Towards Autonomous Scalable Integrated Systems

Pascal Benoit, Gilles Sassatelli, Philippe Maurine, Lionel Torres, Nadine Azemard, Michel Robert, Fabien Clermidy, Marc Belleville, Diego Puschini, Bettina Rebaud, et al.

▶ **To cite this version:**

Pascal Benoit, Gilles Sassatelli, Philippe Maurine, Lionel Torres, Nadine Azemard, et al.. Towards Autonomous Scalable Integrated Systems. Design Technology for Heterogeneous Embedded Systems, Springer, pp.63-89, 2012, 978-94-007-1124-2. 10.1007/978-94-007-1125-9_4 . lirmm-01399454

HAL Id: lirmm-01399454
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01399454

Submitted on 24 Jun 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Chapter 4
Towards Autonomous Scalable Integrated Systems

Pascal Benoit, Gilles Sassatelli, Philippe Maurine, Lionel Torres, Nadine Azemard, Michel Robert, Fabien Clermidy, Marc Belleville, Diego Puschini, Bettina Rebaud, Olivier Brousse, and Gabriel Marchesan Almeida

1 Entering the Nano-Tera Era: Technology Devices Get Smaller (NANO), Gizmos Become Numerous (TERA) and Get Pervasive

Throughout the past four decades, silicon semiconductor technology has advanced at exponential rates in performance, density and integration. This progress has paved the way for application areas ranging from personal computers to mobile systems. As scaling and therefore complexity remains the main driver, scalability in the broad sense appears to be the main limiting factor that challenges complex system design methodologies. Therefore, not only technology (*fabricability*), but also structure (*designability*) and function (*usability*) are increasingly questioned on scalability aspects, and research is required on novel approaches to the design, use, management and programming of terascale systems.

1.1 The Function: Scalability in Ambient Intelligence Systems

Pervasive computing is a novel application area that has been gaining attention due to the emergence of a number of ubiquitous applications where context awareness is important. Examples of such applications range from ad-hoc networks of mobile terminals such mobile phones to sensor network systems aimed at monitoring

P. Benoit (✉) · G. Sassatelli · P. Maurine · L. Torres · N. Azemard · M. Robert · D. Puschini · B. Rebaud · O. Brousse · G.M. Almeida
LIRMM, UMR 5506, CNRS–Université Montpellier 2, 161 rue Ada, 34095 Montpellier Cedex 5, France
e-mail: Pascal.Benoit@lirmm.fr

F. Clermidy · M. Belleville · D. Puschini · B. Rebaud
CEA Leti, MINATEC, Grenoble, France

G. Nicolescu et al. (eds.), *Design Technology for Heterogeneous Embedded Systems*,
DOI 10.1007/978-94-007-1125-9_4, © Springer Science+Business Media B.V. 2012

geographical or seismic activity. This new approach to computing considerably enhances knowledge necessary for devising solutions capable of meeting application requirements. This is due to the emergence of uncertainty in such systems where environmental interactions and real-time conditions may change rapidly. Further, even though the problem may remain tractable for the small-scale systems used, solutions are not adapted, do not scale well and therefore face the curse of dimensionality.

A number of scientific contributions aimed at facilitating specification of applications [1] and formalizing the problem have emerged over the past decade, such as agent orientation, which promotes a social view of computing in which agents exchange messages, exhibit behaviors such as commitment, etc. The underlying challenge to the efficient design of such systems concerns the concepts behind autonomous systems able to monitor, analyze and make decisions. Machine learning/artificial intelligence techniques and bio-inspiration are among possible solutions that have been investigated for tackling such problems.

1.2 The Technology: Scalability in Semiconductor Technologies

Similarly, with the continued downscaling of CMOS feature size approaching the nanometer scale, the recurrent methods and paradigms that have been used for decades are increasingly questioned. The assumption of the intrinsic reliability of technology no longer holds [2] with increasing electric and lithographic dispersions, failure rates and parametric drifts. Beyond technological solutions, there is growing interest in the definition of self-adaptive autonomous tiles capable of monitoring circuit operation (delays, leakage current, etc.) and taking preventive decisions with respect to parameters such as voltage and frequency.

1.3 The Structure: Scalability in On-chip Architectures

Even though the abstraction of such technological issues may prove tractable, efficiently utilizing the ever-increasing number of transistors proves difficult. To this end, one popular design style relies on devising multicore/multiprocessor architectures [3]. Although such solutions have penetrated several market segments such as desktop computers and mobile terminals, traditional architectural design styles are challenged in terms of scalability, notably because of shared-memory oriented design, centralized control, etc. In this area again, there is growing interest for systems endowed with decisional capabilities. It is often believed that autonomous systems are a viable alternative that could provide adaptability at the chip level for coping with various run-time issues such as communication bottlenecks, fault tolerance, load balancing, etc.

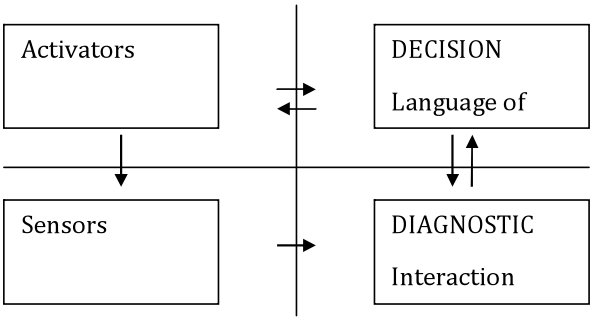

Fig. 4.1 Autonomous system infrastructure

1.4 Towards Multi-scale Autonomous Systems: A General Scheme

Autonomy is the faculty attributed to an entity that can be self-sufficient and act within its environment to optimize its functions. Autonomy also describes a system that can manage itself using its own rules. Autonomy is practiced by living organisms, people, and institutions but not yet by machines. However, the role of the mind in the architecture of autonomic systems is questioned.

In order to apply this concept to the reality of technological systems, this study will start with the abstract view of a system architecture while applying the notion of autonomy.

Figure 4.1 gives a synthetic view of autonomy: the activator creates the physical state of the system and the diagnosis motivates it. In microelectronics, and therefore for an SoC (System On a Chip), autonomy is represented by the fact that a calculation is distributed. In robotics, autonomy is the way to differ, deport or perform sequences of actions without risking damaging to the machine. In both cases, the command language must be able to schedule actions in parallel or in sequence. Autonomy can be used to lower energy consumption in microelectronics. In robotics, the challenge is to increase performance in different environments. In the artificial intelligence domain, autonomy is the consequence of a life cycle where the sensors observe, the diagnosis gives direction, the language of command orders and activators act.

Our objective is to design a fully scalable system and apply autonomy principles to MPSoC (Multiprocessor System-on-Chip). In this chapter, we will first discuss our vision of the infrastructure required for scalable heterogeneous integrated systems. Then we will provide a general model for self-adaptability, exemplified with respect to variability compensation, dynamic voltage and frequency scaling and task migration. Finally, an example of an autonomous distributed system that has been developed in the Perplexus European project will be provided.

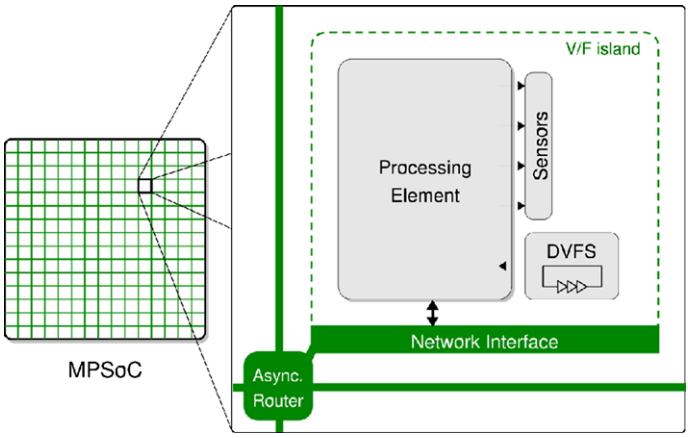

Fig. 4.2 Generic MPSOC architecture

2 Distributed MPSoC Systems

In this section, we discuss our vision of a generic MPSoC architecture supported by two examples. We analyze and suggest the features of a possible scalable and self-adaptive model suitable for future autonomous systems.

2.1 Generic MPSoC Architecture

This section describes a generic MPSoC by only introducing the key elements allowing formulating valid hypotheses on the architecture. The considered MPSoC is composed of several Processing Elements (PE) linked by an interconnection structure as described in Fig. 4.2.

2.1.1 Processing Elements

PEs of an MPSoC depend on the application context and requirements. There are two architecture families. The first includes heterogeneous MPSoCs composed of different PEs (processors, memories, accelerators and peripherals). These platforms were pioneered by the C-5 Network Processor [4], Nexperia [5] and OMAP [6]. The second family represents homogeneous MPSoCs, e.g. as proposed by the Lucent Daytona architecture [3], where the same tile is instantiated several times. This work targets both topologies. Thus, Fig. 4.2 represents a homogeneous or heterogeneous design.

