Pi and pid regulation approaches for performance-constrained adaptive multiprocessor system-on-chip

Gabriel Marchesan Almeida, Remi Busseuil, Luciano Ost, Florent Bruguier, Gilles Sassatelli, Pascal Benoit, Lionel Torres, Michel Robert

PI and PID Regulation Approaches for Performance-Constrained Adaptive Multiprocessor System-on-Chip

Gabriel Marchesan Almeida, *Student Member, IEEE*, Rémi Busseuil, Luciano Ost, Florent Bruguier, Gilles Sassatelli, *Member, IEEE*, Pascal Benoit, *Member, IEEE*, Lionel Torres, and Michel Robert

***Abstract*—Adaptive multiprocessor systems are appearing as a promising solution for dealing with complex and unpredictable scenarios. Given the large variety of possible use cases that these platforms must support and the resulting workload variability, offline approaches are no longer sufficient because they do not allow coping with time changing workloads. This letter presents a novel approach based on the utilization of PI and PID controllers, widely used in control automation, for optimizing resources utilization in Multiprocessor System-on-Chip (MPSoC). Several architecture characteristics such as response time during frequency changing, noise and perturbations are modeled and validated in a high-level model and results are compared to information obtained on a homogeneous MPSoC platform prototype. Power and energy consumption figures are discussed and two controllers are proposed: 1) PI-; and 2) PID-based controllers. Results show the system capability of adapting under disturbing conditions while ensuring application performance constraints and reducing energy consumption.**

***Index Terms*—Closed loop system, linear feedback control systems, multiprocessing systems, multitasking, system-on-a-chip.**

I. Introduction

ADAPTATION is increasingly regarded as a promising technique to further optimizing a number of relevant parameters in future embedded systems, such as power consumption, quality of service, or even reliability. In multiprocessor systems, adaptation is often realized through dynamic task mapping, load balancing. Such techniques increase system efficiency through better use of processing and communication resources. These system-level adaptation strategies often rely on a centralized controller that performs the decision-making, and applies resulting decisions with typical latencies in the order of milliseconds. At lower-level, there also exist opportunities such as dynamic voltage and frequency scaling (DVFS). However, because of the distributed nature of the targeted systems, DVFS is either: 1) decided at design-time on a scenario-driven basis (frequency associated to a given system configuration); or 2) performed at run-time by means of distributed decision-making strategies as centralized approaches would incur a prohibitive latency (local monitoring and 2-way communication with the centralized unit).

DVFS techniques have been used in portable embedded systems in order to reduce the energy consumption and temperature of such systems. Most existing DVFS are defined at design time, where they are typically based on predefined profiling analysis (e.g., application data) that attempts at defining an optimal DVFS configuration [1], [2]. Puschini *et al.* [3] propose a scalable multiobjective approach based on game theory, which adjusts at run-time the frequency of each PE while reducing tile temperature and maintaining the synchronization between application tasks. Due to the dynamic variations in the workload of these systems and its impact on energy consumption, other adaptation techniques such as proportional-integral-derivative (PID)-based control have been used to dynamically scale the voltage and the frequency of processors [4], [5], and recently, of networks-on-chip (NoCs) [6], [7]. These techniques differ in terms of adopted control parameters (e.g., task deadline, temperature) and response times (period necessary to stabilize a new voltage/frequency). In [5], authors propose an analytic (queue-domain network) approach that explores the use of a PID controller, which adjusts the multiple clock domain (MCD) processor frequency according to the workload change. In turn, Zhu *et al.* [5] combine a PID-based controller with a DVS scheduler in order to make the system adaptive to varying workload, while reducing the power consumption.

Ogras *et al.* [6] propose an adaptive control technique for a multiple clock domain NoC, which considers the dynamic workload variation aiming at decreasing the power consumption. Besides, the proposed control technique ensures the operation speed and the frequency limits of each island (clock domain, defined according to a given temperature). The effectiveness of the proposed adaptive control technique was evaluated considering different scenarios (e.g., comparison with a PID controller) for a MPEG-2 encoder. In [7], a PID controller is used to provide a predetermined throughput for multiple voltage-frequency/voltage-clock NoC architectures. The proposed PID-based controller sets the voltage and frequency of each NoC island by reserving virtual channels weights (primary control parameter) in order to provide the necessary throughput for different application communications, while saving energy. In [8], we have proposed a PID-based approach for ensuring application performance requirements. However, only PI components were used and no consideration about power and energy consumption is taken into account.

