ICT-2009.3.2-248603-IP

Modeling, Control and Management of Thermal Effects in Circuits of the Future

	WP no.	Deliverable no.	Lead participant
	WP3	**D3.4.1.**	**CEA-LETI**

Report on distributed algorithm for thermal and energy optimization in a MP-SoC

Prepared by	**Alexandre VALENTIAN (CEA-LETI)**
Issued by	**THERMINATOR WP3 Leader**
Document Number	**THERMINATOR/D3.4.1/v1**
Dissemination Level	**Public**
Date	**28/01/2011**

Document

Title	Report on distributed algorithm for thermal and energy optimization in a MP-SoC
Type	Report
Ref	D3.4.1
Target version	V1
Current issue	V1
Status	Released
File	D3.4.1.doc
Author(s)	Alexandre VALENTIAN (CEA-LETI) Imen MANSOURI (CEA-LETI)
Reviewer(s)	PCT
Approver(s)	PCT
Approval date	
Release date	28/01/2011

Distribution of the release

Dissemination level	PU
Distribution list	

History

Rev.	DATE	Comment
0.1	15-12-2010	Initial version
1.0	28-01-2011	Final version

This page was intentionally left blank.

Contents

1 Introduction

CEA-LETI and CSEM are currently designing a Multi-Processor System-on-a-Chip (MP-SoC) platform, named Genepy. Genepy is composed of 6 clusters called SMEP, each SMEP containing 3 DSP-centered processing elements, interconnected by an asynchronous Network-On-Chip (NoC) – see Figure 1. Each cluster will have the possibility to work at its own clock frequency, i.e. it is an independent frequency island. Exchange of data between clusters is made possible thanks to the asynchronous NoC, which possesses synchronization interfaces with the synchronous world.
Genepy will be fitted with temperature sensors and activity monitors and thus be an ideal platform for assessing the efficiency of dynamic thermal management solutions devised in the frame of the Therminator project. With the appropriate sensors and actuators on hand, what is missing is the conductor to enable run-time thermal and power management. This conductor is realized thanks to a distributed algorithm inspired by game theory, which will determine the workload of the different clusters.
The optimization process is used to set the appropriate frequency of cores each time the application changes. This process has to meet some constraints. First of all, it operates intrusively on the fly to tackle changes in the system. Besides, fast response, low complexity and stability are required to have an efficient and reliable control scheme. The complexity of the optimization process has to scale with the number of cores.

In this report, we present a distributed technique inspired by Game Theory (GT) to solve this optimization issue, which is detailed in Section 2. In order to show the pertinence of the approach, we compare this solution to a centralized one based on state-of-the-art Lagrangian method in Section 3. A telecom test-case application is used to compute the efficiency of both techniques.