2.1.2 Interconnection

The PEs previously described are interconnected by a Network-on-Chip (NoC) [7–10]. A NoC is composed of Network Interfaces (NI), routing nodes and links. NI implements the interface between the interconnection environment and the PE domain. It decouples computation from communication functions. Routing Nodes are in charge of routing the data between the source and destination PEs through links. Several network topologies have been studied [11, 12]. Figure 4.2 represents a 2D mesh interconnect. We consider that the offered communication throughput is enough for the targeted application set. NoC fulfills the "Globally Asynchronous Locally Synchronous" (GALS) concept by implementing asynchronous nodes and asynchronous-synchronous interfaces in NIs [13, 14]. As in [15], GALS properties allow MPSoC partitioning into several Voltage Frequency Islands (VFI). Each VFI contains a PE clocked at a given frequency and voltage. This approach allows real fine-grain power management.

2.1.3 Power Management

Dividing the circuit into different power domains using GALS has facilitated the emergence of more efficient designs that take advantage of fine-grain power management [16]. As in [17, 18], the considered MPSoC incorporates distributed Dynamic Voltage and Frequency Scaling (DVFS): each PE represents a VFI and includes a DVFS device. It consists of adapting the voltage and frequency of each PE in order to manage power consumption and performance. A set of sensors integrated within each PE provides information about consumption, temperature, performance or any other metric needed to manage the DVFS.

2.2 Examples: Heterogeneous and Homogeneous MPSoC

Nowadays, there are several industrial and experimental MPSoC designs targeting different application domains that fulfill part or all of the characteristics enumerated in the previous section. We briefly describe two examples: ALPIN from CEA-LETI and HS-Scale from LIRMM.

2.2.1 Alpin

Asynchronous Low Power Innovative Network-on-Chip (ALPIN) is a heterogeneous demonstrator [17, 18] developed by CEA-LETI. The ALPIN circuit is a GALS NoC system implementing adaptive design techniques to control both dynamic and static power consumption in CMOS 65 nm technology. It integrates 6 IP (Intellectual Property) units: a TRX-OFDM unit, 2 FHT units, a MEMORY unit,

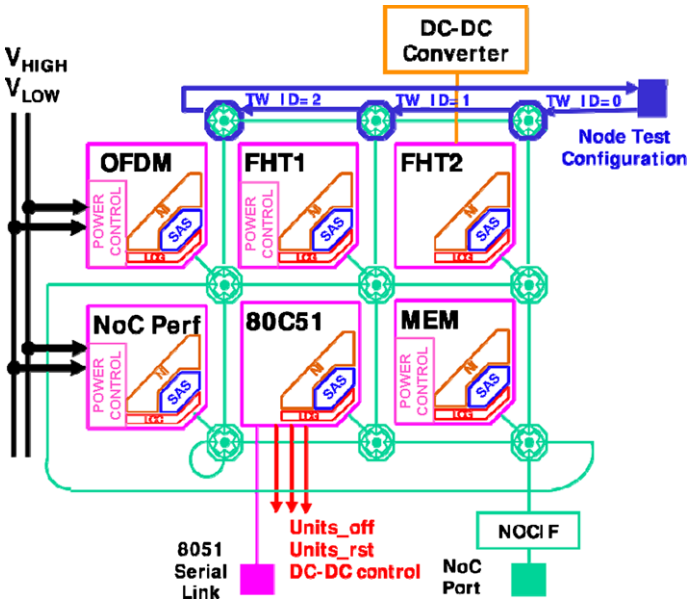

Fig. 4.3 ALPIN architecture

a NoC performance analysis unit and a 80c51 for power mode programming, as shown in Fig. 4.3. The interconnection is provided by 9 NoC asynchronous nodes and one NOC synchronous external interface. The asynchronous Network-On-chip provides 17 GBit/s throughput and automatically reduces its power consumption by activity detection.

Both dynamic and static power are reduced using adaptive design techniques. ALPIN IP units handle 5 distinct power modes. Using VDD-Hopping, dynamic power consumption can be reduced by 8-fold. By using Ultra-Cut-Off, static power consumption can be reduced by 20-fold.

2.2.2 HS-Scale: A Homogeneous MPSoC from LIRMM

Hardware-Software Scalable (HS-Scale) is a regular array of building blocks (Fig. 4.4) [19, 20]. Each tile is able to process data and to forward information to other tiles. It is named NPU (Network Processing Unit) and is characterized by its compactness and simplicity.

The NPU architecture is represented in Fig. 4.4. This architecture contains: a processor, labeled PE in Fig. 4.4; memory to store an Operating System (OS), a given application and data; a routing engine which transfers messages from one port to another without interrupting processor execution; a network interface between the router and the processor based on two hardware FIFOs; an UART that allows uploading of the operating system and applications; an interrupt controller to manage interrupt levels; a timer to control the sequence of an event; and a decoder to address

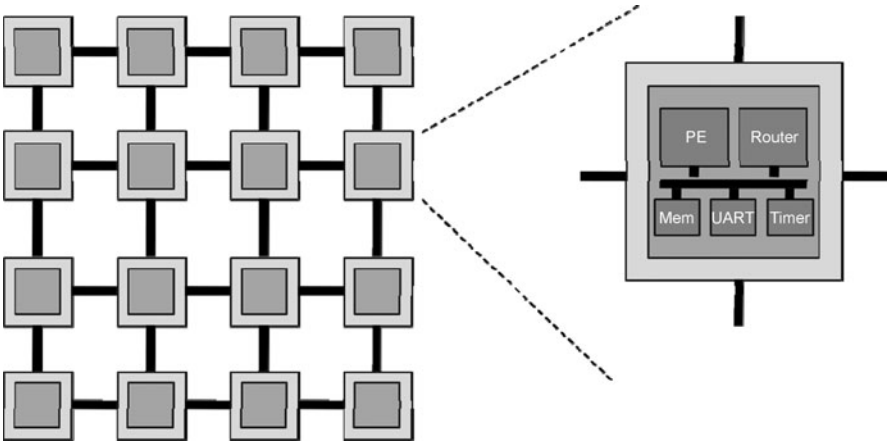

Fig. 4.4 HS-Scale architecture (from [19])

these different hardware entities. An asynchronous wrapper interfaces the processor with the routing engine, allowing several frequency domains and guaranteeing GALS behavior. The system is controlled by a distributed OS specifically designed for this platform, which provides self-adaptive features. It ensures load balancing by implementing task migration techniques.

2.3 Conclusion

Most industrial approaches for embedded systems are heterogeneous. Performance, power efficiency and design methods have been the biggest drivers of such technologies, but the lack of scalability is becoming a major issue with respect to tackling the inherent complexity [21]. Regular designs based on homogeneous architectures such as HS-Scale provide potential scalability benefits in terms of design, verification, program, manufacturing, debug and test. Compared to heterogeneous architectures, the major drawback could be the performance and power efficiency, but our goal is homogeneity in terms of regularity: each processing element could be "heterogeneous", i.e. composed of several processing engines (general purpose processor, DSP, reconfigurable logic, etc.) and instantiated many times in a regular design: we talk about globally homogeneous and locally heterogeneous architectures.

With a homogeneous system, each task of a given application can potentially be handled by any processing element of the system. Assuming that we can design a self-adaptive system with many possible usages, as illustrated in Fig. 4.5: task migration to balance workload and reduce hot spots, task remapping after a processing element failure, frequency and voltage scaling to reduce the power consumption, etc. But to benefit such a potential, we need to define a complete infrastructure, as outlined in the next section.

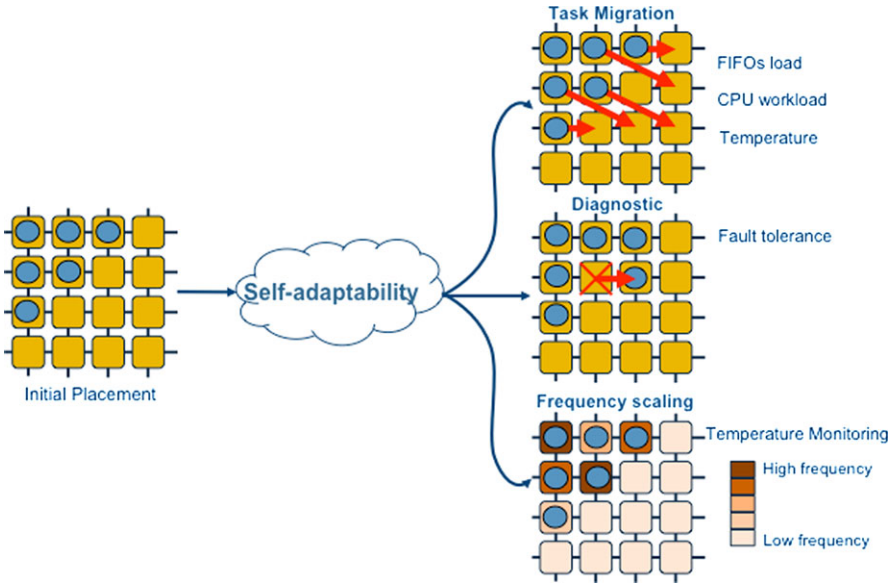

Fig. 4.5 Potential usage of a self-adaptive homogeneous system

3 Self-adaptive and Scalable Systems

In our approach, the infrastructure of a self-adaptive system should enable monitoring of the system, diagnosis, and the optimization process for making the decisions to modify a set of parameters or actuators. Applied to a homogeneous MPSOC system, this infrastructure is presented in Fig. 4.6 and should be completely embedded into the system itself.

