games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[29] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*, MIT Press, Cambridge Mass, USA, 1994.

[30] K. Niyogi and D. Marculescu, "Speed and voltage selection for GALS systems based on voltage/frequency islands," in *Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC '05)*, pp. 292–297, ACM, Shanghai, China, January 2005.

[31] W. Hung, C. Addo-Ouaye, T. Theocharides, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Thermal-aware IP virtualization and placement for networks-on-chip architecture," in *Proceedings of the IEEE International Conference on Computer Design (ICCD '04)*, pp. 430–437, IEEE Computer Society, San Jose, Calif, USA, October 2004.

[32] N. Saint-Jean, P. Benoit, G. Sassatelli, L. Torres, and M. Robert, "Application case studies on HS-scale, a MP-SOC for embbeded systems," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS '07)*, pp. 88–95, Samos, Greece, July 2007.

[33] N. Saint-Jean, G. Sassatelli, P. Benoit, L. Torres, and M. Robert, "HS-scale: a hardware-software scalable MP-SOC architecture for embedded systems," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*, pp. 21–28, IEEE Computer Society, Porto Alegre, Brazil, March 2007.

[34] J. W. Herrmann, "A genetic algorithm for minimax optimization problems," in *Proceedings of the Congress on Evolutionary Computation (CEC '99)*, vol. 2, pp. 1099–1103, IEEE Press, Washington, DC, USA, July 1999.