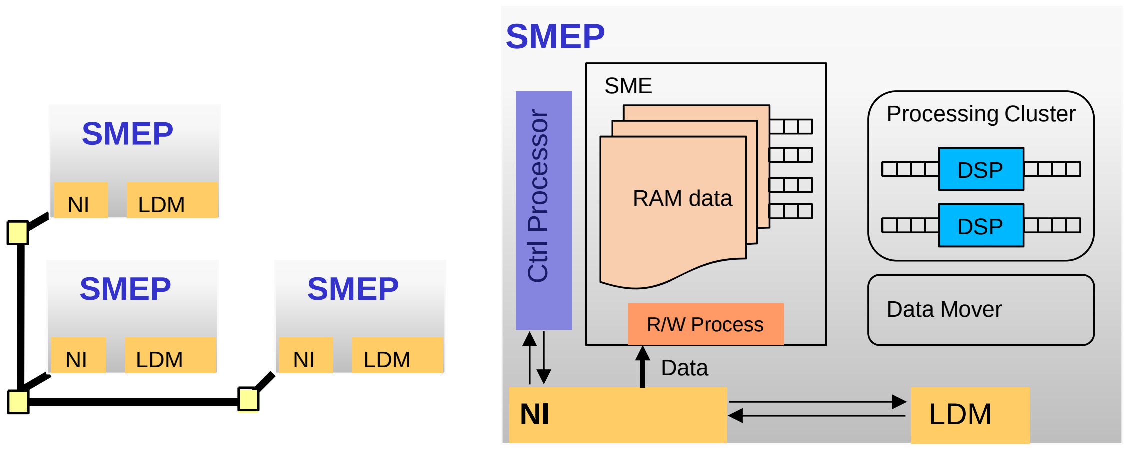

Figure 1. GENEPY and SMEP units

2 A Game Theoretic-Approach For MP-SoC Energy Management

Game theory (GT) is usually considered as a branch of applied mathematics. It aims at modeling the interactions between rational agents or decision makers. A game is basically a scenario composed of rational players, each one having a set of possible actions. Regarding the whole system including interactions, these actions produce consequences quantified as outcomes. In such scenario, each player makes strategic choices over the set of possible actions to execute the best one, pursuing some predefined objective in terms of outcome. The individual success of each player (the quantitative outcome) depends not only on its own choice but also on the actions chosen by other players in the game.
In this section, we present some basic definitions of GT useful to understand our approach. Then, the modeling approach is discussed to finally present the game-based optimization scheme.

2.1 Game-Theoretic Models

A non-cooperative game is a scenario with several players interacting by actions and consequences. Basically, the players individually choose an action within a defined set, resulting in consequences or outcomes. Each player tries to maximize its outcome according to its preferences. If this sequence is repeated, under certain conditions, the game will find an equilibrium solution with quasi-optimal outcomes.

Mathematically, non-cooperative games are represented in normal-form as follows:

$$\Gamma = \langle N, S_i, u_i \rangle, \forall i \in N , \qquad (1)$$

where $N = \{1, \ldots, n\}$ is the set of n players, S_i is the set of actions for player i and u_i is a function describing its outcome.

The discrete set of actions for player i is defined as:

$$S_i = \{s_{i1}, s_{i2}, \ldots, s_{im}\}, \qquad (2)$$

where *mi* is the number of possible actions for this player. Note that *mi* is not necessarily equal to *mk* for $i \neq k$ and $i, k \in N$, i.e. players can have different numbers, and thus heterogeneous actions.

The outcome of player i is represented by the cost function u_i coding its quantified outcome. Because of interactions with other players, this outcome is a function of the choices of current player as well as other players. Mathematically, it is represented by:

$$u_i : S_1 \times S_2 \times \ldots \times S_n \to \Re , \qquad (3)$$

The notion of solution in a game is the way to describe how it should be played according to the preferences of each player. These descriptions specify which strategic set of actions will be played, thus predicting the resulting outcomes of the game. A powerful definition is the Nash equilibrium: it proposes an equilibrium point as a possible solution of a game. Nash proved that each n-player non-cooperative game has at least one equilibrium point, known as Nash Equilibrium (NE) [1]. It can be defined by pure strategies or by mixed strategies. In the first case, players choose and play only one action among the possible set. On the contrary, in mixed strategies, the solution is chosen in a set of actions, each played with a given probability.
For pure strategies, a NE is defined as:

Definition: *For a given non-cooperative strategic game* $\Gamma = \langle N, S_i, u_i \rangle, \forall i \in N$ *with n players, a solution:*

$$S^* = \{s_1^* \in S_1, s_2^* \in S_2, \dots s_n^* \in S_n\}, \qquad (4)$$

is a Nash Equilibrium if:

$$u_i\{s_1^*, \dots, s_i^*, \dots, s_n^*\} \geq u_i\{s_1^*, \dots, s_i, \dots, s_n^*\}, \qquad (5)$$

$\forall i \in N, \forall s_i \in S_i$, *being* s_i^* *the Nash Equilibrium strategy for player i.*

An interesting feature is when a NE is reached, players cannot improve their outcomes by unilaterally changing their strategies. This indicates that if a NE is reached, there will be no deviation from this solution without cooperation, i.e. without making coalitions: a player in this situation cannot improve its own outcome, given the actions of the other players. It is at least a local maximum and it ensures the stability of the method.

2.2 Energy consumption model

The system under consideration is an MPSoC composed of *n* tiles interconnected by an asynchronous NoC. For the sake of generality, we consider that each Processing Element (PE) can adapt its supply voltage value, in addition to its clock frequency, thanks to a Dynamic Voltage and Frequency Scaling (DVFS) engine – see Figure 2. We are interested in estimating the amount of energy needed to feed an operating tile during a given period of time. This magnitude must be optimized under certain constraints, for instance maintaining real-time performances in terms of calculation latency.
The DVFS device allows choosing among a finite number of clock periods, each one associated to the best corresponding voltage. We denote T_i the clock period corresponding to the frequency of PE_i. The energy consumption E_i of the whole tile *i* is considered during period T0. For a CMOS circuit, it is given by the sum of static and dynamic consumptions, for both memories and logic circuitries:

$$E_i(T_i) = E_{dyn,i} + E_{stat,i}, \qquad (6)$$

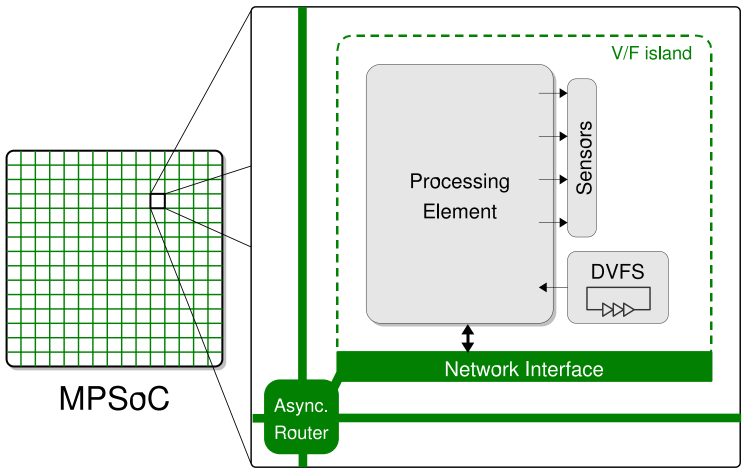

Figure 2. Multi-Processor System-on-Chip

The static power dissipation $P_{stat,i}$ is considered constant. In a more complex model, this term can be defined as a function of different variables of the circuit such as temperature, supply voltage, etc. The static energy needed to feed the tile during a given time T_0 is:

$$E_{stat} = P_{stat,i} T_0 , \qquad (7)$$

The dynamic consumption of tile *i* is a function of its activity and is modeled as follows. The task assigned to *PE*$_i$ is processed in *N*$_i$ clock periods, or in $N_i T_i$ seconds. During this period, tile *i* has a high activity, needing a dynamic energy equal to $E_{Hdyn,i}$. During the rest of the time until the task is launched again, the PE is in low-activity state, due to classical low-power techniques such as gated clocks. Thus, during $(T_0 - N_i T_i)$ seconds, it needs $E_{Ldyn,i}$. This energy distribution over time is shown in Figure 3. Note that typically $E_{Ldyn,i} < E_{Hdyn,i}$. Thus, the total dynamic energy required by tile *i* becomes:

$$E_{dyn,i} = E_{Hdyn,i} + E_{Ldyn,i} , \qquad (8)$$

We define $(1-\lambda_i)$ the power reduction of tile *i* when it is using low-power mechanisms (gated clocks or similar):

$$\lambda_i = \frac{E_{Ldyn,i} / (T_0 - N_i T_i)}{E_{Ldyn,i} / N_i T_i} , \qquad (9)$$

Considering the voltage supply V_{dd}, the clock period T and the average gate switching activity α, the dynamic power consumption is represented by $P_{dyn} = \alpha V_{dd}{}^2 / T$.

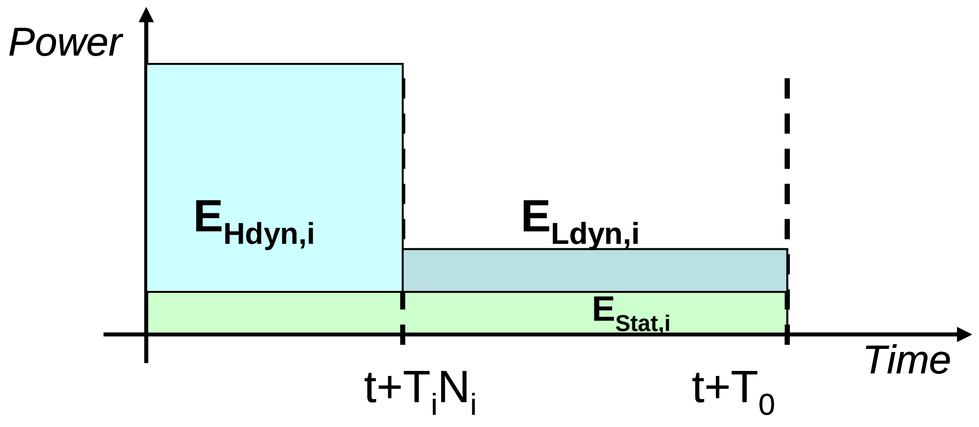

Figure 3. Energy profile of PE-*i*

Thus, the dynamic energy needed to feed the block during a nominal period T_{nom} at a nominal voltage supply V_{nom} is:

$$E_{nom} = P_{nom}T_{nom} = \alpha V_{nom}{}^2, \qquad (10)$$

Since α remains constant, for a different voltage supply V_{dd}, we obtain $\alpha = E_{nom} / V_{nom}{}^2 = E_{dd} / V_{dd}{}^2$. Thus, the energy after applying DVFS becomes $E_{dd} = E_{nom}(V_{dd}/V_{nom})^2$. The circuit frequency is proportional to the supply voltage, i.e. $V_{nom}T_{nom} = V_{dd}T_{dd}$. Thus, for a given clock period T_{dd}, the corresponding dynamic energy E_{dd} is:

$$E_{dd} = E_{nom}(T_{nom}/T_{dd})^2, \qquad (11)$$

Using expression (11) in equation (8), we obtain the dynamic energy of tile *i* for a given clock period T_i:

$$E_{dyn,i} = \left[N_i + \lambda_i\left(\frac{T_0}{T_i} - N_i\right)\right]\frac{E_{nom,i}T_{nom,i}{}^2}{T_i^2}, \qquad (12)$$

Finally, replacing expressions (7) and (12) in (6), the energy required by tile *i* when applying T_i becomes:

$$E_i(T_i) = \left[N_i + \lambda_i\left(\frac{T_0}{T_i} - N_i\right)\right]\frac{E_{nom,i}T_{nom,i}{}^2}{T_i^2} + T_0P_{stat,i}, \qquad (13)$$

Note that $E_{nom,i}$ should be measured and corresponds to the energy consumed when clocking at $T_{nom,i}$. Parameter λ_i should also be estimated considering the used technology and low-power techniques. Finally, parameters N_i and T_0 are the number of clock periods needed to process the task and the total processing time of the given application.

2.3 Optimization scheme for MPSoC energy management

In section 2.1, the basic definitions of game theory were presented and in section 2.2, a model was developed to describe the energy consumption of multi-core tiles of a parallel embedded system with fine-grain voltage and frequency controls. These two elements must now be combined to provide an optimization scheme for MPSoC energy management under calculation latency constraint. To do this, we will place an MPSoC in a game-theoretic context by modeling the PEs as players in a non-cooperative game. The latency and power consumption combination represents the local objective function which depends on other PEs global state.

As stated above, each PE integrates a DVFS engine able to set a finite number of different frequencies. We denote T_i the clock period corresponding to the frequency of *PE_i* and T_{i-} as the frequency vector of all other PEs in the NoC:

$$T_{i-} = [T_1, T_2, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n], \qquad (14)$$

For the sake of explanation, we assume the MPSoC system running one application composed of a set of communicating tasks, each PE executing only one task. The objective is to minimize the global energy consumption of the whole system under application latency constraints.

The problem can be stated as follows:

$$\begin{aligned} &\text{minimize } \sum_i U_i(T_i, T_{i-}), \forall i \in N \\ &\text{where } U_i(T_i, T_{i-}) = -(EC_i + LC_i + L_{pen,i}) \end{aligned} \qquad (15)$$

where U_i represents the local utility function computed by PE_i at the operating periods (T_i, T_{i-}), defined as the sum of three normalized terms characterizing the state of PE_i: EC_i, LC_i and $L_{pen,i}$. EC_i is the energy contribution of PE_i versus the whole system consumption, LC_i is the local latency impact weighted by the global latency constraint, and $L_{pen,i}$ is a penalty function expressing latency constraint violations in real-time system.

The energy model developed in the previous section describes the energy of each tile independently from the other ones. For example, while tile 1 consumes energy E1, tile 2 consumes E2 independently of the first one for a given calculation latency. Nevertheless, when optimizing an MPSoC, we are interested not only in the trade-off between these quantities (energy and latency) but also among different tiles. In other words, we are interested in optimizing E1 together with E2. Also, the energy model developed before should be modified for describing the interactions between quantities among all players in the system.