In the following sections, we present three contributions to this infrastructure at three levels:

- the design of sensors for process monitoring allowing PVT (Process Voltage Temperature) compensation
- the implementation of a distributed and dynamic optimization inspired by Game-Theory for power optimization
- the implementation of task migration based on software monitors to balance the workload

3.1 Dynamic and Distributed Monitoring for Variability Compensation

To move from fixed integrated circuits to self-adaptive systems, designers must develop reliable integrated structures providing, at runtime, the system (or any PVT

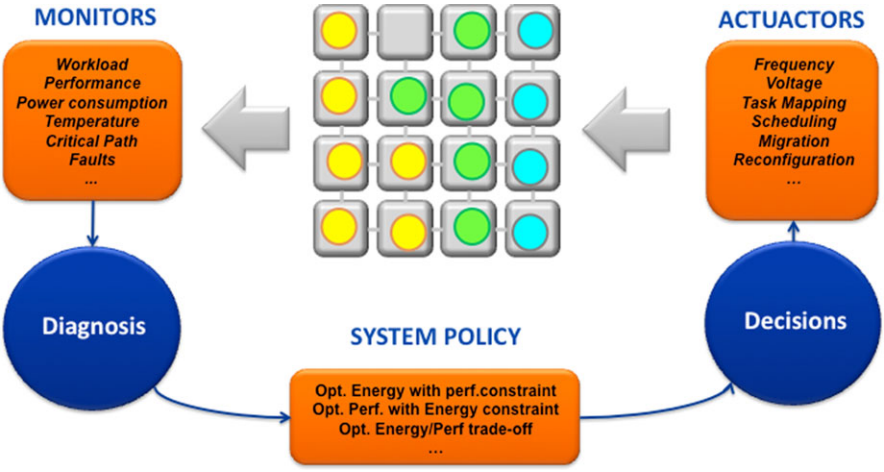

Fig. 4.6 Infrastructure of a self-adaptive system

hardware manager) with trustable and valuable information about the state of the hardware.

Monitoring Clocked System on Chips made of a billion transistors at a reasonable hardware and performance cost is an extremely difficult task for many reasons. Among them one may found, the increasing random nature of some process parameters, the spatial dependence of process (including aging), voltage and temperature variations, but also the broad range of time constants characterizing variations in these physical quantities.

Two different approaches to the monitoring problem can be found in the literature. The first one consists of integrating specific structures or sensors to monitor, at runtime, the physical and electrical parameters required to dynamically adapt the operating frequency and/or the supply voltage and/or the substrate biasing. Several PVT sensors commonly used for post fabrication binning have been proposed in the literature [22–26] for global variability compensation. However, there are some limitations to the use of such PVT sensors.

First, their area and power consumption may be high, so their number has to be limited. Second, their use requires: (a) integration of complex control functions in LUT, and (b) intensive characterization of the chip behavior w.r.t. the considered PVT variables.

Finally, another limitation of this approach concerns the use of Ring Oscillator (RO) structures [22–24] to monitor the circuit speed since RO may be sensitive to PVT variables which are quite different from those of data paths. However, this second limitation can be overcome by adopting a replica path approach, as proposed in [25]. It involves monitoring the speed of some critical paths which are duplicated in the sensors to replace the traditional RO.

The second approach, to compensate for PVT variations and aging effects, is to directly monitor sampling elements of the chip (Latches or D-type Flip Flop) to detect delay faults. This can be achieved by inserting specific structures or using ad-hoc sampling elements [26, 27] to detect a timing violation by performing a delayed comparison or by detecting a signal transition within a given time window.

This approach has several advantages, with the main one being its ability to detect the effects of local and dynamic variations (such local hot spots, localized and brief voltage drops) on timings. A second and significant advantage is the interpretation of the data provided by the sensors, which is simple and binary.

However, this second approach has some disadvantages. One of them is that a high number of sensors might be required to obtain full coverage of the circuit. Therefore, these structures must be as small as possible and consume a small amount of energy when the circuit operates correctly. A second and main disadvantage of such kind of sensors [26, 27] is that error detection requires full replay of the processor instruction at a lower speed. However, this replay is not necessarily possible if the 'wrong' data has been broadcasted to the rest of the chip.

In this setting, a solution is to monitor the timing slack pattern with PVT variations of the critical sampling elements of circuits rather than detecting errors. We thus developed a new monitoring structure, in line with [26–29] concepts, aimed at anticipating timing violations over a wide range of operating conditions. This timing slack monitor, which is compact and has little impact on the overall power consumption, may allow application of dynamic voltage and/or frequency scaling as well as body bias strategies.

Figure 4.7 shows the proposed monitoring system and its two blocks detailed in [30]: the sensor and the specific programmable Clock-tree Cell (CC). The sensor, acting as a stability checker, is intended to be inserted close to the D-type Flip-Flops (DFF) located at the endpoints of the critical timing paths of the design while the CC are inserted at the associated clock leaves. Note that critical data paths to be monitored can be chosen by different means such through the selection of some critical paths provided either by a usual STA (Static Timing Analysis) or a SSTA (Statistical STA).

To validate the monitoring system and its associated design flow, the monitoring system has been integrated, according to [31], in an arithmetic and reconfigurable block of a 45 nm telecom SoC. This block contains about 13400 flip-flops, which leads to a 600×550 μm^2 core floorplan implementation. Intensive simulations of certain *part* of the arithmetic block demonstrated the efficiency of the monitoring system which allows anticipating timing violations (a) over the full range of process and temperature conditions considered to validate actual designs, and (b) for supply voltage values ranging from 1.2 V to 0.8 V thanks to the programmable CC.

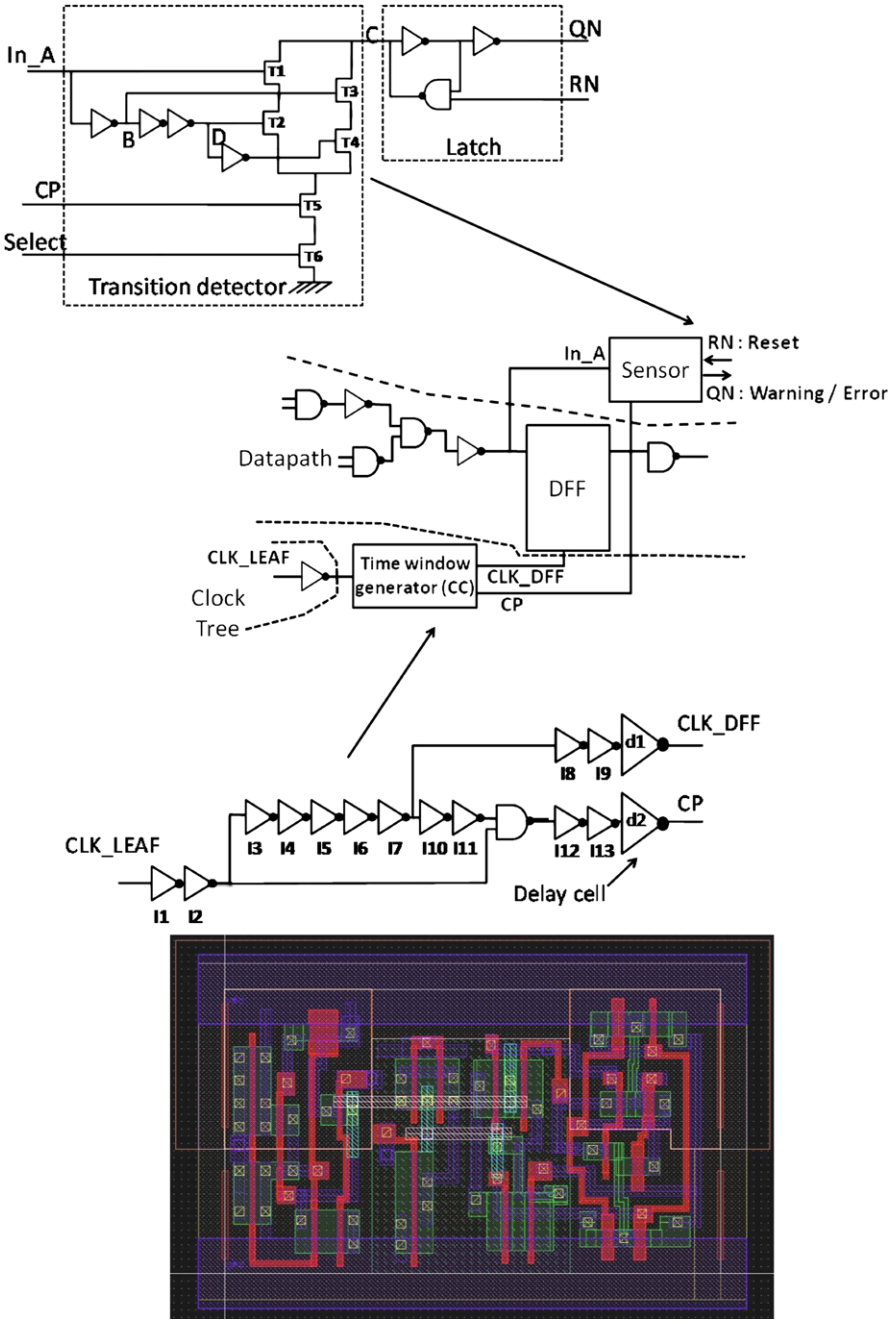

Fig. 4.7 Monitoring system implemented on a single path and the sensor layout in 45 nm technology

