Performance Evaluation 67 (2010) 417–431

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

High level performance metrics for FPGA-based multiprocessor systems

Marta Beltrán *, Antonio Guzmán, Fernando Sevillano

Computing Department, Rey Juan Carlos University, Madrid, Spain

ARTICLE INFO

Article history:
Received 18 October 2007
Received in revised form 22 January 2009
Accepted 21 December 2009
Available online 29 December 2009

Keywords:
Efficiency
Multiprocessor on chip
Reconfigurable hardware
Robustness
Scalability

ABSTRACT

New high performance architectures combining high and low level design techniques are widely used today, and FPGA platforms offer excellent solutions for this kind of system. There are a lot of multiprocessor systems implemented on FPGAs but the performance metrics usually considered with the traditional multiprocessors have not been adapted for systems on chip yet. For these high performance systems the classic hardware metrics, such as area, cost and minimum period, are still used; but sometimes they do not provide enough information to guide users and designers in their decisions. In this paper, new definitions for high level performance metrics such as efficiency, scalability and robustness are proposed to overcome these limitations with FPGA-based multiprocessor systems.

1. Introduction

The increase in FPGA's configurable logic capacity and the decrease in their costs allow designers to incorporate varying numbers of processors, processor cores or processing units within a single FPGA design [1–4].

These new on-chip multiprocessors provide high performance and power efficient solutions for a number of applications. But there are numerous challenges that must be faced during the design and utilization of these high performance systems on chip [5].

In fact, the disappointing acceptance of this kind of system in certain application areas comes from several reasons such as the lack of a unifying model, the lack of programs portability and, above all, the lack of high level performance metrics to quantify these systems performance [6].

During the design steps of such systems, designers have to make important decisions to define efficient and flexible multiprocessors on chip. To optimize the low level system design the dominant hardware performance metrics can be used: area, cost, minimum period, etc. But a deep analysis of other system attributes should be performed too and in this case the designer has an added responsibility to make the right choices without universally accepted high level performance metrics.

Efficiency, scalability and robustness are three of the most important performance attributes of high level parallel systems [7]. The first two features quantify how the system behaves when the number of processors is increased while the third feature quantifies how the system behaves when external perturbations occur.

These three concepts are very clearly stated for the case of classic multiprocessor systems. In fact, all the metrics and methods for estimating their values which are available nowadays have been developed for this kind of system. The situation for systems on chip is quite different since this new design approach with multiple processors distributed around a single FPGA is very new.

Consequently there is a strong need for high level performance metrics for this kind of system. The starting hypothesis for the work presented in this paper is that it is possible to define high level performance metrics to quantify the efficiency, scalability and robustness of FPGA-based multiprocessors.

* Corresponding author. Tel.: +34 914888114; fax: +34 914887049.
E-mail address: marta.beltran@urjc.es (M. Beltrán).

0166-5316/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2009.12.004

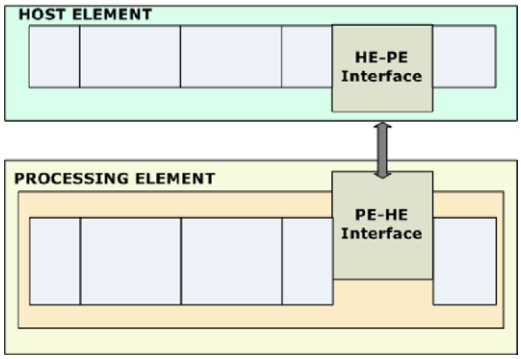

Fig. 1. Architecture model for the on-chip multiprocessor.

The main contribution of this paper is the definition of these high level performance metrics for on-chip multiprocessors adapting the definitions used with traditional high level systems. Furthermore, new procedures for performance evaluation of on-chip multiprocessors based on these new metrics have been proposed. These two contributions, the metrics definition and the performance analysis procedures, allow designers and users to have a complete characterization of their systems performance from a high level point of view and to optimize their designs.

The rest of the paper is organized as follows. Section 2 discusses some general models and basic metrics necessary to understand the rest of the paper. Sections 3–5 present the proposed definitions for the efficiency, scalability and robustness performance metrics for on-chip multiprocessors. Section 6 shows the experimental results obtained with one specific design to validate the proposed metrics and to illustrate how to use them. And finally Section 7 presents conclusions and suggestions for future work.

2. Previous models and definitions

The key contribution of this research is the proposal of high level performance metrics for quantifying the behavior of multiprocessors implemented on FPGA systems. But to derive general metrics it is necessary to develop a conceptual architecture model that can be applied in a wide variety of contexts.

This model is presented in Fig. 1. Obviously it is a high level model that does not care about the low level implementation details: it is a functional model to describe the general architecture considered in this work to characterize the on-chip multiprocessors performance. This on-chip multiprocessor is composed of two main modules or subsystems:

- *Processing Element (PE)*: This first module is in fact the multiprocessor system. It is implemented on a single FPGA composed of a number of general purpose processors, processor cores or application specific processing units. The processing element can be homogeneous or heterogeneous depending on the features of all these processors. And the memory model may be shared, shared-distributed or distributed depending on the design approach.
- *Host Element (HE)*: This second module is optional, therefore, it may be present or not in the on-chip multiprocessor design. If it is present, it can be placed on a software module (external host) or on a hardware one as the processing element (on-chip host). This module would be responsible for the configuration and reconfiguration of the processing element and for the task allocation in the available resources. Furthermore, in certain cases it can execute the sequential part of the application and use the processing element as a coprocessor.

In this kind of system some basic metrics can be defined to begin with the high level performance quantification:

- *Cost (C):* The cost of an on-chip multiprocessor design can be defined:

$$C = \frac{1}{K} \cdot \sum_{j=1}^{K} \frac{d_j}{D_j} \tag{1}$$

where K is the number of elemental constituent resources of the FPGA (lookup tables, flip-flops, registers, IO blocks, etc.), d_j is the number of consumed elements of the j_{th} type and D_k is the total number of elements of this type available in the FPGA. This cost metric is dimensionless and can vary from 0 to 1. But a specific design can use up completely one of the elemental resources of the FPGA, being therefore impossible to reach a cost equal to 1. This is the maximum cost only when the synthesis is ideal and all the FPGA resources are consumed in the same degree. To solve this problem a relative cost can be defined taking into account the real maximum cost value.

- *Relative Cost* (C_{rel}): The relative cost of an on-chip multiprocessor design can be defined:

$$C_{rel} = \frac{C}{C_{max}} \tag{2}$$

where C is the real cost defined in Eq. (1) and C_{max} is the maximum cost of a certain design on the FPGA considering that there is a first consumed elemental resource (and therefore, $C_{max} < 1$).