The authors are with Laboratory of Informatics, Robotics, and Microelectronics of Montpellier (LIRMM), University of Montpellier II, Montpellier 34095, France (e-mail: gmalmeida@ieee.org; remi.busseuil@lirmm.fr; ost@lirmm.fr; florent.bruguier@lirmm.fr; gilles.sassatelli@lirmm.fr; pascal.benoit@lirmm.fr; lionel.torres@lirmm.fr; michel.robert@lirmm.fr).

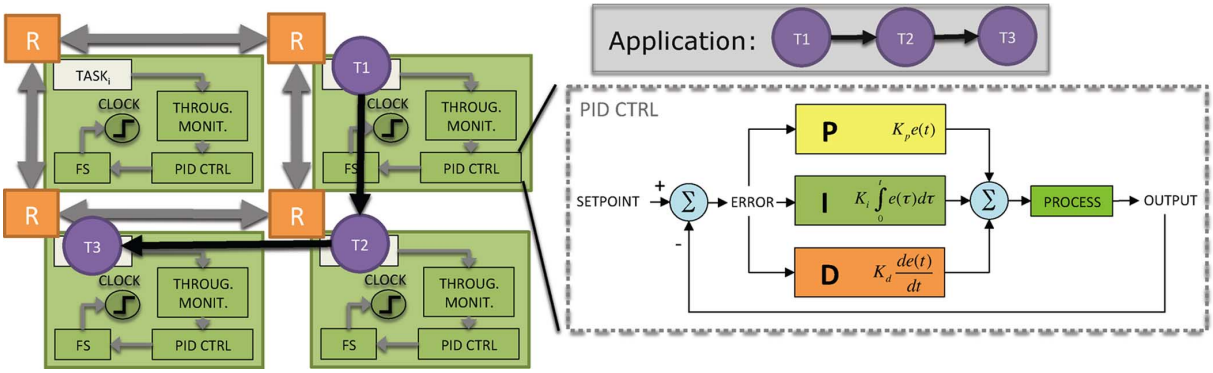

Fig. 1. Architectural overview of the MPSoC platform with PID controllers.

The present letter presents the following contributions: 1) power and energy consumption are considered when tuning processor frequency; and 2) a PI-only controller is proposed and compared to a PID-controller;

The work presented in this letter relies on an open-scale MPSoC system that has been proposed in [9]. It aims at devising a smart distributed frequency scaling strategy that makes it possible to meet real-time application requirements in the presence of perturbations originated from various sources among which the application-related time-changing workloads and especially the continuous adaptation process that the system undergoes, such as task migrations. This letter is organized as follows. Section II illustrates the proposed strategy while Section III discusses experimental results and conclusions are drawn in Section IV.

II. Proposed Strategy

Fig. 1 gives an overview of the proposed approach. As it can be observed, one PID controller is devoted to each task in the system that must ensure soft-real time constraints. In this example, there is one task per network processing unit (NPU), so one PID controller for each processor is required. In the case where multiple tasks are executed in the same NPU, a system with multiple PID controllers in the same NPU could be proposed. In this case, a simple heuristic would choose the highest frequency calculated by the different PID controllers in order to avoid deadline misses. The PID controller is implemented as a service in the operating system and it represents an overhead of less than 1% in terms of total occupied memory footprint.

Monitoring information such as task throughput is fed into the PID controller module that will match the actual throughput with the desired throughput (*setpoint*) and will then calculate an error value (e). This e value is used for calculating P, I and D components and as result the PID controller will indicate a frequency such as to reach a given *setpoint* according to reactiveness factor initially set. It is important to observe that the PID management and calculation is performed at run-time. In cases where actual throughput is lower than the *setpoint*, the PID controller will indicate a frequency superior to the actual one in order to reach the *setpoint* and to satisfy application performance constraints. On the other hand, when the actual throughput is much higher than the expected one, the controller will sign to a lower frequency in order to save platform resources and consequently energy.

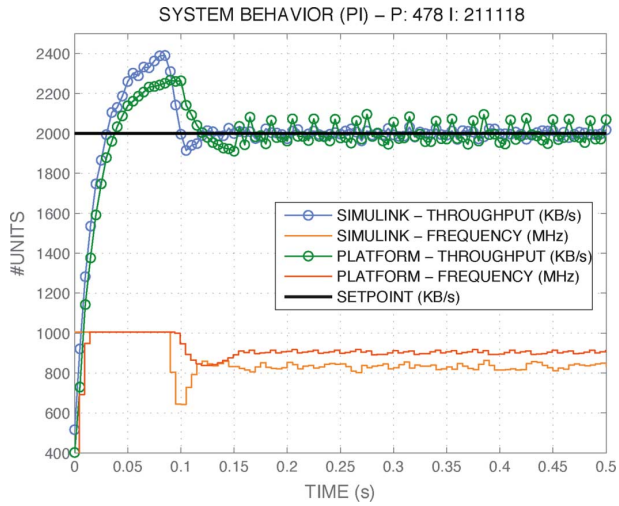

Fig. 2. System behavior—simulink *versus* MPSoC platform.