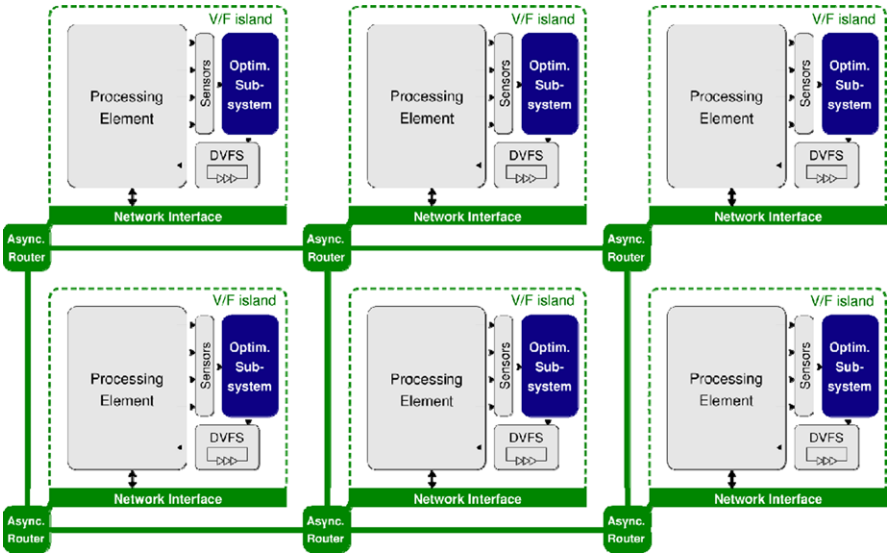

Fig. 4.8 Distributed dynamic optimization of MPSoC

3.2 Dynamic and Distributed Optimization Inspired by Game Theory

3.2.1 Distributed and Dynamic Optimization

Existing methods [32–40], even if they operate at run time, are not based on distributed models. An alternative solution to centralized approaches is to consider distributed algorithms. Our proposal is to design an architecture, as illustrated in Fig. 4.8, where each processing element of an MPSoC embeds an optimization subsystem based on a distributed algorithm. This subsystem manages the local actuators (DVFS in Fig. 4.8) that take the operating conditions into account. In other words, our goal is to design a distributed and dynamic optimization algorithm.

3.2.2 Game Theory as a Model for MPSoC Optimization

Game theory involves a set of mathematical tools that describe interaction among rational agents. The basic hypothesis is that agents pursue well-defined objectives and take their knowledge and behaviors of other agents in the system into account to make their choices. In other words, it describes interactions of players in competitive games. Players are said to be rational since they always try to improve their score or advance in the game by making the best move or action. Game theory is based on a distributed model: players are considered as individual decision makers. For these reasons, game theory provides a promising set of tools to model distributed optimization on MPSOC and, moreover, this is an original approach in this context.

(a) Players analyse the scenario in order to choose the best action (b) Players choose and play their actions

Fig. 4.9 A non-cooperative simultaneous game

As illustrated in [41], a non-cooperative strategic game Γ is composed of a set N of n players, a set of actions per player S_i and the outcomes u_i, $\forall i \in N$. In such a game, players N interact and play through their set of actions S_i in a non-cooperative way, in order to maximize u_i. Consider the non-cooperative game of Fig. 4.9 consisting of 4 players. In Fig. 4.9(a), players analyze the scenario. Each one incorporates all possible information by communicating or estimating it. The information serves to build a picture of the game scenario, to analyze the impact of each possible action on the final personal outcome. Finally, each player chooses the best action that maximizes his/her own outcome. Then, as shown in Fig. 4.9(b), players play their chosen actions and recalculate the outcome. Note that due to a set of interactions and choices of other players, the results are not always the estimated or desired ones. If this sequence is repeated, players have a second chance to improve their outcomes, but then they know the last movements of the others. Thus, players improve their chances of increasing their outcomes when the game is repeated several times. In other words, they play a repetitive game. After a given number of repetitions, players find a Nash equilibrium solution if it exists. At this time, players no longer change their chosen action between two cycles, indicating that they can no longer improve their outcomes.

Consider now that the game objective is to set the frequency/voltage couple of each processing element of the system represented in Fig. 4.10 through the distributed fine-grain DVFS. The figures represent a MPSoC integrating four processing elements interconnected by an NoC. The aim of the frequency selection is to optimize some given metrics, e.g. power consumption and system performance. These two metrics usually depend not only on the local configuration but also on the whole system due to the applicative and physical interactions.

In such scenarios, each processing element is modeled as a player in a game like the one in Fig. 4.9. In this case, the set of players N consists of n tiles of the system ($n = 4$ in the figure). The set of actions S_i is defined by each possible frequency set by the actuator (DVFS). Note that now communications between players are made through the interconnection system. In Fig. 4.10(a), tiles analyze the scenario like in Fig. 4.9(a). They estimate the outcome of each possible action depending on the global scenario in terms of the optimization metrics (energy consumption and performance). The estimation is coded in the utility function u_i. Then, in Fig. 4.10(b), processing elements choose the actions that maximize the outcome. Finally, they execute them, like in Fig. 4.9(b).

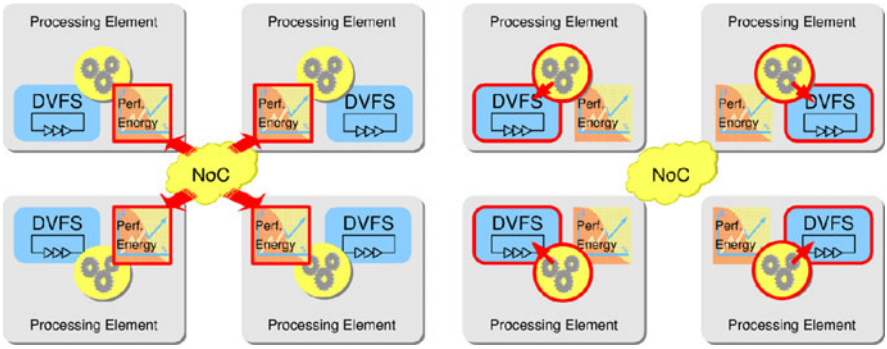

Fig. 4.10 MPSoC modeled as a non-cooperative simultaneous game

MPSoC are distributed architectures. In addition, the presence of distributed actuators such as fine-grain DVFS, justifies the use of a non-cooperative models. These models are based on the principle that decisions are made by each individual in the system. This hypothesis matches the described MPSoC scenario. In MPSoCs, tiles cannot be aware of the state of the whole system and decisions of others, but they have partial information. This is the case of incomplete and imperfect information games. If players do not have a correct picture of the whole scenario, the NE can be hardly reached in the first turn of the game. An iterative algorithm providing several chances to improve the choices will also provide more chances to reach the NE. The distributed nature of MPSoCs also make it hard to synchronize the decision time of all players. In other words, no playing order is set in order to avoid increasing the system complexity. Players are allowed to play simultaneously. For these reasons, our proposal is based on a non-cooperative simultaneous repetitive game.

3.2.3 Scalability Results

The evaluation scenario proposed in [41] illustrates the effectiveness of such techniques. The objective of this proof of concept is to provide a first approach and to characterize its advantages and problems. The metric models used in this formulation are very simple, offering a highly abstracted view of the problem. However, they provide a strong basis for presenting our approach. The statistical study proved the scalability of our method (Fig. 4.11). An implementation based on a well-known microcontroller has highlighted its low complexity. This conclusion comes from an abstracted analysis. In addition, the statistical study showed some deficiencies in terms of convergence percentage, leading to the development of a refined version of the algorithm.

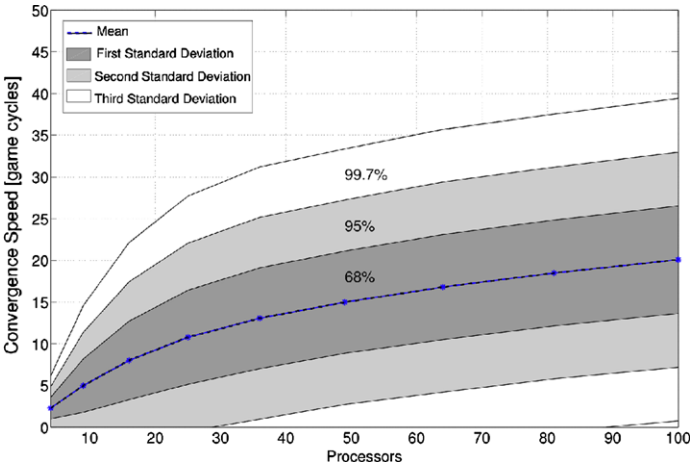

Fig. 4.11 Convergence speed from 4 to 100 PEs

3.2.4 Energy and Latency Optimization Results

In [42] and [43], a new algorithm has been proposed. The number of comparisons per iteration cycle has been markedly reduced, thus simplifying the implementation. The new procedure was examined using four TX 4G telecommunication applications. The results (Fig. 4.12) show that the system adapts the performances when the application changes during execution time. The proposed procedure adapts, in a few cycles, the frequency of each PE. Moreover, when the external constraints (energy and latency bounds) change, the system also reacts by adapting the frequencies. For the tested applications, we have observed improvements of up to 38% in energy consumption and 20% in calculation latency. Compared to an exhaustive optimal search, our solution is less than 5% of the Pareto optimal solution.