- *Speedup (S):* While there is a general agreement that the speedup in a parallel system is the ratio of serial response time to parallel execution time, there are diverse definitions of these times in a system like the one considered in this work, and this leads to different speedup definitions [8]:
  (1) *Relative speedup:* The ratio of the application response time when it is entirely executed on the Host Element to the application response time when the Host Element sends the application or a part of it to the Processing Element with only one processor:

$$S_R = \frac{t_{HE}}{t_{PE:1}}. \tag{3}$$

  If there is no Host Element in the system, this speedup can be defined as the ratio of the sequential application response time in its software version (running on a traditional general purpose processor) to the application response time when it is executed on the Processing Element with only one processor:

$$S_R = \frac{t_{software}}{t_{PE:1}}. \tag{4}$$

  (2) *Real speedup:* The ratio of the application response time when it is executed on the Host Element with its best instance (fully optimized) to the application response time when the Host Element sends the application or a part of it to the Processing Element with N processors:

$$S_{real} = \frac{t_{HE_{best}}}{t_{PE:N}}. \tag{5}$$

  If there is no Host Element, this speedup can be defined as the ratio of the sequential application response time in its best software instance (running on a traditional general purpose processor and fully optimized) to the application response time when it is executed on the Processing Element with N processors:

$$S_{real} = \frac{t_{software_{best}}}{t_{PE:N}}. \tag{6}$$

  (3) *Absolute speedup:* The ratio of the application response time when it is executed in the Processing Element with only one processor to the response time in the Processing Element with N processors:

$$S_{ABS} = \frac{t_{PE:1}}{t_{PE:N}}. \tag{7}$$

Analyzing the proposed definitions of the speedup it is possible to see that the relative speedup quantifies the benefits obtained from the use of the on-chip multiprocessor instead of the software version running on a general purpose processor or the application version on the Host Element. But only sequential executions are being compared with this definition, because only one processor is considered in the Processing Element. The possible code parallelization is not being considered yet.

To know how much faster an application runs on the multiprocessor taking advantage of the parallel fraction of the application, the real speedup has been defined. Therefore, this metric compares the application performance in its best instance, running on a general purpose processor or on the Host Element, with its performance on the multiprocessor with N processors or processing units. And finally, the last definition for the absolute speedup can help the designer to decide the optimum number of processors, because it compares the multiprocessor design with one processor and with N processors. This comparison can be very interesting, because, as in other parallel systems, the performance improvement to be gained from increasing the number of system processors is limited by the parallel fraction of the application and by the design bottlenecks.

3. Efficiency metric

Once a system model is proposed and the basic performance metrics have been defined, the first high level performance metric for the on-chip multiprocessors can be discussed.

Efficiency (ε) for parallel systems is consistently defined as the ratio of the speedup to the number of system processors [8]. Therefore, with the traditional definition, it would be:

$$\varepsilon = \frac{S_{ABS}}{N} \tag{8}$$

where S_{ABS} is the absolute speedup and N is the number of processors. The absolute speedup has been selected for the efficiency computation because it is the metric used to compare different versions of the on-chip multiprocessor with a different number of processors.

But for an on-chip multiprocessor two different designs may have the same S_{ABS} with the same number of processors but very different cost values. This must be considered in the efficiency metric, because the design with the lower cost would be the more efficient in this case.

The following definition is proposed for the on-chip multiprocessor efficiency:

$$\varepsilon = \frac{S_{ABS}}{N} \cdot \frac{1}{1+C} \tag{9}$$

where C is the design cost defined in Eq. (1). With this definition, the efficiency values can be between 0 and 1, and the cost term modifies the traditional efficiency definition multiplying it by 1 in the best case and by 0.5 in the worst case. The factor $1/(1+C)$ has been used instead of $1/C$ because we wanted a normalized efficiency with a maximum value of 1. With the $1/(1+C)$ weight, the minimum cost implies the maximum efficiency S_{ABS}/N and not an ∞ value.

As a result of this discussion it can be seen that the efficiency metric proposed for on-chip multiprocessors is the ratio of the absolute speedup to the number of system processors but considering the design cost. Therefore, it quantifies the system ability to take advantage of all the available resources and facilities in the FPGA design platform.

4. Scalability metric

Rather than having a definition of scalability which is universally accepted, it is possible to find a number of scalability models that have been proposed during the past years.

The main difference between all these models is in the selection of the performance metric used to characterize the system's behavior. Intuitively, a parallel system is scalable if its performance continues improving as the system size is increased [7,9]. But, how this performance is quantified? A variety of metrics have been proposed: speedup [10,11], efficiency [12], latency [13], power [14], average speed [15].

In this research the P-scalability definition [14], based on the power performance metric, has been chosen as the starting point due to a number of reasons:

- The P-scalability allows one to use as a performance metric a general aspect called Quality of Service (QoS) which can be selected depending on the evaluated system. It can be used any measure of the goodness of a service in the system and this is very important to adapt the scalability metric to the different on-chip multiprocessors, used in a wide variety of applications and implemented on a wide variety of platforms.
- It allows taking into consideration the design cost too, very important in hardware designs.
- This metric can be used with homogeneous and heterogeneous multiprocessors.
- And finally, the strategy for scaling up the system is not only based on adding more processors, it can be much more complex considering another system resources such as memory banks or storage resources.

In [14], the scalability metric from configuration 1 to configuration 2 is defined:

$$\psi_P = \frac{\frac{P_2}{C_2}}{\frac{P_1}{C_1}} \tag{10}$$

where P is the system power and C is the system cost. Therefore, with the P-scalability definition a system is scalable from configuration 1 to configuration 2 if the additional cost introduced by the system scaling is worthwhile considering the system power gain.

The system cost can be quantified with the definition given in the previous section (Eq. (1)). On the other hand the power definition P selected for the scalability metric in this work is:

$$P = \lambda \cdot f\left(QoS, \overline{QoS}\right) \tag{11}$$

where λ is the system throughput, QoS is the quality of service metric chosen for the evaluated design and $\overline{QoS}$ is the expected or desired quality of service. It has to be pointed that this last value is not a mathematical expectation or average; it is constant specified by the designer or user for the system.

The two general QoS metrics proposed in this work are the application response time (T) and the design efficiency (ε):

- The function f proposed in [14] for the response time metric can be easily used with on-chip multiprocessors:

$$f\left(T, \overline{T}\right) = \frac{1}{1+\frac{T}{\overline{T}}}. \tag{12}$$

With this definition, the system power is high when its throughput is high and when the T value is close to its expected value, $\overline{T}$.

And the scalability definition with this QoS metric is:

$$\psi_P(T) = \frac{\lambda_2 \cdot \overline{T_2} \cdot C_1 \cdot (T_1 + \overline{T_1})}{\lambda_1 \cdot \overline{T_1} \cdot C_2 \cdot (T_2 + \overline{T_2})}. \tag{13}$$

- On the other hand, for the efficiency metric, a new f function has been proposed in this work. Since a high response time value and a high efficiency value have opposite effects on the power figure, this function for the efficiency is:

$$f\left(\varepsilon, \overline{\varepsilon}\right) = \frac{1}{1+\frac{\overline{\varepsilon}}{\varepsilon}}. \tag{14}$$

Therefore, the scalability definition with this QoS metric is now:

$$\psi_P(\varepsilon) = \frac{\lambda_2 \cdot \varepsilon_2 \cdot C_1 \cdot (\varepsilon_1 + \overline{\varepsilon_1})}{\lambda_1 \cdot \varepsilon_1 \cdot C_2 \cdot (\varepsilon_2 + \overline{\varepsilon_2})}. \tag{15}$$

The two proposed quality of service figures are the most widely used, but if another measure of this quality is needed for a specific application or design, only a new f function have to be defined by the designer or the user to compute the system power with Eq. (11).