Our idea is to combine, in the tile equation, the quantity (energy or latency) of this tile with the total sum of the quantity available in the system. We proceed through a comparison of the contribution of a given quantity to the whole system. Mathematically, the energy contribution of PE_i to the total energy can be deduced:

$$EC_i(T_i, T_{i-}) = \frac{\rho_i E_i(T_i)}{\sum_k E_k(T_k)}, \qquad (16)$$

where $\rho_i = N_i / \sum_k N_k$ is the weight of PE_i in terms of number of clock cycles required over the total application requirements. Note that EC_i is a function of the clock periods of all the tiles in the system. In the same way, the latency contribution of PE_i to the total latency of the critical path of an application can be calculated as:

$$LC_i(T_i, T_{i-}) = \frac{\rho_i L_i(T_i)}{\sum_k L_k(T_k)}, \qquad (17)$$

Once again, ρ_i is the weight of PE_{-i} in terms of number of clock cycles required over the total application requirements, and LC_i depends on the clock periods of all tiles in the system. $\sum_k L_k(T_k)$ is the total latency of the application considering the critical path. The latency L_i is given by the product between the clock period T_i in seconds and the number of cycles N_i that PE_{-i} needs to process the assigned task.
The system is running a given application represented by a set of tasks introducing a computational latency L_i. The whole latency should not exceed L_{Max} for real-time purpose. We consider the critical path of the applications, meaning that the total latency is the sum of all local latencies. $L_{pen,i}$ is introduced to keep the total latency below L_{Max}:

$$L_{pen,i}(T_i, T_{i-}) = \begin{cases} 0 & \text{if } \sum_k L_k(T_k) \le L_{Max} \\ a(T_i - T_{ml}) & \text{otherwise} \end{cases}, \qquad (18)$$

Here, *a* represents the penalty constant and T_{ml} is the largest clock period for PE$_i$ needed to satisfy latency constraint.

2.4 Distributed algorithm

As explained in the previous section, we model the PEs as reactive players holding the utility function U_i (Eq. 15) as its game objective. At game cycle k, PE_i starts to calculate U_i for every possible T_i, between T_{min} and T_{max} with a granularity ΔT; then, the best solution over the whole discrete range is pointed as the effective strategy for the next game cycle. This solution noted as $T_{i,k}^*$ is then applied using the DVFS engine.

In order to reduce the complexity of the algorithm implementation, we limit this search to only three values of possible T_i. We compare U_i considering $T_{i,k-1}^*$ and its two adjacent periods $\{ T_{i,k-1}^* \pm \Delta T \}$. Each PE executes this algorithm taking into account the state of the whole system resulted from the previous game. In other words, to process $U_i(k)$, the algorithm needs global information about latency and energy consumption obtained from the previous game decisions. In accordance with game theory models, we call this optimization process the Local Decision Maker (LDM).

Local Decision Maker process

1. Scan the network and get outcomes of previous cycle: L_{tot} (k-1) and E_{tot} (k-1)
2. Compute $\{U_i\ (T_{i,k-1}^*\ )\ ,\ U_i\ (T_{i,k-1}^*\ \pm \Delta T)\}$ according to Eq. (15)
3. Compare the resulting values and find the new solution $T_{i,k}^*$

$$T_{i,k}^* = \arg\min \left\{ U_i \left(T_{i,k-1}^* \right), U_i \left(T_{i,k-1}^* \pm \Delta T \right) \right\} \qquad (19)$$

4. Stop the algorithm when a steady-state is reached

3 Distributed Vs Centralized Energy Management

In order to prove the efficiency of the proposed approach, it is necessary to compare it to a centralized optimization scheme. In this section, we first introduce a description of the centralized algorithm developed to solve the energy minimization problem detailed in Section 2. The centralized method used as a reference for the comparison is based on a standard gradient-type iterative strategy, commonly used to solve optimization problems in many engineering fields. It implements the « augmented Lagrangian » [2]. Then, we compare both approaches on the same test-benches in terms of optimum quality and convergence speed for different number of PEs.

3.1 Reference centralized approach: Augmented Lagrangian Method

The Lagrangian technique permits maintaining application constraints during the energy minimization iterative process. The Lagrangian technique reformulates the constrained optimization into an unconstrained problem. The function to optimize includes both the energy consumption and a quadratic penalty term associated to the performance constraint. The penalization is expressed by the factor λ, known as Lagrangian factor. For normalization purposes, we divide each part of this function with their initial values: E_{ini} and T_{ini} denote respectively the whole system consumption and latency measured in an initial state. In the following, the vector T denotes the clock periods associated to a set of n processors.
The new objective function becomes:

$$\text{minimize } \frac{\sum_i E_i(T_i)}{E_{ini}} + \lambda\left(\frac{\sum_i N_i T_i - T_0}{T_{ini}}\right)^2, \forall i \in [1:n] \quad (20)$$

where n is the size of the system.