3.3 Workload Balancing with Self-adaptive Task Migration

As the key motivations of HS-Scale [19, 20] are scalability and self-adaptability, the system is built around a distributed memory/message passing system that provides efficient support for task migration. The decision-making policy that controls migration processes is also fully distributed for scalability reasons. This system therefore aims at achieving continuous, transparent and decentralized run-time task placement on an array of processors for optimizing application mapping according to various potentially time-changing criteria.

Each NPU has multitasking capabilities, which enable time-sliced execution of multiple tasks. This is implemented thanks to a tiny preemptive multitasking Operating System, which runs on each NPU. Structural (a) and functional (b) views of the NPU are depicted in Fig. 4.13. The NPU is built around two main layers, the

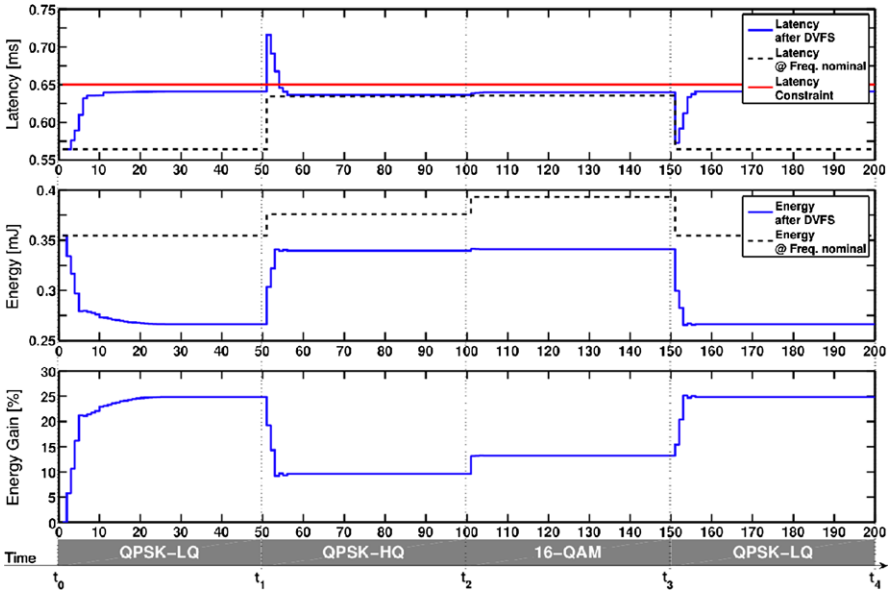

Fig. 4.12 Energy consumption minimization under latency constraints

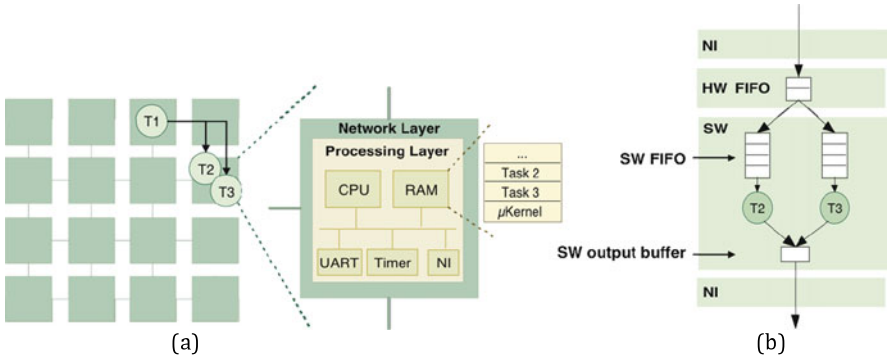

Fig. 4.13 HS-scale principles

network layer and the processing layer. The Network layer is essentially a compact routing engine (XY routing). Packets are read from incoming physical ports, and then forwarded to either outgoing ports or the processing layer. Whenever a packet header specifies the current NPU address, the packet is forwarded to the network interface (NI). The NI buffers incoming data in a small hardware FIFO (HW FIFO) and simultaneously triggers an interrupt to the processing layer. The interrupt then activates data de-multiplexing from the single hardware FIFO to the appropriate software FIFO (SW FIFO), as illustrated. The processing layer is based on a simple and compact RISC microprocessor, its static memory, and a few peripherals (one

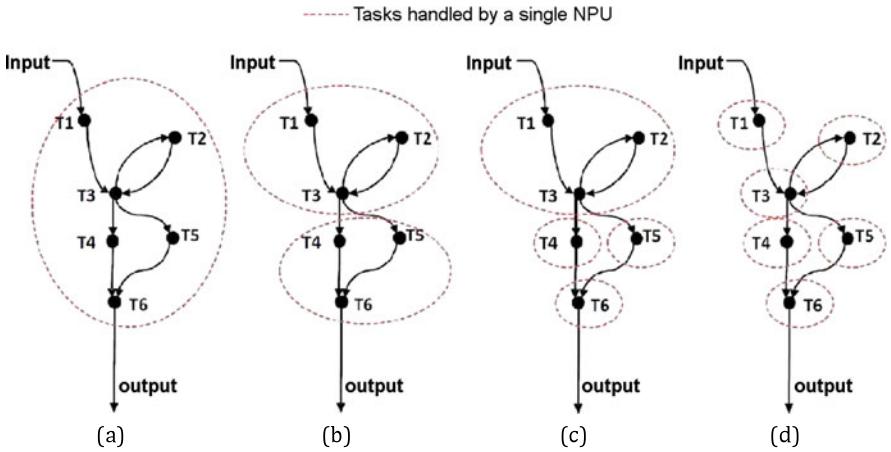

Fig. 4.14 Dynamic task-graph mapping

timer, one interrupt controller, one UART). A multitasking microkernel (μKernel) implements the support for time-multiplexed execution of multiple tasks.

The platform is entitled to make decisions that relate to application implementation through task placement. These decisions are taken in a fully decentralized fashion as each NPU is endowed with equivalent decisional capabilities. Each NPU monitors a number of metrics that drive an application-specific mapping policy. Based on this information, an NPU may decide to push or attract tasks, which results in respectively parallelizing or serializing the corresponding task executions, as several tasks running on the same NPU are executed in a time-sliced manner.

Figure 4.14 shows an abstract example showing that upon application loading the entire task graph runs on a single NPU, subsequent remapping decisions then tend to parallelize application implementation as the final step exhibits one task per NPU. Similarly, whenever a set of tasks become subcritical the remapping could revert to situation (c), where T1, T2 and T3 are hosted on a single NPU while the other supposedly more demanding tasks do not share NPU processing resources with other tasks. These mechanisms help in achieving continuous load-balancing in the architecture but can, depending on the chosen mapping policy, help in refining placement for lowering contentions, latency or power consumption.

3.3.1 Task Migration Policies

Mapping decisions are specified on an application-specific basis in a dedicated operating system service. Although the policy may be focused on a single metric, composite policies are possible. Three metrics are available to the remapping policy for making mapping decisions:

- **NPU load:** The NPU operating system has the capability of evaluating the processing workload resulting from task execution.

- **FIFO queue filling level:** As depicted in Fig. 4.13, every task has software input FIFO queues. Similarly to NPU load, the operating system can monitor the filling of each FIFO.
- **Task distance:** The distance that separates tasks is also a factor that impacts performance, contentions in the network and power consumption. Each NPU microkernel knows the placement of other tasks of the platform and can calculate the Manhattan distance with the other tasks it communicates with.

The code below shows an implementation of the microkernel service responsible for triggering task migration. The presented policy simply triggers task migration in case one of the FIFO queues of a task is used over 80%.

```
void improvement_service_routine(){
   int i, j;
   //Cycles through all NPU tasks
   for(i=0; i < MAX_TASK; i++){
   //Deactivates policy for dead/newly instantiated tasks
      if(tcb[i].status != NEW && tcb[i].status != DEAD){
         //Cycles through all FIFOs
         for(j=0; j < tcb[i].nb_socket; j++){
            //Verifies if FIFO usage > MAX_THRESHOLD
            if(tcb[i].fifo_in[j].average > MAX_THRESHOLD){
               //Triggers migration procedure if task
               //is not already alone on the NPU
               if(num_task > 1)
                  request_task_migration(tcb[i].task_ID);
            }
         }
      }
   }
}
```

The request task migration() call then sequentially emits requests to NPUs in proximity order. The migration function will migrate the task to the first NPU which has accepted the request, the migration process is started according to the protocol described previously in Sect. 4.2. This function can naturally be tuned on an application/task specific basis and select the target NPU while taking not only the distance but also other parameters such as available memory, current load, etc., into account.