The scalability quantification has a number of applications and it is usually taken into consideration by designers and users. But another key idea in describing systems scalability is to perform an analysis on a complete system scaling path controlled by a selected scaling factor from a reference configuration rather on taking into account only the scalability between two configurations. As it has been mentioned before, with the P-scalability the scaling factor is completely general, it can be the traditional number of processors or the number of memory banks, the communications bandwidth, etc.

At each scale factor the system can be optimally tuned by adjusting all its scalability enablers, which are the parameters that can improve the system performance after its scaling. Then, the scalability at this scale factor is computed comparing the power at this configuration with the power of the reference case.

For example, if the selected scaling factor for the analysis is the number of processors, N, the proposed scalability analysis is performed following these steps:

(1) Obtain the maximum cost value (C_{max}) for the evaluated design estimating the maximum number of processors (N_{max}) in the available hardware resources.
(2) Define the scaling strategy, from the reference case ($N = 1$) to the maximum system size (N_{max}). The followed scaling path can be:
   - *Fixed:* When the scaling is performed only increasing the number of system processors.
   - *Variable:* When the scaling takes advantage of the scalability enablers, tuning the system as best as possible for each N value.
(3) Select the quality of service metric interesting for the considered application and design, and the corresponding f function.
(4) Measure C, λ and QoS for all the N values.
(5) Compute the P-scalability for each N value comparing it with the reference case, $N = 1$:

$$\psi_P = \frac{\lambda_N \cdot f_N(QoS, \overline{QoS}) \cdot C_1}{\lambda_1 \cdot f_1(QoS, \overline{QoS}) \cdot C_N} \ \forall\, N. \tag{16}$$

This kind of analysis is very important in on-chip multiprocessors, because selecting an adequate scaling strategy, important information can be obtained to identify the design bottlenecks, to find the optimum tuning of the system or to decide the optimum number of processors in the design stage.

5. Robustness metric

In this section two robustness metrics (ρ_{QoS} and ρ_P) are proposed to quantify the degradation of the system performance when perturbations occur, that is, when unpredictable state changes take place in the system. Two different robustness metrics have been defined because they are based on two different performance metrics to characterize the system's behavior. The first is the quality of service (QoS), and the second is the power (P), both defined in the previous section. Therefore the ρ_{QoS} metric considers the degradation of the QoS achievement when perturbations exist and, on the other hand, the ρ_P takes into account the P degradation with these perturbations.

These two robustness metrics depend on both, the system architecture and the task allocation performed in the different system processors. The application executed on the on-chip multiprocessor must be divided into tasks or processes or some kind of work set to take advantage of its parallel fraction on the available system resources. And these tasks can be allocated to the system processors in different ways. A task assignment is defined to be robust with respect to a system performance feature against some specific perturbations if the degradation in this feature is limited when the perturbations occur [16,17].

5.1. QoS-robustness

For the considered system and performance feature (QoS), a design is robust when:

$$A \geq \tau_R \cdot \overline{A} \tag{17}$$

where A denotes the real QoS achievement, $\overline{A}$ denotes its expected achievement and τ_R is the robustness tolerance defined by the system designer or user.

Therefore, the system is robust if it obtains a certain degree of the expected QoS, given by τ_R, despite all the possible perturbations in the system.

The achievement value (A) has to be defined to quantify the degree of the QoS achievement, therefore:

$$A = \frac{W_C}{W} \tag{18}$$

where W_C is the number of tasks executed on the system processors accomplishing with the expected QoS and W is the total number of tasks composing the application.

Siegel proposed the FePIA procedure [16,17] to quantify the system-tasks allocation combination robustness. This method, very frequently used for analyzing systems robustness, constitutes the starting point which will be extended in

this paper to quantify the robustness of on-chip multiprocessors. The steps of this procedure having an architecture such as the one studied in this work are the following:

(1) *Performance features* (Ψ): The selected performance feature is:

$$\Psi = \{\psi_j\} = \{w_j | 1 \leq j \leq N\}$$

where w_j is the number of tasks allocated to the j_{th} system processor that finally achieve the expected *QoS* and N is the number of system nodes. There is an equivalent magnitude, $\overline{w_j}$, denoting the total number of tasks allocated to this processor that are supposed to achieve the expected *QoS* if there are not system perturbations to avoid it.

(2) *Perturbation parameters* (Π): The parameters whose values can impact the proposed performance feature must be identified. System perturbations may modify the system state preventing tasks to accomplish with the expected *QoS*. The perturbation parameter is then:

$$\Pi = L$$

where L denotes the total number of tasks that are allocated to system processors to achieve the expected *QoS* but finally are not able to do it due to perturbations. Therefore, $\overline{L} = 0$, because the system expects not to fail any prediction about the achievement of the *QoS* requirements.

(3) *Impact of the perturbation parameter on the system performance feature:* With the previous definitions, this relation can be summarized with the following expressions:

$$W_C = \sum_{j=1}^{N} w_j$$

$$\overline{W_C} = \sum_{j=1}^{N} \overline{w_j}$$

$$W_C = \overline{W_C} - L.$$

(4) *Robustness metric:* It is the smallest variation in the value of the perturbation parameter that would cause the performance feature to violate its acceptable variation, i.e. a violation of the condition expressed in Eq. (17). And the system-task allocation is not robust if this condition is not fulfilled. The degree of the *QoS* achievement can be related to the performance feature and to the perturbation parameter by:

$$A = \frac{W_C}{W} = \frac{\sum_{j=1}^{N} w_j}{W} = \frac{\overline{W_C} - L}{W}.$$

And, therefore, the robustness radius is:

$$\rho_{QoS}(\Psi, \Pi) = \min_{L:A(L) \geq \tau_R \cdot \overline{A} \wedge \exists L+1:A(L+1) < \tau_R \cdot \overline{A}} (L). \tag{19}$$

Therefore, the robustness radius is a discrete magnitude: the largest number of processes executed on the system that can finally fail in achieving the expected *QoS* without causing unacceptable performance degradations in terms of the robustness tolerance given by the system designer or user.

This procedure allows an algorithmic evaluation of the system robustness radius. And this discrete radius can be used to decide about system design, configuration and reconfiguration. The robustness metric warns the designer or user when the system-task allocation combination is not able to obtain the expected performance in terms of expected *QoS* achievement (A). This warning can be obtained in two different situations:

- Before executing the application, if there is a priori information about it and about the system.
- After executing the application, once this information has been collected, to improve the performance for later executions.

Anyway, if the robustness degree (ρ_{QoS}) is not enough for the designer or user requirements, they can:

- Try to avoid the perturbation causes.
- Relax their performance requirements, modifying the robustness tolerance, τ_R.
- Improve the system tuning or the task allocation mechanism.
- Reconfigure the system, if it is possible, to improve its design.