The resolution of such a system is strongly dependent on the value of the Lagrangian factor λ. As a consequence, choosing a small λ leads to a solution that does not fulfill constraints while, on the contrary, choosing a significantly large value of λ prevents the algorithm to find the optimum. The augmented Lagrangian approach was introduced to avoid this conditioning problem and to reach the exact solution independently of this factor. The Lagrange factor λ is incremented during the optimization process to maintain the problem constraints when the program is getting closer to the optimum. A second factor, noted as δ, is introduced to calculate λ increments between successive iterations.
The system (20), adapted to the augmented Lagrangian formulation, can be written as:

$$\text{minimize } \frac{\sum_i E_i(T_i)}{E_{ini}} + \delta\left(\frac{\lambda}{2\delta} + \frac{\sum_i N_i T_i - T_0}{T_{ini}}\right)^2, \forall i \in [1:n] \quad (21)$$

$$\text{where } \lambda = \max\left\{\lambda + 2\delta\left(\sum_i N_i T_i - T_0\right), 0\right\}$$

In such a formulation, the solution of the constrained problem is replaced by the solution of a sequence of quasi-unconstrained minimizations. A novel optimization problem is set each time the factor λ is updated. The augmented Lagrangian algorithm is described below, the index *k* denoting the iteration number. Initially, we set the system with its best performances: T1 corresponds to the rapid clock frequency of each processor; λ and δ are fixed to 0 and 10 respectively; the constant *c*, used to increase δ after each iteration, is arbitrary chosen in the interval [0:10].

Outline of the optimization solver

1. solve the quasi-unconstrained problem :

$$\text{minimize } \frac{\sum_i E_i(T_i)}{E_{ini}} + \delta_{k-1}\left(\frac{\lambda_{k-1}}{2\delta_{k-1}} + \frac{\sum_i N_i T_i - T_0}{T_{ini}}\right)^2, \forall i \in [1:n] \qquad (22)$$

   w*i*th $T_{i,k-1}$ as an initial solution.

2. Set T_k as the solution of the sub-system (22)
3. Check if the constraint is sufficiently satisfied; if true then the iteration process terminates, and T_k is the solution; else increase δ and pass to step 4.

$$\delta_k = \delta_{k-1} \times c \qquad (23)$$

4. Update λ for the next cycle according to formula (24), and then repeat the process from step (1):

$$\lambda_k = \max\left\{\lambda_{k-1} + 2\delta_k\left(\sum_i N_i T_{i,k} - T_0\right), 0\right\} \qquad (24)$$

The search direction of the optimum in step (1) is not restricted by the constraints. Thus, the search space is better explored and the optimization quality is guaranteed. To find the solution for each newly defined sub-problem (in step (1)), we chose to use the steepest descent method. Compared to other gradient techniques, the steepest method results in faster convergence and can operate with reasonable computing features. For example, simple descent gradient is ineffective unless the step size used is very small and it is then very slow. The steepest descent performs far better than the simple scheme by employing second-order information derived from the Hessian matrix of the objective function (Hessian matrix is the second partial derivatives of Eq.22 in terms of T). On the other hand, conjugate gradient methods converge very quickly and are independent of the problem scaling, but from a computational aspect, they cannot be used in our context. Conjugate gradient methods require the computation of the inverse of Hessian matrix in each iteration, which is clearly time and resource consuming. The reader can refer to [2] for more detailed information on numerical optimization methods.
Before launching the steepest iterates, all the function parameters as well as the vector T are

set by the Lagrangian algorithm. Then the following procedure is performed to solve the sub-problem in step (1). This process is repeated as long as the maximum frequency variation among the processor is less than the DVFS resolution RDVFS. Herein, *j* denotes the number of iteration inside this loop, and < , > is the Cartesian product in IRn space.

Outline of the sub-problem solver

1. Evaluate the gradient vector G_j at T_j and the Hessian matrix H_j
2. Compute the search direction α_j according to the formula:

$$\alpha_j = \frac{\langle G_j , G_j \rangle}{\langle H_j G_j , G_j \rangle} \qquad (25)$$

3. Update the next period vector in the direction of the gradient via the rule

$$T_{j+1} = T_j - \alpha_j \times G_j \qquad (26)$$

4. Project the result on the permissible ranges : [T_{min}, T_{max}]
5. Check if max ($1/T_j$) < R_{DVFS} , then finish, otherwise go back to step 1.