The proposed strategy consists in deciding controller components on a task basis. To this purpose, a SystemC/TLM-based MPSoC simulation is executed in order to obtain the step-response of the process (see right box in Fig. 1) which corresponds to the NPU executing the task it hosts. In this scenario, processor frequency is changed from 55 MHz to 1005 MHz and task throughput is monitored. The system is linear, that means the system's behavior remains the same under such frequency change. Based on the high-level model, a number of different configurations of controllers can be explored. Each one exhibits different features such as speed, overshoot and static error. Once the process is modeled, PID components are fine tuned by using Simulink and the values of P, I and D are fed as input to the MPSoC platform. Fig. 2 presents the cycle-accurate simulation results showing both high-level model and platform behaviors. It is important to observe a very close match between the two models. In this scenario a PI-based model was used and components have been set to $P = 478$ and $I = 211,118$.

Fig. 3 illustrates the network processing unit represented in a high-level model. The system is mainly composed of:

1) *process:* represented by the *DISCRETE FILTER* in Fig. 3. It characterizes the system behavior based on the step-response process previously described;
2) *noise generator:* it represents system throughput variations of a given application;

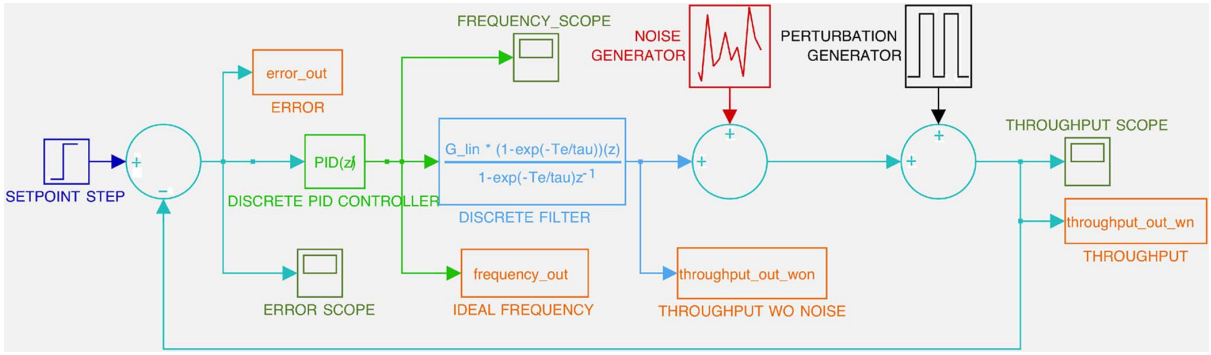

Fig. 3. High-level model of a network processing unit with PID controller, system characterization, and perturbation generator.

TABLE I
REPRESENTATION OF THREE DIFFERENT PERTURBATION SCENARIOS WITH THEIR RESPECTIVE IMPACT

Case	Perturb. Level	Processor Frequency (*MHz*)	Average Through. Before Perturb. (*KB/s*)	Average Through. After Perturb. (*KB/s*)	Impact Factor
$P1$	*low*	425	1,050	610	42%
$P2$	*medium*	425	1,050	540	49%
$P3$	*high*	425	1,050	440	58%

3) *nerturbation generator:* it reproduces the system behavior in the occurrence of perturbation in the system such as task migration, application's workload changes, etc;
4) *PID controller:* represented by the *DISCRETE PID CONTROLLER* in Fig. 3. It symbolizes both PI and PID controllers used for tuning processor frequency according to application constraints.

In the perturbation scenario, an application task is migrated from a different processor generating a perturbation in the system. Once our operating system is implemented in a thread-based way where CPU processing resources are shared among applications, whenever there is a new task running on it, the task will consume CPU power and then the application performance of the tasks previously running on it will decrease. The process modeling is composed of three steps: 1) perturbation scenario is implemented and validated in a real platform and performance figures are obtained; 2) based on this information, platform behaviors are modeled in a high-level model and perturbation amplitude in terms of performance are considered; and the last step consists in 3) matching both high-level model results in order to calibrate and fine adjust high-level model components.