5.2. P-robustness

For this robustness metric the performance feature whose degradation is quantified is the system power P defined in the previous section. In this case, the system is robust when:

$$P \geq \tau_R \cdot \overline{P} \tag{20}$$

where P denotes the real system power (Eq. (11)), $\overline{P}$ denotes its expected value and τ_R is the robustness tolerance defined by the system designer or user again.

Therefore, the system is robust if it obtains a certain degree of the expected system power, given by τ_R, despite all the possible perturbations in the system.

The steps to obtain the system robustness in this case are the following :

1. *Performance features* (Ψ): The selected performance feature is the throughput of each system processor (number of finished tasks per time unit):

$$\Psi = \{\psi_j\} = \{\lambda_j | 1 \leq j \leq N\}$$

where λ_j is the throughput of the j_{th} system processor, $\overline{\lambda_j}$ is its expected value and N is the number of system processors

2. *Perturbation Parameters* (Π): The parameters whose values can impact the proposed performance feature are the processes response times in this case. Therefore the perturbation parameters are continuous in this case:

$$\Pi = \{\pi_i^j\} = \{t_i^j | 1 \leq i \leq q_j,\ 1 \leq j \leq N\}$$

where q_j denotes the number of processes executed on the j_{th} processor and t_i^j denotes the response time of the i_{th} process on the node it has been allocated, the j_{th} node. Again $\overline{t_i^j}$ is the expected response time for this task.

3. *Impact of the perturbation parameter on the system performance feature:* With the previous definitions, this relation can be summarized with the performance factor, λ_j (the throughput of the j_{th} system processor) computation. One assumption is made: the processors composing the Processing Element are not capable of executing processes in parallel, therefore, the processes are executed sequentially and one process cannot begin its execution until the previous has finished.

$$\lambda_j = \frac{number\ of\ executed\ tasks}{execution\ time} = \frac{q_j}{\sum_{i=1}^{q_j} t_i^j + T_C + T_W}$$

where T_C and T_W denote the communication and waiting times for the j_{th} system processor respectively. The first time is used by the processor to communicate with other processes and with the Host Element. The second time has to be considered because these communications, the memory accesses, the context switches etc. always imply waiting times in which the processor is idle. In the rest of this research these times will be considered negligible compared with the total execution time of the q_j processes allocated to the j_{th} node.

4. *Robustness metric:* Again, it is the smallest variation in the value of the perturbation parameter that would cause the performance feature to violate its acceptable variation, i.e. a violation of the condition expressed in Eq. (20).

$$P \geq \tau_R \cdot \overline{P}.$$

Therefore, given the system power definition:

$$\lambda \geq \tau_R \cdot \overline{P} \cdot f^{-1}(QoS, \overline{QoS}) \geq \tau_R \cdot \overline{P} \cdot \overline{f^{-1}(QoS, \overline{QoS})} = \tau_R \cdot \overline{\lambda}.$$

From this point it is necessary to study two different situations: homogeneous and heterogeneous multiprocessors, depending on if all the system processors have the same hardware features or not.

- *Case 1: Homogeneous multiprocessor*

  Taking into account the traditional throughput definition:

$$\frac{W}{T} \geq \tau_R \cdot \frac{W}{\overline{T}}$$
$$\frac{1}{T} \geq \tau_R \cdot \frac{1}{\overline{T}}$$
$$T < \frac{1}{\tau_R} \cdot \overline{T}$$

  where T is the application response time. Since all the system processors work in parallel, the total application response time can be obtained with the following expression:

$$T = max(T_j) = max\left(\sum_{i=1}^{q_j} t_i^j\right).$$

  And following an equivalent reasoning, the expected response time for the application will be:

$$\overline{T} = max(\overline{T_j}) = \overline{q_j} \cdot \overline{t} = \left\lceil \frac{W}{N} \right\rceil \cdot \overline{t}.$$

  Because in the expected situation all the system processors must execute the same number of tasks (it is a homogeneous system) and all these tasks have the same expected response time $\overline{t}$.

Therefore:

$$max(T_j) < \frac{1}{\tau_R} \cdot \left\lceil \frac{W}{N} \right\rceil \cdot \bar{t}.$$

And if the processor with the maximum T_j in the system fulfill this condition, all the processors in the system must do it:

$$T_j < \left\lceil \frac{W}{N} \right\rceil \cdot \frac{\bar{t}}{\tau_R}.$$

And the robustness radius for the j_{th} processor can be obtained with:

$$r_j(\Psi, \Pi) = \min_{t:T_j=\lceil \frac{W}{N} \rceil \cdot \frac{\bar{t}}{\tau_R}} \|t - \bar{t}\|_2 \tag{21}$$

Hence, for the entire system the robustness radius for the P-robustness will be:

$$\rho_P(\Psi, \Pi) = min(r_j(\Psi, \Pi)). \tag{22}$$

Eq. (21) is the Euclidean distance between any vector composed of the real task response times and the vector composed of the expected response times. It can be interpreted as the distance from a point to a hyperplane, therefore, using the point to plane expression, this equation can be reduced to:

$$\rho_P(\Psi, \Pi) = min(r_j(\Psi, \Pi)) = min\left(\frac{\lceil \frac{W}{N} \rceil \cdot \frac{\bar{t}}{\tau_R} - T_j(\bar{t})}{\sqrt{q_j}} \right). \tag{23}$$

With:

$$T_j(\bar{t}) = q_j \cdot \bar{t}.$$

Because it is expected that all the processes allocated to the j_{th} system processor have the expected response time $\bar{t}$.

To determine the robustness radius of the system it is necessary to compute individually the robustness radius for all the system processors, because it is the minimum of all these values. And it is a continuous radius with time units, because in this case it is the maximum distance between any vector of the real tasks response times and the vector of the expected response times which does not cause an unacceptable degradation of the system power even when perturbations occur.

- *Case 2: Heterogeneous multiprocessor* If the multiprocessor is composed of different kinds of processors, the system will have a robustness radius for each kind of processor. Therefore the system designer or user has to define a robustness tolerance for each kind of processor, τ_R^k (related to the different kinds of tasks executed on the system).

Knowing the expected response time for the kth type of task ($\overline{t^k}$), the number of tasks executed on this kind of processor (W^k) and the number of processors of this kind (N^k), the robustness radius for the k_{th} type can be obtained with the same expressions than in the homogeneous multiprocessors:

$$\rho_P^k(\Psi, \Pi) = min(r_j(\Psi, \Pi)) | j \in k. \tag{24}$$

Therefore:

$$\rho_P^k(\Psi, \Pi) = min\left(\frac{\lceil \frac{W^k}{N^k} \rceil \cdot \frac{\overline{t^k}}{\tau_R^k} - T_j(\overline{t^k})}{\sqrt{q_j}} \right). \tag{25}$$

With:

$$T_j(\overline{t^k}) = q_j \cdot \overline{t^k}.$$

6. Experimental results

It is very important to know how a system designer or user must apply the proposed performance metrics and what usefulness they hold. In this section some experimental results with a specific FPGA-based multiprocessor are presented to illustrate the proposed metrics utilization and to validate their definitions.