We also implemented a migration policy based on the CPU load. The idea is very similar to the first one and it consists of triggering a migration of a given task when the CPU load is lower or greater than a given threshold. This approach may be subdivided in two subsets:

(1) Whenever the tasks time $\geq$ MAX THRESHOLD, this means that tasks are consuming more than or equal to the maximum acceptable usage of the CPU time;
(2) Whenever the tasks time $<$ MIN THRESHOLD, this means the tasks are consuming less than the minimum acceptable usage of the CPU time.

For both subsets, the number of tasks inside one NPU must be verified. For the first subset, it is necessary to have at least two tasks running in the same NPU. For

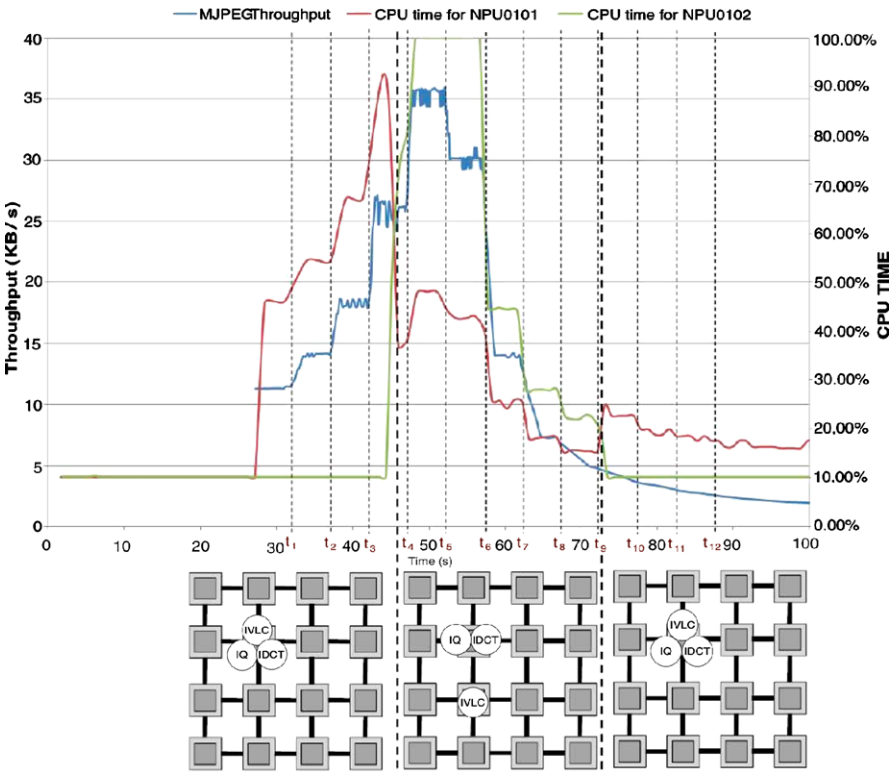

Fig. 4.15 MJPEG throughput with the diagnosis and decision based on CPU workload

the second subset, the migration process may occur whenever there are one or more tasks in the same NPU.

In the same way, the migration process occurs whenever the CPU load is less than MIN_ THRESHOLD (20%). When this occurs, the migration function must look for an NPU that is being used at given CPU usage threshold, i.e. 60% usage in this case. To keep tasks with less than MIN_THRESHOLD from migrating every time, we inserted a delay to reduce the number of migrations.

3.3.2 Results: Task Migration Based on CPU Workload

The example on Fig. 4.15 shows the results of applying this migration policy based on the CPU workload. The experimental protocol used for these results involves varying the input data rate to observe how the system adapts.

At the beginning, all tasks (IVLC, IQ and IDCT) are running on the same NPU(1,1) but the input throughput on the MJPEG application is lower, so the CPU time consumed is around 47%. The input throughput is increased at each step (t_1, t_2 and t_3) so we can see an increase in the CPU time consumed step by step. When the

CPU time used exceeds the threshold (i.e. 80%), the operating system detects that the NPU(1,1) is overloaded (at 45 s), so it decides to migrate the task which uses the most CPU time on a neighboring NPU. In this example, IVLC tasks migrate on NPU(1,2), which decreases the CPU time used by NPU(1,1) by around 35% and increases the CPU used by NPU(1,2) by around 80%. At t_4, the input throughput increases more, which leads to an MJPEG throughput increase of around 35 KB/s and overloads the NPU(1,2) at 100% but no migration is triggered because just one task is computed. From t_5 to t_{12}, the input throughput of the MJPEG application is decreased step by step and, when the CPU time of NPU(1,2) is less than 20% (at 72 s), the operating system decides to move task on the same NPU (the NPU(1,1)). After this migration, we can see a decrease in CPU time used by the NPU(1,2) and an increase in CPU time used by NPU(1,1) but without saturating it. We can observe that the MJPEG application performance is lower than in the static mode because the operating system uses more CPU time (around 10%) to monitor CPU time.

4 Towards Autonomous Systems

The growing interest in pervasive systems that seamlessly interact with their environment motivates research in the area of self-adaptability. Bio-inspiration is often regarded as an attractive alternative to the usual optimization techniques since it provides capability to handle scenarios beyond the initial set of specifications. Such a feature is crucial in multiple domains such as pervasive sensor networks where nodes are distributed across a broad geographical area, thus making on-site intervention difficult. In such highly distributed systems, the various nodes are loosely coupled and can only communicate by means of messages. Further, their architecture may differ significantly as they may be assigned tasks of different natures. One interesting opportunity is to use agent-orientation combined with bio-inspiration to explore the resulting adaptive characteristics.

4.1 Bio-inspiration & Agent-Orientation: at the Crossroads

Programming distributed/pervasive applications is often regarded as a challenging task that requires a proper programming model capable of adequately capturing the specifications. Agent-oriented programming (AOP) derives from the initial theory of agent orientation, which was first proposed by Yoav Shoham [44]. Agent-orientation was initially defined for promoting a social view of computing and finds natural applications in areas such as artificial intelligence or social behavior modeling. An AOP computation consists of making agents interact with each other through typed messages of different natures: agents may be informing, requesting, offering, accepting, and rejecting requests, services or any other type of information. AOP also sets constraints on the parameters defining the state of the agent (beliefs, commitments and choices).

For exploring online adaptability, bio-inspiration appears to be an attractive alternative that has been used for decades in many areas. Optimization techniques such as genetic programming, artificial neural networks are prominent examples of such algorithms. There are several theories that relate to life, its origins and all of its associated characteristics. It is, however, usually considered that life relies on three essential mechanisms, i.e. phylogenesis, ontogenesis and epigenesis [45] (referred to as P, O and E, respectively, throughout this chapter):

- Phylogenesis is the origin and evolution of a set of species. Evolution gears species towards a better adaptation of individuals to their environment; genetic algorithms are inspired from this principle of life.
- Ontogenesis describes the origin and the development of an organism from the fertilized egg to its mature form. Biological processes like healing and fault tolerance are ontogenetic processes.
- Epigenesis refers to features that are not related to the underlying DNA sequence of an organism. Learning as performed by Artificial Neural Networks (ANN) is a process whose scope is limited to an individual lifetime and therefore is epigenetic.

4.2 The Perplexus European Project

The PERPLEXUS European project aims at developing a platform of ubiquitous computing elements that communicate wirelessly and rely on the three above-mentioned principles of life. Intended objectives range from the simulation of complex phenomena such as culture dissemination to the exploration of bio-inspiration driven system adaptation in ubiquitous platforms.

Each ubiquitous computing module (named Ubidules for Ubiquitous Modules) is made of an XScale microprocessor that runs a Linux operating system and a bio-inspired reconfigurable device that essentially runs Artificial Neural Networks (ANN). The resulting platform is schematically described in Fig. 4.16, which shows the network of mobile nodes (MANET) that utilize moving vehicles, and the Ubidules that control them.

4.3 Bio-mimetic Agent Framework

The proposed framework is based on the JADE (Java Agent DEvelopment kit) open-source project. The lower-level mechanisms such as the MANET dynamic routing engine are not detailed here, refer to [46] for a complete description. This section focuses on two fundamental aspects of the proposed BAF: on one hand a description of the BAF and overview of the provided functionality, on the other a description of POE specific agents. Further information on BAF can be found in [47].