3.2 Convergence speed and optimization quality of distributed vs. centralized approaches

We analyze the performances of both centralized and the game theoretic-based approaches, as the problem size increases using a statistical method. In order to have a fair comparison, we analyze the properties of each algorithm using the same initial configuration (default starting point) and we use the same stopping criteria. In our experiments, we generate different application graphs sized from 10 to 120 mapped to a 120-PE platform. For simplicity purposes, each PE is handling a unique task. The latency of each task in the graph and the circuit parameters (figuring in the energy model) are randomly reconfigured at each simulation following a uniform distribution. All tests are performed on a 2GHz PC with 4GBytes of memory. The aim of this experiment is to cover a large set of system parameters to optimize, so that we can better characterize each method.
Simulations results are given in Table 1. They show that, for the same precision on the applied constraint, the principal loop of the Lagrangian method maintains a nearly constant number of iterations. However, the sum of performed iterations in the secondary process (the steepest algorithm) is variable and depends on the number of processors: the gradient optimization is more sensitive to the system parameters when the number of cores is limited.
Concerning the distributed approach, we observe that the game theoretic algorithm converges after the same number of iterations, independently of the number of cores. This is due to the greedy search used to limit the computational load of our proposal (see section 2.4). In fact, by testing just three sequential frequencies for each iteration, the convergence rate of this algorithm is constant, and the game theory approach is more likely to stagnate in the close local minimum. Figure 4 summarizes optimum error by reference to fmincon results of Matlab. Results show that our method is efficient in most of the cases, despite the local search algorithm used.

The total time to solution for the overall algorithm (augmented Lagrangian combined with the steepest descent) is given in the bottom plot of Figure 5. To study the scalability of one iteration of this algorithm, we measure the average processing time needed by one iteration inside the steepest gradient. The top plot in Figure 5 shows how a single iteration of this algorithm becomes dramatically more expensive as the problem size increases. Meanwhile, the game theoretic process converges faster in each test case, and as we expected, it is linearly scalable with the number of PE.

Table 1 – Convergence properties of the Steepest Algorithm

Platform Size	**Speed (number of iterations)**			
	Average	**Best case**	**Worst case**	**STD***
10 PEs	31.4	17	39	3.11
20 PEs	33.39	20	40	2.49
30 PEs	34.27	24	40	2.17
50 PEs	35.15	25	39	1.74
60 PEs	35.39	29	39	1.7
70 PEs	35.65	28	39	1.47
80 PEs	35.61	28	39	1.69