6.1. Experimental setup

To perform the proposed experiments the HIPAOC (HIgh Performance Architecture On Chip) system has been used. This design has been performed with a high level description language, Handel-C [18], coherent with the high level design philosophy pursued in this research.

The HIPAOC system is based on a hardware/software codesign approach (Fig. 2), being therefore a combination of an external host (software block) and a configurable multiprocessor on a single FPGA working as a coprocessor (hardware block). Therefore, in this architecture a Host Element (HE) and a Processing Element (PE) can be distinguished. The HE is

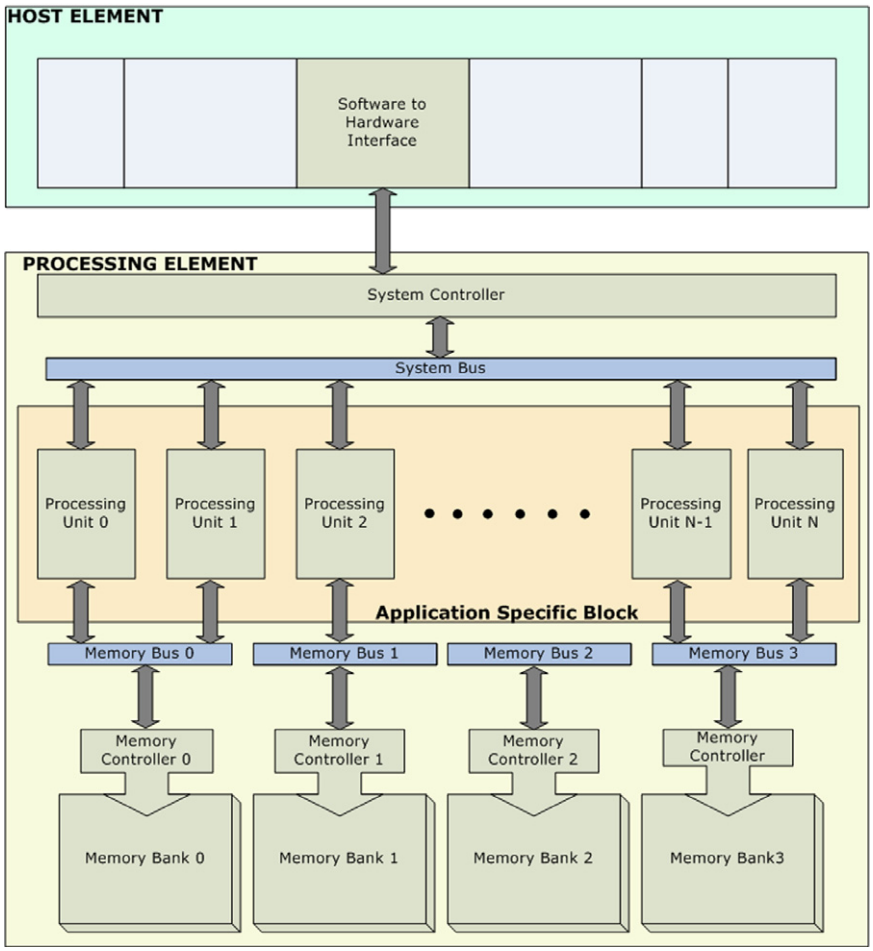

Fig. 2. An example of on-chip multiprocessor: HIPAOC architecture.

running on a general purpose processor and the PE, implemented on a FPGA Platform, is composed of a number of application specific processing units (PU). This system is a general architecture in which the designer or user only has to design the application-dependent modules: the Host Element and on the Processing Element, only the PUs.

This system can be mapped to the model proposed in Section 2 in the following way:

- *Processing Element (PE):* The configurable multiprocessor can be decomposed in the following modules:
  - *System Controller (SC):* This module is the element which controls the multiprocessor operation. It is responsible for three main tasks: managing the communications with the Host Element, distributing the work between the PUs and monitoring the PU's state. The SC monitors the PUs' state and when one of these PUs produces an event, this controller informs the HE using a simple channel. There are two kinds of events: when a PU has finished its work and the results have to be gathered and when a PU is free and it is able to begin a new work. The first kind of event is managed in the HE by a thread responsible for gathering the hardware results and the second one is managed by a thread responsible for sending new tasks to the multiprocessor. When this last kind of event occurs, the communication with the HE has been implemented using a blocking interconnection. The HE uses this interconnection to send tasks to the SC. When a new task arrives to the hardware, the SC decides which PU is going to execute it, and due to the blocking connection, the HE only accesses the hardware resources when the SC tells it how to perform this access. For example, the SC specifies what memory space has to be read from or written to by the HE depending on the PU selected to execute the task and on the memory model.
  - *SC/PU Interconnection:* This interconnection is very easy, only for synchronization purposes, because there is not any information to be transferred between the SC and the PUs. Therefore it is composed only of control lines. These synchronization lines have been implemented with two blocking channels, *Start* and *Event*. The first tells the PUs when to begin to work due to the allocation of a new task. And the second, in the opposite direction, informs the SC about the events produced by the PUs.
  - *Processing Units (PU):* This is the only reconfigurable module in the HIPAOC design because it is application-dependent. The HE decides the number of PUs needed in the system and reconfigures the architecture of these modules to perform

the desired functionality. There are two main restrictions for the implementation of these modules. The first, they must be homogeneous, in other words, the design can only contain multiple instances of the same PU, all executing the same kind of task. And the second restriction, the PU's description must be compatible with the Handel-C description of the rest of the HIPAOC system. Although the PU's design is application-dependent, these aspects must be common to all the designs. Since the design of the rest of system modules is completely general, the other common aspects that must be considered are the interconnection and synchronization protocols with these modules. First, each PU must contain a small ROM to store a unique processing unit identifier (PUID) needed for all the HIPAOC operations. And second, the following protocol must be used. When the SC activates the *Start* channel of one PU, this PU begins to work on its assigned memory space. During the execution of the new task, all the memory accesses are performed using a memory controller (MC), depending on the selected memory model. And when the PU finishes its work, it notifies this event to the SC using the *Event* channel. Then, when the SC informs the host of this change and the host gathers the PU results, the PU uses again the *Event* channel to inform the SC of its availability.