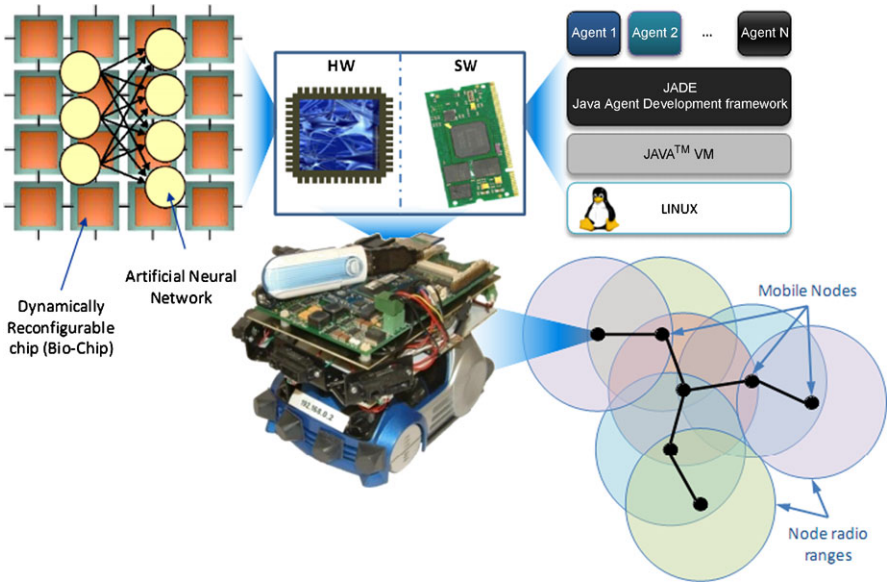

Fig. 4.16 Overview of the Perplexus platform

As bio-inspiration and the three fundamentals of life are at the core of the project, the proposed framework extends JADE default agents by defining agents whose purpose is related to both interfacing and bio-inspired (POE) mechanism support as well as pervasive computing platform management agents.

The BAF specifies 7 agents belonging to 2 families:

- Application agents: Phylogenetic agent(s), Ontogenetic agent(s) and Epigenetic agent(s).
- Infrastructure agents: UbiCom agent(s), Interface agent(s), Network agent(s) and Spy agent(s).

Figure 4.17 shows both the infrastructure and application agents and their interactions (for clarity, JADE-specific agents are omitted):

- P agent: The Phylogenetic agent is responsible for execution of the distributed Genetic Algorithms: it calculates the local fitness of the individual (the actual Ubidule) and synchronizes this information with all other Ubidules. It is responsible for triggering the death (end of a generation) and birth of the embodied individual hosted on the Ubidule.
- O agent: The Ontogenetic agent is tightly coupled to the P agent: it takes orders from this agent and has the capability of creating other software agents (in case of full software implementation).
- E agent: The Epigenetic agent embodies the individual and its behavior: it is a software or hardware neural network.

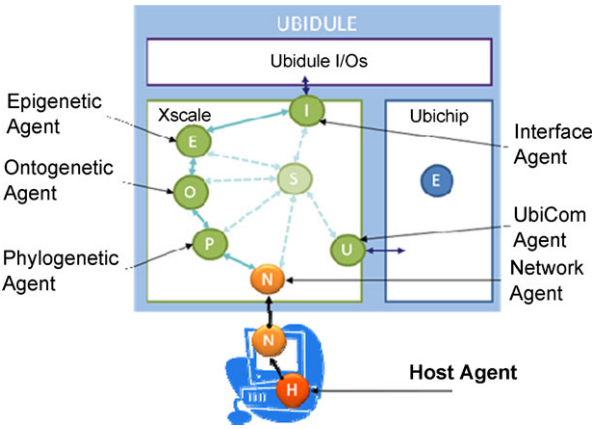

Fig. 4.17 BAF agents at the Ubidule-level

Next to the three POE agents, there are four additional agents for interfacing and networking purposes:

- I agent: The Interface agent provides a set of methods for issuing commands to the actuators or retrieving data from the Ubidule sensors.
- U agent: The UbiCom agent provides software API-like access to the Ubichip and manages hardware communications with the chip.
- S agent: The Spy agent provides information on the platform state (agent status/results, activity traces, bug notification).
- N agent: The Network agent provides a collection of methods for network-related aspects: time-synchronization of data among Ubidules, setting/getting clusters of Ubidules, obtaining a list of neighbors, etc. As it requires access to low-level network-topology information, it also implements MANET functionalities.

Finally, a Host agent (H agent) instantiated on a workstation allows remote control of the PERPLEXUS platform (Start/Stop/Schedule actions).

4.4 Application Results: Online Collaborative Learning

Figure 4.18 schematically depicts the robots used, their sensors and actuators, as well as the framework agents presented previously. Robots use online learning (Epigenesis) to improve their performance. Robots are enclosed in an arena scattered with obstacles (collision avoidance is the main objective here). As this application only targets learning, the P and O agents are not used here.

Besides the three front sensors that return the distance to the nearest obstacle, a bumper switch is added to inform the robot whenever a collision with an object occurs; it is located on the front side of the robot. These robots move by sending speed commands on each of the two motors. As depicted in Fig. 4.18 an Artificial Neural Network (ANN) controls the robot movement: the E agent is a multi-layer Perceptron ANN that uses a standard back-propagation learning algorithm.

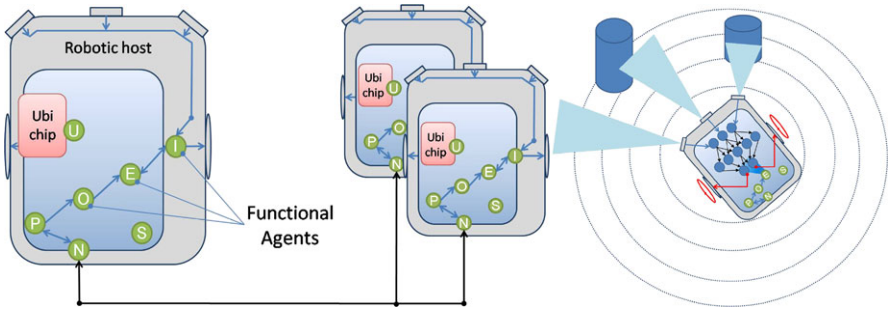

Fig. 4.18 Mapping agents onto the robots and overview of the obstacle avoidance application

Inputs of the ANN are the three values measured by the sensors, five areas have been defined for each sensor, area 0 means that an obstacle is present within a distance of less than 200 mm, subsequent areas are 200 mm deep, therefore enabling detection of objects at up to 800 mm distance. The ANN outputs are speed values sent to the two motors, with each being set as an integer value from -7 to $+7$, -7 being the maximum negative speed of a wheel (i.e. fast backward motion), and $+7$ being with the maximum positive speed of a wheel (i.e. fast forward motion). The robot can turn by applying two different speeds on the motors.

Robots are moving in an unknown environment. Each time they collide into an obstacle, a random modification of the relevant learning pattern is applied and an ANN learning phase is triggered online. The robot then notifies all its peers that this pattern shall be modified, and the modification is registered by all robots, therefore collectively speeding up convergence toward a satisfactory solution.

Our experiments show that this technique exhibits a speedup (versus a single robot) that is almost linear with the number of robots used. Furthermore, it has been observed that a convergence threshold is reached after a number of iterations, which is a function of the complexity of the environment. Once this threshold is reached, adding more obstacles in the arena retriggers learning until a new threshold is reached, thus demonstrating the adaptability potential of the proposed solution. Further experiments presented in [48] utilizing evolutionary techniques also show promising results (demonstration videos are available at http://www.lirmm.fr/~brousse/Ubibots).

5 Conclusion

Not only technology but also the rapidly widening spectrum of application domains raises a number of questions that challenge design techniques and programming methods that have been used for decades. Particularly, design-time decisions prove inadequate in a number of scenarios because of the unpredictable dimension of the environment, technology and applicative requirements that often give rise to major scalability issues.

Techniques that rely on assessing the system state and adapting at run-time appear attractive as they relieve designers of the burden of devising tradeoffs that perform reasonably well in a chosen set of likely scenarios.

This chapter stresses two important guidelines that are believed to be the cornerstone to the design of systems for the decade to come: self-adaptability and distributiveness. To this end, the presented work stressed the associated benefits of systems that comply with these two rules.

In some cases in which the scope of monitored parameters is limited, the results are remarkable as they permit to achieve significant improvements with limited overhead.

For the most ambitious techniques that rely on completely distributed decision-making based on heuristics, experiments show promising results but also highlight some limitations such as suboptimality and uncertainty. Such techniques will nevertheless be unavoidable for very-large scale systems that currently only exist in telecommunication networks.

We believe that there is no single technique that will answer every requirement but it is rather important to promote a panel of tools that will be made available to designers for devising systems tailored for a particular application area.