STD* : Standard Deviation value

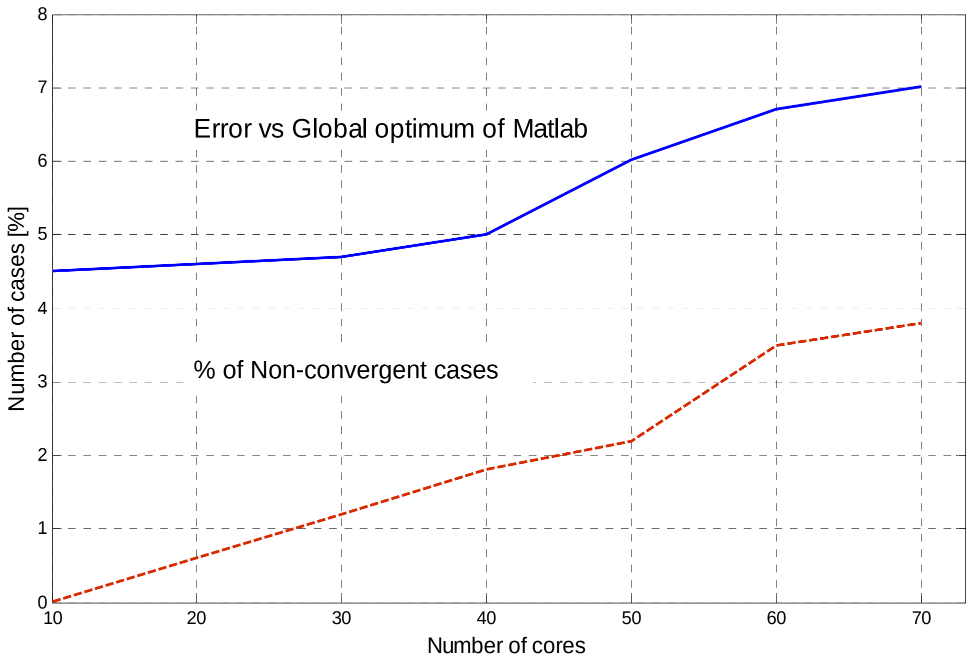

Figure 4. Optimization quality of the GT method vs number of cores

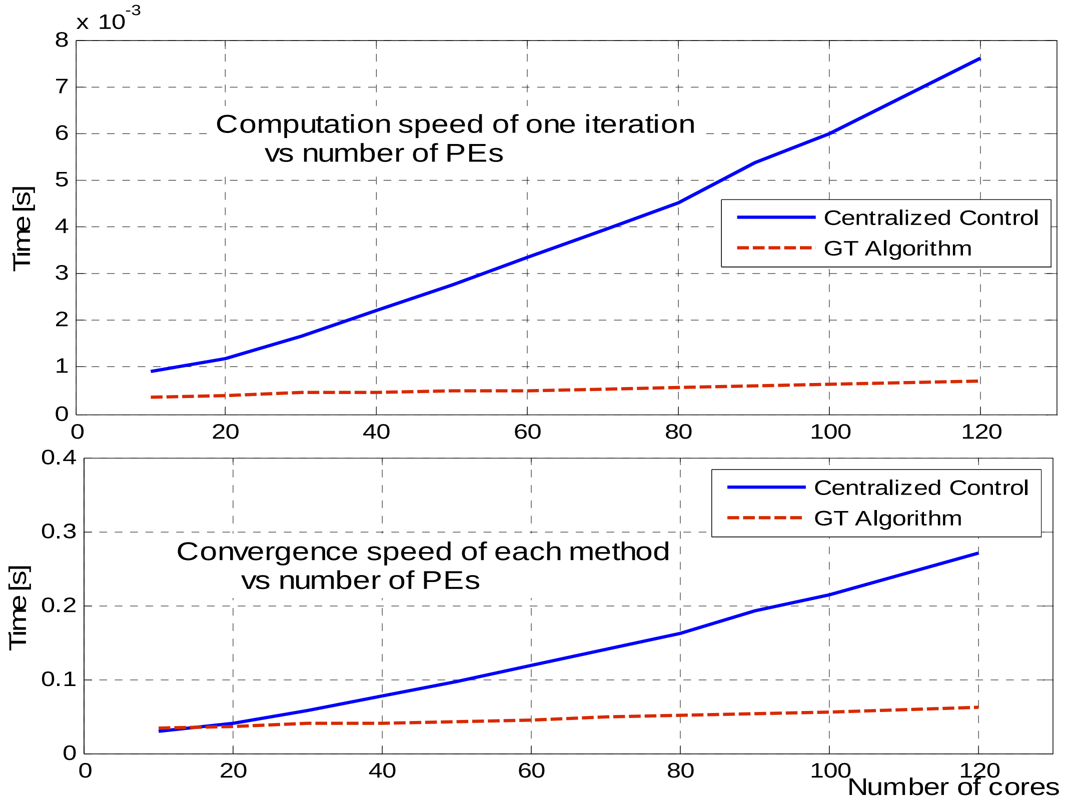

Figure 5. Convergence speed of the Lagrangian and GT methods vs number of cores

4 Conclusion

In this report, we have presented a distributed algorithm to optimize the energy consumption in MPSoC architectures while maintaining application constraints. Our proposal, inspired by game theory, aims at modeling PEs as independent players, which can adjust their local frequencies via an in situ DVFS engine. Each PE tries to minimize its power consumption with respect to the global system performances when running a given application. Game theory concepts are used to lead the system to a global satisfying equilibrium.

We have compared the convergence rate and the scalability of our algorithm to a centralized approach based on the augmented Lagrangian method, which is widely used to solve constrained optimization problem. As results show, the centralized algorithm computations scale poorly when the problem dimension increases. Hence, the reactivity time of such scheme increases when we consider large scale systems. On the contrary, the game theory approach has similar performances with linear scalability. It may even converge to a local minimum. Simulations show that this decentralized method can achieve high optimization quality and requires much less computations.

References

[1] J. F. Nash, "Non cooperative games," Annals of Mathematics, vol. 54, pp. 286–295, 1951.
[2] Nocedal, J. and Wright, S. (1999) Numerical Optimization, Springer Verlag.