  - *PU/MC Interconnection:* An efficient interconnection architecture has been implemented for these modules combining both blocking and non-blocking channels. The non-blocking interconnection has been implemented with a bus composed of three shared registers. These links allow each PU to ask for memory accesses to the correspondent MC depending on the selected memory model. The *Address* and *Data* registers are used to perform the memory access. The *Check* register has control functions and allows a PU to know when its requested memory access has been performed. In addition, two blocking channels have been added to perform this protocol and to ensure synchronization, the *End* and *Ack* channels. The first one is activated by a MC when it has finished a memory access and it has been notified to the PU's writing in the *Check* register the PUID of the requesting PU. The second is used by the PU which sees its PUID in the *Check* register to notify the MC that the memory access has been checked.
  - *Memory Controllers (MC):* The number of these modules and their organization determines the memory model of the HIPAOC system. There are two alternatives: shared and distributed memory.
- *Host Element (HE):* The host design cannot be general because it has a strong relationship with the application and the selected FPGA platform. However, some general considerations can be made, because the software module of the system always perform two stages or steps: a configuration/reconfiguration stage for the Processing Element and the execution stage. In the first stage, the host initially configures or reconfigures the HIPAOC system. The parameters that have to be fixed in this first stage are: number and architecture of the PUs and memory model (shared or distributed), and therefore, the number of memory controllers. And, during the execution stage, the host has to generate the appropriate tasks for the multiprocessor processing units, sending them to the SC and gathering the results when they are finished. To perform these tasks three different threads have been implemented: one to execute the serial part of the application on software, one to send tasks to the Processing Element and the last one to gather the results generated by the hardware.

6.2. Results and discussion

Two main objectives were pursued in the experiments presented in this section:

- To validate the proposed definitions for the high level performance metrics and to demonstrate that they are accurate enough to quantify the performance of on-chip multiprocessors.
- To exemplify how to perform the proposed performance analysis procedures on a configurable multiprocessor implemented on a single FPGA. This kind of reconfigurable system has more opportunities to use the proposed metrics as optimization criteria.

6.2.1. Scalability experiment

The first performed experiment uses a very simple design based on the HIPAOC system to illustrate the utility of the scalability analysis proposed in Section 4.

The selected application performs 128 matrix products. With this design the Host Element sends to the PU's in the Processing Element homogeneous tasks, each one to perform a 64x64 matrix product. The system has been designed with a shared memory approach and it has been implemented on a Xilinx4000 FPGA [19].

To perform the scalability analysis, the two quality of service metrics proposed in this work have been considered, the application response time (T) and the design efficiency (ε).

Applying the proposed methodology using the number of processing units as the scaling factor:

(1) The maximum number of processing units in the selected platform for this design is $N_{max} = 64$, and the maximum cost value is $C_{max} = 0.29$ because the lookup tables were the first consumed element in the selected FPGA platform with the applied design synthesis.
(2) With this information the scaling strategy can be defined. In this case the following values are considered in the scalability analysis : $N = 1, 2, 3, 4, 6, 8, 12, 16, 32, 64$. Two scaling paths are proposed:
   - *Fixed:* The number of processing units is increased without modifying the rest of the system design.
   - *Variable:* The number of processing units is increased using the scalability enablers in each new system configuration to optimize its performance. In this case there is clearly one scalability enabler: the memory model. For medium and large design sizes, the shared memory is an important bottleneck, therefore above 8 processing units the memory model is changed to a distributed one.

Table 1
Scalability analysis with fixed scaling path.

N	C	T (μs)	S_{ABS}	λ ($task$/s)	ε	$\psi(T)$	$\psi(\varepsilon)$
1	0.234	20805160	1.000	6.152	0.810	1.000	1.000
2	0.236	19113727	1.088	6.697	0.440	1.381	1.402
3	0.237	14473180	1.437	8.844	0.387	2.460	1.927
4	0.238	13514286	1.539	9.471	0.311	2.920	2.198
5	0.239	13152176	1.582	9.732	0.255	3.150	2.363
6	0.240	13176801	1.579	9.714	0.212	3.200	2.453
7	0.241	13794328	1.508	9.279	0.174	2.970	2.428
8	0.242	15024717	1.385	8.519	0.139	2.544	2.304
10	0.244	17999056	1.156	7.111	0.093	1.804	2.009
12	0.246	23932086	0.869	5.348	0.058	1.031	1.560
16	0.249	27240111	0.764	4.699	0.038	0.795	1.389
32	0.263	40800519	0.510	3.137	0.013	0.341	0.916
64	0.288	60858393	0.342	2.103	0.004	0.140	0.577

Table 2
Scalability analysis with variable scaling path and $QoS = T$.

N	C	T (μs)	S_{ABS}	λ ($task$/s)	ε	$\psi(T)$
1	0.234	20805160	1.000	6.152	0.810	1.000
2	0.160	6566954	1.375	19.492	0.593	1.718
3	0.198	4851851	1.860	26.382	0.518	2.652
4	0.236	3760459	2.400	34.038	0.485	3.749
5	0.237	4027552	2.241	31.781	0.362	3.621
6	0.238	3689979	2.446	34.689	0.329	4.431
7	0.239	3720330	2.426	34.406	0.280	4.556
8	0.240	3608789	2.501	35.469	0.252	4.959
10	0.242	3674022	2.457	34.839	0.198	5.019
12	0.243	3522863	2.562	36.334	0.172	5.583
16	0.247	3945642	2.288	32.441	0.115	4.685
32	0.261	5186781	1.740	24.678	0.043	2.803
64	0.288	5687966	1.587	22.504	0.019	2.182

Table 3
Scalability analysis with variable scaling path and $QoS = \varepsilon$.

N	C	T (μs)	S_{ABS}	λ ($task$/s)	ε	$\psi(\varepsilon)$
1	0.234	20805160	1.000	6.152	0.810	1.000
2	0.236	19113727	1.088	6.697	0.440	1.402
3	0.237	14473180	1.437	8.844	0.387	1.927
4	0.238	13514286	1.539	9.471	0.311	2.198
5	0.239	13152176	1.582	9.732	0.255	2.363
6	0.240	13176801	1.579	9.714	0.212	2.453
7	0.241	13794328	1.508	9.279	0.174	2.428
8	0.242	15024717	1.385	8.519	0.139	2.304
10	0.242	3674022	2.457	34.839	0.198	2.192
12	0.243	3522863	2.562	36.334	0.172	2.336
16	0.247	3945642	2.288	32.441	0.115	2.189
32	0.261	5186781	1.740	24.678	0.043	1.728
64	0.288	5687966	1.587	22.504	0.019	0.596

(3) The values of C, λ and T and ε are measured for each N value in the two proposed scaling paths.
(4) The P-scalability value is computed for all the possible system configurations using the system with $N = 1$ as the reference case and the f functions proposed in Section 4:

$$\psi_P = \frac{\lambda_N \cdot f_N(QoS, \overline{QoS}) \cdot C_1}{\lambda_1 \cdot f_1(QoS, \overline{QoS}) \cdot C_N} \; \forall N.$$

The results obtained with this analysis are shown in Table 1 for the fixed scaling path and in Tables 2 and 3 for the variable scaling path. All these results are summarized too in Figs. 3 and 4.

The results with the fixed scaling path show how the design cost has a constant increment as the number of processing units is increased in the system. But the maximum theoretical value of 1 is never reached because one of the elemental constituents of the FPGA is consumed after reaching this value.