III. CASE STUDY AND RESULTS

The following scenarios are based on a MJPEG video decoder application composed of five tasks running on a processor and a different application (a synthetic application implemented in such way that it requires a considerable amount of processing power) is migrated to the processor, disturbing the MJPEG application performance. Three different perturbation scenarios ($P1$, $P2$ and $P3$) are created based on different perturbation factors defined respectively as follows: 1) low; 2) medium; and 3) high, represented in Table I.

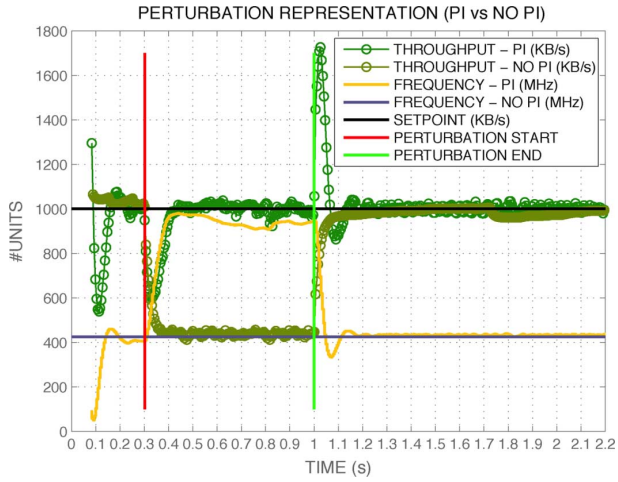

Fig. 4. Perturbation representation—PI *versus* no PI.

Fig. 4 depicts the system response under perturbation condition ($P3$). Application performance is analyzed for two approaches: 1) no adaptation is done and processor frequency keeps constant; and 2) processor frequency is tuned at real-time according to PID controller output values. The perturbation is configured as follows: 1) *perturbation start:* 300 ms; 2) *perturbation period:* 700 ms; and 3) *application performance impact factor:* 58%.

It is possible to observe that when there is no PI controller, the task throughput is reduced to approximately 440 KB/s during the perturbation period. If we consider soft-real time applications where performance constraints must be ensured, this scenario could imply on, for instance, freezing a video during the decoding process or having frame losses during the perturbation period. By adopting our proposed strategy it is possible to notice that the PI controller takes only 100 ms to react and reach the initial *setpoint*, that represents the minimal performance requirements of a given application. When perturbation is over, we can clearly observe that task throughput increases due to the fact that the processor is running in a high frequency in order to compensate the previous existing perturbation. Thus, the PI controller starts decreasing the processor frequency in order to save platform resources, getting back close to the initial *setpoint*.

In Table II, the application performance considering three different perturbation scenarios is presented. For each scenario we have analyzed three different configurations: 1) no PI-/PID-

TABLE II
APPLICATION PERFORMANCE FOR THREE DIFFERENT SCENARIOS

Case	Version	Max. Thr. (*KB/s*)	Min. Thr. (*KB/s*)	Aver. Thr. (*KB/s*)	Max. Freq. (*MHz*)	Min. Freq. (*MHz*)	Aver. Freq. (*MHz*)
$S1$	No PID	$1,756$	$1,011$	$1,416$	710	710	710
	PI	$1,317$	726	$1,000$	716	392	497
	PID	$1,301$	766	$1,000$	731	369	498
$S2$	No PID	$2,040$	$1,047$	$1,548$	800	800	800
	PI	$1,451$	693	$1,000$	809	370	523
	PID	$1,440$	685	$1,000$	834	336	523
$S3$	No PID	$2,440$	979	$1,675$	980	980	980
	PI	$1,727$	599	$1,000$	980	334	567
	PID	$1,752$	609	$1,000$	1005	282	583

TABLE III
NETWORK PROCESSING UNIT (NPU) POWER CONSUMPTION USING STMICROELECTRONICS DESIGN KIT FOR 65 NM TECHNOLOGY

Library	Voltage (V)	Static Power (mW)	Dynamic Power (mW)	Frequency (*MHz*)
$Nominal$	1	0.14	7.46	50

based controller is used and processor frequency keeps constant; 2) PI- and; 3) PID-based controllers are implemented. We can observe that in scenario $S3$, where perturbation impact factor is higher, the average task throughput when using PI-/PID-based controllers is of 1,000 KB/s, resulting on getting the desired performance, represented by the *setpoint*. When using the approach that does not apply PI-/PID-based controller, we can observe that the average throughput is of 1,675 KB/s, representing a waste of platform resources. This is due to the fact that it is necessary to configure the processor to run in a higher frequency to be capable of dealing with scenarios under perturbation conditions. The metric used for choosing the processor frequency consists in selecting the lowest possible frequency where processor can guarantee soft real-time application requirements under perturbation conditions.