References

1. Complex systems and agent-oriented software engineering. In: Engineering Environment-Mediated Multi-Agent Systems. Lecture Notes in Computer Science, vol. 5049, pp. 3–16. Springer, Berlin (2008)
2. Borkar, S.: Thousand core chips: a technology perspective. In: Annual ACM IEEE Design Automation Conference, pp. 746–749 (2007)
3. Wolf, W., Jerraya, A., Martin, G.: Multiprocessor System-on-Chip (MPSoC) technology. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **27**(10), 1701–1713 (2008)
4. Freescale Semiconductor, Inc.: C-5 Network Processor Architecture Guide, 2001. Ref. manual C5NPD0-AG. http://www.freescale.com
5. Dutta, S., Jensen, R., Rieckmann, A.: Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. IEEE Des. Test Comput. **18**(5), 21–31 (2001)
6. Texas Instruments Inc.: OMAP5912 Multimedia Processor Device Overview and Architecture Reference Guide, 2006. Tech. article SPRU748C. http://www.ti.com
7. Guerrier, P., Greiner, A.: A generic architecture for on-chip packet-switched interconnections. In: DATE '00: Proceedings of the 2000 Design, Automation and Test in Europe Conference and Exhibition, pp. 250–256 (2000)
8. Dally, W.J., Towles, B.: Route packets, not wires: on-chip interconnection networks. In: DAC '01: Proceedings of the 38th Conference on Design Automation, pp. 684–689. ACM, New York (2001)
9. Benini, L., De Micheli, G.: Networks on chips: a new SoC paradigm. Computer **35**(1), 70–78 (2002)
10. Bjerregaard, T., Mahadevan, S.: A survey of research and practices of Network-on-Chip. ACM Comput. Surv. **38**(1), 1 (2006)
11. Pande, P.P., Grecu, C., Jones, M., Ivanov, A., Saleh, R.: Performance evaluation and design trade-offs for network-on-chip interconnect architectures. IEEE Trans. Comput. **54**(8), 1025–1040 (2005)
12. Bertozzi, D., Benini, L.: Xpipes: a network-on-chip architecture for gigascale systems-on-chip. IEEE Circuits Syst. Mag. **4**(2), 18–31 (2004)

13. Beigne, E., Clermidy, F., Vivet, P., Clouard, A., Renaudin, M.: An asynchronous NOC architecture providing low latency service and its multi-level design framework. In: ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 54–63. IEEE Comput. Soc., Washington (2005)
14. Pontes, J., Moreira, M., Soares, R., Calazans, N.: Hermes-GLP: A GALS network on chip router with power control techniques. In: IEEE Computer Society Annual Symposium on VLSI, ISVLSI'08, April 2008, pp. 347–352 (2008)
15. Ogras, U.Y., Marculescu, R., Choudhary, P., Marculescu, D.: Voltage-frequency island partitioning for GALS-based Networks-on-Chip. In: DAC '07: Proceedings of the 44th Annual Conference on Design Automation, pp. 110–115. ACM, New York (2007)
16. Donald, J., Martonosi, M.: Techniques for multicore thermal management: Classification and new exploration. In: ISCA '06: Proceedings of the 33rd International Symposium on Computer Architecture, pp. 78–88 (2006)
17. Beigne, E., Clermidy, F., Miermont, S., Vivet, P.: Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC. In: NOCS, pp. 129–138 (2008)
18. Beigne, E., Clermidy, F., Miermont, S., Valentian, A., Vivet, P., Barasinski, S., Blisson, F., Kohli, N., Kumar, S.: A fully integrated power supply unit for fine grain DVFS and leakage control validated on low-voltage SRAMs. In: ESSCIRC'08: Proceedings of the 34th European Solid-State Circuits Conference, Edinburgh, UK, Sept. 2008
19. Saint-Jean, N., Benoit, P., Sassatelli, G., Torres, L., Robert, M.: Application case studies on HS-scale, a mp-soc for embedded systems. In: SAMOS'07: Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece, July 2007, pp. 88–95 (2007)
20. Saint-Jean, N., Sassatelli, G., Benoit, P., Torres, L., Robert, M.: HS-scale: a hardware-software scalable mp-soc architecture for embedded systems. In: ISVLSI '07: Proceedings of the IEEE Computer Society Annual Symposium on VLSI, pp. 21–28. IEEE Comput. Soc., Washington (2007)
21. ITRS Report/Design 2009 Edition, http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Design.pdf
22. Nourani, M., Radhakrishnan, A.: Testing on-die process variation in nanometer VLSI. IEEE Des. Test Comput. **23**(6), 438–451 (2006)
23. Samaan, S.B.: Parameter variation probing technique. US Patent 6535013, 2003
24. Persun, M.: Method and apparatus for measuring relative, within-die leakage current and/or providing a temperature variation profile using a leakage inverter and ring oscillators. US Patent 7193427, 2007
25. Lee, H.-J.: Semiconductor device with speed binning test circuit and test method thereof. US Patent 7260754
26. Abuhamdeh, Z., Hannagan, B., Remmers, J., Crouch, A.L.: A production IR-drop screen on a chip. IEEE Des. Test Comput. **24**(3), 216–224 (2007)
27. Drake, A., et al.: A distributed critical path timing monitor for a 65 nm high performance microprocessor. In: ISSCC 2007, pp. 398–399 (2007)
28. Das, S., et al.: A self-tuning DVS processor using delay-error detection and correction. IEEE J. Solid-State Circuits **41**(4), 792–804 (2006)
29. Blaauw, D., et al.: Razor II: In situ error detection and correction for PVT and SER tolerance. In: ISSCC 2008, pp. 400–401 (2008)
30. Rebaud, B., Belleville, M., Beigne, E., Robert, M., Maurine, P., Azemard, N.: An innovative timing slack monitor for variation tolerant circuits. In: ICICDT'09: International Conference on IC Design & Technology (2009)
31. Rebaud, B., Belleville, M., Beigne, E., Robert, M., Maurine, P., Azemard, N.: On-chip timing slack monitoring. In: IFIP/IEEE VLSI-SoC—International Conference on Very Large Scale Integration, Florianopolis, Brazil, 12–14 October 2009, paper 56
32. Niyogi, K., Marculescu, D.: Speed and voltage selection for GALS systems based on voltage/frequency islands. In: ASP-DAC '05: Proceedings of the 2005 Conference on Asia South Pacific Design Automation, pp. 292–297. ACM, New York (2005)

33. Deniz, Z.T., Leblebici, Y., Vittoz, E.: Configurable on-line global energy optimization in multi-core embedded systems using principles of analog computation. In: IFIP 2006: International Conference on Very Large Scale Integration, Oct. 2006, pp. 379–384 (2006)
34. Deniz, Z.T., Leblebici, Y., Vittoz, E.: On-Line global energy optimization in multi-core systems using principles of analog computation. In: ESSCIRC 2006: Proceedings of the 32nd European Solid-State Circuits Conference, Sept. 2006, pp. 219–222 (2006)
35. Murali, S., Mutapcic, A., Atienza, D., Gupta, R.J., Boyd, S., De Micheli, G.: Temperature-aware processor frequency assignment for MPSoCs using convex optimization. In: CODES+ISSS '07: Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis, pp. 111–116. ACM, New York (2007)
36. Murali, S., Mutapcic, A., Atienza, D., Gupta, R.J., Boyd, S., Benini, L., De Micheli, G.: Temperature control of high-performance multi-core platforms using convex optimization. In: DATE'08: Design, Automation and Test in Europe, Munich, Germany, pp. 110–115. IEEE Comput. Soc., Los Alamitos (2008)
37. Coskun, A.K., Simunic Rosing, T.J., Whisnant, K.: Temperature aware task scheduling in MPSoCs. In: DATE '07: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1659–1664. EDA Consortium, San Jose (2007)
38. Coskun, A.K., Simunic Rosing, T.J., Whisnant, K.A., Gross, K.C.: Temperature-aware MPSoC scheduling for reducing hot spots and gradients. In: ASP-DAC '08: Proceedings of the 2008 Conference on Asia and South Pacific Design Automation, pp. 49–54. IEEE Comput. Soc., Los Alamitos (2008)
39. Ykman-Couvreur, Ch., Brockmeyer, E., Nollet, V., Marescaux, Th., Catthoor, Fr., Corporaal, H.: Design-time application exploration for MP-SoC customized run-time management. In: SOC'05: Proceedings of the International Symposium on System-on-Chip, Tampere, Finland, November 2005, pp. 66–73 (2005)
40. Ykman-Couvreur, Ch., Nollet, V., Catthoor, Fr., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In: SOC'06: Proceedings of the International Symposium on System-on-Chip, Tampere, Finland, November 2006, pp. 195–198 (2006)
41. Puschini, D., Clermidy, F., Benoit, P., Sassatelli, G., Torres, L.: A game-theoretic approach for run-time distributed optimization on MP-SoC. International Journal of Reconfigurable. Computing, ID(403086), 11 (2008)
42. Puschini, D., Clermidy, F., Benoit, P.: Procédé d'optimisation du fonctionnement d'un circuit intégré multiprocesseurs, et circuit intégré correspondant. Report No. PCT/FR2009/050581 32, France (2009)
43. Puschini, D., Clermidy, F., Benoit, P., Sassatelli, G., Torres, L.: Dynamic and distributed frequency assignment for energy and latency constrained MP-SoC. In: DATE'09: Design Automation and Test in Europe (2009)
44. Shoham, Y.: Agent oriented programming. Artif. Intell. **60**, 51–92 (1996)
45. Sanchez, E., Mange, D., Sipper, M., Tomassini, M., Perez-Uribe, A., Stauffer, A.: Phylogeny, ontogeny, and epigenesis: three sources of biological inspiration for softening hardware. In: Higuchi, T., Iwata, M., Liu, W. (eds.) Evolvable Systems: From Biology to Hardware. LNCS, vol. 1259, pp. 33–54. Springer, Berlin (1997)
46. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley, New York (2007)
47. Brousse, O., Sassatelli, G., Gil, T., Guillemenet, Y., Robert, M., Torres, L., Grize, F.: Baf: A bio-inspired agent framework for distributed pervasive applications. In: GEM'08, Las Vegas, July 2008
48. Sassatelli, G.: Bio-inspired systems: self-adaptability from chips to sensor-network architectures. In: ERSA'09, Las Vegas, July 2009