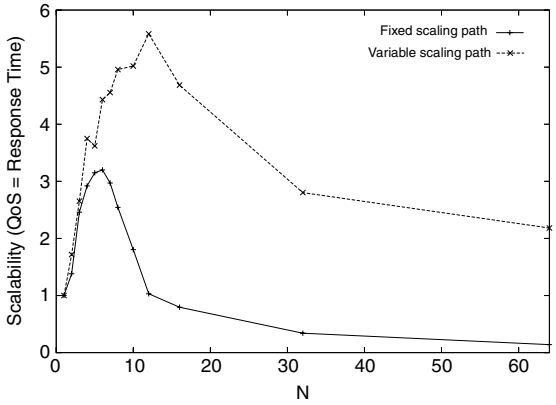

Fig. 3. Scalability Analysis with $QoS = T$.

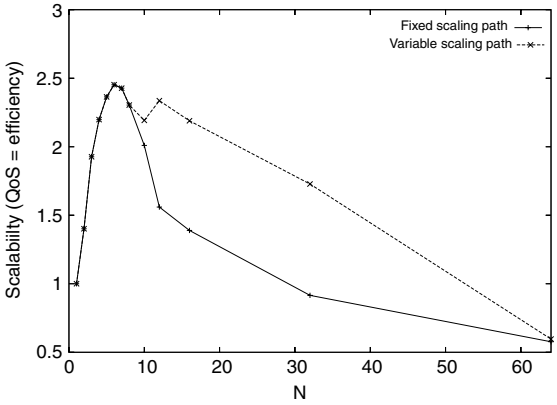

Fig. 4. Scalability analysis with $QoS = \varepsilon$.

On the other hand, it has to be pointed that the application response time initially decreases with the system scaling but for N values above 8 this time worsens due to some bottleneck in the design. With the fixed scaling path this bottleneck cannot be avoided, therefore, the scalability for this design without using the scalability enablers reaches its maximum value below $N = 8$. More precisely, for the quality of service metric based on the response time, the maximum value is $\psi(T) = 3.2$ with $N = 6$. And for the quality of service metric based on the efficiency, the maximum value is $\psi(\varepsilon) = 2.45$ with $N = 6$ too.

These results show that in this design with the selected FPGA platform it is easier to improve the application response time scaling the system than improving the efficiency. Anyway in both cases, the design bottlenecks make the $N = 6$ value, the optimum number of processing units if the design cannot be changed. On the other hand, the results for the variable scaling path strategy show that the main bottleneck in the system design is the shared memory model. That is the reason why using the memory model as the scalability enabler, the results significantly improve. This can be done changing the system memory model above $N = 8$, having a distributed model for all the N values above this.

It can be seen that with this strategy, the maximum scalability value for the quality of service based on the application response time improves, this maximum value appears in a system configuration with more processing units: $\psi(T) = 5.583$ with $N = 12$. Since the shared memory model bottleneck has been avoided changing the system design, this result could be expected. For the efficiency quality of service no significant differences can be observed with the variable scaling path due to the difficulty in improving the design efficiency with the system scaling. Anyway, the scalability decrease with the system scaling is slower than with the fixed scaling path.

6.2.2. Robustness experiment

In this case the selected application performs 256 vector orderings. With this design the Host Element generate random vectors and sends to the PU's in the Processing Element homogeneous tasks to order them. The system has been designed with a distributed memory approach and it has been again implemented on a Xilinx4000 FPGA. The application vectors have 64 elements and for the QoS-robustness the tasks response time (t) has been selected as the quality of service metric. Each vector ordering is supposed to be performed in around 4000 μs, therefore $\bar{t} = 4000$ μs. And the robustness tolerance for this experiment is $\tau_R = 0.8$ for the two kinds of robustness metrics.

Table 4
Robustness radius for the QoS-robustness and *P*-robustness with different system configurations.

N	$\rho_{QoS}(L)$ (*tasks*)	$\rho_P(t)$ (μs)
1	51	16000
2	51	11313
3	51	9273
4	51	8000
5	51	7211
6	51	6557
7	51	6082
8	51	5656
10	51	5099
12	51	4690
16	51	4000
32	51	2828

Table 5
Experimental analysis of the QoS-robustness.

N	L	A	Kind of system
1	0	1.00	*QoS Robust*
2	0	1.00	*QoS Robust*
3	0	1.00	*QoS Robust*
4	0	1.00	*QoS Robust*
5	2	0.99	*QoS Robust*
6	4	0.98	*QoS Robust*
7	6	0.98	*QoS Robust*
8	8	0.97	*QoS Robust*
10	106	0.58	*No–QoS Robust*
12	256	0	*No–QoS Robust*
16	256	0	*No–QoS Robust*
32	256	0	*No–QoS Robust*

For the *P*-robustness, the expected *P* value for the j_{th} processing unit is:

$$\overline{P_j} = \overline{\lambda_j}\frac{1}{1+\frac{\overline{T_j}}{\overline{T_j}}} = 0.5 \cdot \overline{\lambda_j}.$$

Because in the perfect scenario $T_j = \overline{T_j}$. And to compute $\overline{\lambda_j}$ for this processing unit:

$$\overline{\lambda_j} = \frac{\overline{q_j}}{\overline{T_j}} = \frac{\lceil \frac{W}{N} \rceil}{\bar{t} \cdot \lceil \frac{W}{N} \rceil} = \frac{1}{\bar{t}}.$$

Because it is a homogeneous system and all the processing units execute the same kind of task with the same expected response time ($\bar{t}$), therefore they all are supposed to execute the same number of tasks.

With these expressions and all the equations proposed in Section 5 to perform the robustness analysis, the robustness radius for different system configurations can be theoretically computed. The results are shown in Table 4.

The meaning of the results shown in Table 4 is very simple to interpret. In the case of the QoS-robustness, only 51 tasks or less can fail in achieving the objective of having a response time below 4000 μs to comply with the robustness condition given in Eq. (17). It has to be pointed that it is a discrete radius (number of tasks) independent on the number of processing units in the system.

On the other hand, for the *P*-robustness the robustness radius is continuous (it is a time measure) and it is dependent on the number of processing units: the more processing units in the system the less robustness radius, and therefore, the more difficult to have a robust design. The meaning of these radius values is the maximum deviation a task can have from its expected response time without causing a violation of the condition given in Eq. (20). For example, suppose a system configuration with $N = 16$. If all the tasks have a response time below 4000 μs except one, and this one has $t = 6000\mu$ s, the system is robust and comply with this condition, being therefore a *P*-robust system. But if this task has $t = 9000$ μ s, the deviation is above the robustness radius and the robustness condition is not fulfilled.

To validate these conclusions, an experimental analysis has been performed. The obtained results are shown in Tables 5 and 6 for the QoS-robustness and for the *P*-robustness respectively. Real executions of the studied applications have been performed on real implementations of the HIPAOC system to measure the L values (number of tasks that are executed without achieving the desired response time), the objective achievement value (A), the maximum deviations of the tasks response times from their ideal response time ($(t - \bar{t})_{max}$), the application response time (T) and the system power (P). And the conditions expressed in Eqs. (17) and (20) have been evaluated to verify if the predictions made with the robustness radius about the system robustness are correct.

Table 6
Experimental analysis of the P-robustness.