In order to obtain the network processing unit power consumption as precisely as possible, we have synthesized the processor on a 65 nm CMOS technology. The design kit is provided by STMicroelectronics and two libraries are used: high voltage threshold (HVT) is used on the critical path of the circuit and low VT (LVT) is used on the remaining of the circuit. The power consumption measurements were obtained from an execution of an embedded operating system's boot loader on a processor running at 50 MHz. Note that, the memory consumption is not considered due to the fact that its operating frequency does not change with the processor frequency. The network processing unit power consumption is presented in Table III.

Table IV presents the power and energy consumption for three different perturbation scenarios. We can clearly see that power and energy consumption are significantly reduced when using PI- and PID-based controllers. In $S1$, the energy consumption is reduced by 32% while power consumption is reduced by 42% in $S3$. It is important to observe that the energy/power consumption saving can be much higher in scenarios with longer perturbation periods and where impact factor of perturbations over applications is higher. We emphasize that by using our proposed approach it is possible not only

TABLE IV
NPU POWER AND ENERGY CONSUMPTION REPRESENTATION FOR THREE DIFFERENT SCENARIOS USING 65 NM TECHNOLOGY

Case	Version	Dynamic Power (mW)	Total Power (mW)	Energy (mJ)	Time (s)
$S1$	No PID	295.50	295.64	802.10	2.56
	PI	206.90	207.04	545.10	
	PID	207.10	207.24	545.70	
$S2$	No PID	333.00	333.14	903.80	
	PI	217.70	217.84	573.70	
	PID	217.90	218.04	574.10	
$S3$	No PID	407.90	408.04	$1,107.10$	
	PI	236.00	236.14	620.70	
	PID	242.90	243.04	638.70	

complying with application performance constraints and saving energy/power but also it is possible to reduce waste of platform resources in order to reach an optimal solution.

IV. CONCLUSION

This letter has presented a novel approach based on the utilization of PID controllers for energy consumption reduction while ensuring application performance constraints. The strategy does not rely only on preventing application deadline-misses but also attempts to save energy, once the processor frequency is adjusted at real-time according to application requirements. We claim that our approach can be easily integrated to linear systems and, as result, platform resources utilization can be optimized. Moreover, by using PID controllers it is possible to save up to 42% in terms of power by tuning processor frequency according to application needs. The letter also presents: 1) PI- and; 2) PID-based controllers. As can be observed, PID-controllers are intended to react faster under disturbing conditions when compared to PI-only controllers, however its power consumption is higher. The good choice of which controller to use in multiprocessor system-on-chip platforms will be a trade-off between power consumption and the desired system's reactiveness.

REFERENCES

[1] G. Magklis *et al.*, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor," *SIGARCH Comput. Archit. News*, vol. 31, pp. 14–27, May 2003.

[2] F. Xie *et al.*, "Compile-time dynamic voltage scaling settings: Opportunities and limits," *SIGPLAN Not.*, vol. 38, pp. 49–62, May 2003.

[3] D. Puschini *et al.*, "A game-theoretic approach for run-time distributed optimization on MP-SoC," *Int. J. Reconfig. Comput.*, 2008.

[4] Q. Wu *et al.*, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," *SIGARCH Comput. Archit. News*, vol. 32, pp. 248–259, Oct. 2004.

[5] Y. Zhu and F. Mueller, "Feedback edf scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling," *SIGPLAN Not.*, vol. 40, pp. 203–212, Jun. 2005.

[6] U. Y. Ogras *et al.*, "Variation-adaptive feedback control for networks-on-chip with multiple clock domains," in *Proc. 45th Annu. Des. Autom. Conf. (DAC'08)*, Jun. 2008, pp. 614–619.

[7] A. Sharifi *et al.*, "Feedback control for providing qos in NoC based multicores," in *Proc. Conf. Des., Autom. Test Eur. (DATE'10)*, Mar. 2010, pp. 1384–1389.

[8] G. M. Almeida *et al.*, "Predictive dynamic frequency scaling for multi-processor systems-on-chip)," in *IEEE Int. Symp. Circuit. Syst. (ISCAS'2011)*, May 2011.

[9] G. M. Almeida *et al.*, "An adaptive message passing MPSoC framework," *Int. J. Reconfig. Comput.*, Oct. 2009.