N	$(t-\bar{t})_{max}$ (μs)	T (μs)	P	$\bar{P}$	Kind of system
1	-	372165	0.000504	0.000125	*P Robust*
2	-	158150	0.001248	0.000241	*P Robust*
3	-	114456	0.001678	0.000372	*P Robust*
4	-	88155	0.002177	0.000485	*P Robust*
6	860	113990	0.001538	0.000516	*P Robust*
7	711	108543	0.001632	0.000525	*P Robust*
8	750	106393	0.001314	0.001000	*P Robust*
10	3552	115988	0.001140	0.001032	*P Robust*
12	4691	122345	0.000875	0.001454	*No–P Robust*
16	7245	127400	0.000672	0.002000	*No–P Robust*
32	22337	155743	0.000280	0.004000	*No–P Robust*

The results shown in Tables 5 and 6 verify the proposed robustness metrics and the expressions proposed to compute their associated robustness radius. When L is above the QoS-robustness radius of 51, the system is not able to fulfill the robustness condition expressed in Eq. (17). And when the $(t-\bar{t})_{max}$ value is above the P-robustness radius (Table 4) the same thing occurs for the condition given in Eq. (20).

7. Conclusions and future work

The most important conclusion that can be drawn from the presented research is the starting hypothesis confirmation: it is possible to define high level performance metrics to quantify the behavior of on-chip multiprocessors on FPGA platforms.

Often, the traditional solutions for hardware performance evaluation (area, traditional cost measures and minimum period) are not appropriate for this kind of multiprocessor and are not capable of guiding designers and users in their decisions. High level metrics such as efficiency, scalability or robustness are needed to find the optimum configuration of the system: optimum number of processors, memory model, communications system, etc. In this paper, new definitions for efficiency, scalability and robustness for this kind of system have been proposed. And they have demonstrated to be an important aid to designers and users in all their decisions.

In addition, two procedures for performance evaluation have been described in this paper. The first one to analyze the systems scalability through different configurations and using different scaling paths. The second is to derive the system robustness when unavoidable perturbations take place.

For validation and illustration, the proposed metrics and procedures have been employed to evaluate the performance of a specific FPGA-based multiprocessor design: the HIPAOC system. This example validates the proposed metrics definitions and shows the utility of the performance evaluation procedures in the design and system utilization stages.

The most interesting line for future research is perhaps related to extending the proposed set of high level performance metrics to quantify new system attributes such as its sensitivity.

References

[1] A. Baghdadi, D. Lyonnard, N. Zergainoh, A. Jerraya, An efficient architecture model for systematic design of application-specific multiprocessor SoC, in: Proceedings of the European Workshop on Design Automation and Test, 2001.
[2] F. Gharsalli, A. Baghdadi, M. Bonaciu, G. Majauskas, W. Cesario, A.A. Jerraya, An efficient architecture for the implementation of message passing programming model on massive multiprocessor, in: Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping, 2004, pp. 80–87.
[3] S. Meftali, F. Gharsalli, F. Rousseau, A. A. Jerraya, An optimal memory allocation for application-specific multiprocessor system-on-chip, in: Proceedings of the 14th International Symposium on Systems Synthesis, 2001, pp. 19–24.
[4] R.Dimond, O.Mencer, W. Luk, Application-specific customisation of multi-threaded soft processors, in: IEEE Proceedings on Computers and Digital Techniques, vol. 153, 2006, pp. 173–180.
[5] M. Maxfield, The Design Warrior's Guide to FPGAs, Elsevier, Newnes, 2004.
[6] A. Jerraya, W. Wolf, Multiprocessor Systems-on-Chip, Morgan Kaufmann, 2005.
[7] J.L. Hennessy, D.A. Patterson, Computer Architecture. A Quantitative Approach, Morgan Kauffmann, 2003.
[8] S. Sahni, V. Thanvantri, Performance metrics: Keeping the focus on runtime, IEEE Parallel and Distributed Technology 4 (1) (1996) 43–56.
[9] R. Buyya, High Performance Cluster Computing: Architectures and Systems, Prentice Hall, 1999.
[10] D. Nussbaum, A. Agarwal, Scalability of parallel machines, Communications of the ACM 34 (3) (1991) 57–61.
[11] J.L. Gustafson, Reevaluating Amdahl's law, Communications of the ACM 31 (5) (1988) 532–533. http://doi.acm.org/10.1145/42411.42415.
[12] V. Kumar, V.N. Rao, Parallel depth first search. Part II: Analysis, International Journal on Parallel Programming 16 (6) (1987) 501–519. http://dx.doi.org/10.1007/BF01389001.
[13] X. Zhang, Y. Yan, K. He, Latency metric: An experimental method for measuring and evaluating parallel program and architecture scalability, Journal on Parallel and Distributed Computing 22 (3) (1994) 392–410. http://dx.doi.org/10.1006/jpdc.1994.1100.
[14] P. Jogalekar, M. Woodside, Evaluating the scalability of distributed systems, IEEE Transactions on Parallel and Distributed Systems 11 (6) (2000) 589–603.
[15] X.-H. Sun, D. Rover, Scalability of parallel algorithm-machine combinations, IEEE Transactions on Parallel and Distributed Systems 5 (6) (1994) 599–613.
[16] S. Ali, A. Maciejewski, H. Siegel, J. Kim, Definition of a robustness metric for resource allocation, in: Proceedings of the International Symposium on Parallel and Distributed Processing, 2003.

[17] S. Ali, H. Siegel, A. Maciejewski, The robustness of resource allocation in parallel and distributed computing systems, in: Proceedings of the Third International Symposium on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004, pp. 2–10.
[18] HandelC language reference manual, Tech. Rep. RM-1003-3.0, Celoxica, 2002.
[19] Xilinx, XC4000E and XC4000X series Field Programmable Gate Arrays, Product Specification 1.6, Xilinx, 1999.

Marta Beltrán received the master degree in electrical engineering from University Complutense of Madrid, Spain, in 2001, the master degree in industrial physics from UNED, Spain in 2003 and the Ph.D. degree from the Computing Department, Rey Juan Carlos University, Madrid, Spain in 2005. She is currently working with this department as a lecturer. She is the leader of the GAAP research group and has extensively published in high-quality national and international journals and conference proceedings in the areas of computer architecture and parallel and distributed systems. Her current research interests are high performance computing, performance modeling, load balancing and heterogeneous systems.

Antonio Guzmán received the master degree in applied physics from University Autonoma of Madrid, Spain, in 1999 and the Ph.D. degree from the Computing Department, Rey Juan Carlos University, Madrid, Spain in 2006. He is a coauthor of more than 20 papers in international conference proceedings and journals. He is currently an associate professor in the Computing Department of Rey Juan Carlos University and his current research interests are high performance computing, performance modeling, interoperability and security.

Fernando Sevillano received the master degree in economics and business from University Complutense of Madrid, Spain, in 1994. He has been working as consultant and manager in different IT companies and he is currently collaborating with Rey Juan Carlos University. His Ph.D. is centered on performance metrics and performance evaluation procedures for real time systems and on interoperability in industrial environments.