Predictive Dynamic Frequency Scaling for Multi-Processor Systems-on-Chip

Gabriel Marchesan Almeida*, Rémi Busseuil*, Everton Alceu Carara†, Nicolas Hébert*, Sameer Varyani*, Gilles Sassatelli*, Pascal Benoit*, Lionel Torres*, Fernando Gehm Moraes†
*Laboratory of Informatics, Robotics and Microelectronics of Montpellier (LIRMM) - Department of Microelectronics
Email: {firstname.lastname}@lirmm.fr
†Pontifical Catholic University of Rio Grande do Sul (PUCRS) - Faculty of Informatics
Email: {firstname.lastname}@pucrs.br

***Abstract*—This paper proposes a novel strategy for optimizing resources in Multi-Processor Systems-on-Chip (MPSoC). The approach is based on using control-loop feedback mechanism to maximize the efficiency on exploiting available resources such as CPU time, operating frequency, etc. Each Processing Element (PE) in the architecture is equipped with a frequency scaling module responsible for tuning the frequency of processors at run-time according to the application requirements. Results show the system's capability of adapting to disturbing conditions. For validation purposes we have implemented a multi-threaded MJPEG decoder together with an ADPCM audio decoder and a FIR.**

I. Introduction

Dynamic Frequency Scaling (DFS) is a widely used technique aimed at adjusting computational power to application needs. It is often associated to Dynamic Voltage Scaling (DVS) therefore enabling to achieve significant power reductions when computing demand is low; some cited benefits also comprise the reduction of thermal hotspots that participate in the accelerated aging of the circuits due to the thermal stress.

In multicore systems such as general purpose processors and high performance embedded processors, the operating system is responsible of dynamically adjusting the frequency of each processor to the current workload. This is facilitated by the presence of dedicated hardware monitors that the OS can rapidly access. In Linux based systems, two popular policies are used at kernel-level: on-demand and conservative. The on-demand governor switches to the highest available frequency whenever a load is detected whereas the conservative policy incrementally increases the frequency in a step-by-step fashion, yielding to better power savings at the expense of a lesser reactivity. Such schemes are possible in centralized shared memory systems only as a unique operating system has an immediate global observability of the current system state.

In distributed memory multiprocessor systems where communications take place through exchanged messages, such techniques cannot be applied due to the difficulty of obtaining an instant snapshot of the system workload due to communication delays, etc. Consequently, in such systems DFS is decided at design-time based on a number of possible run-time scenarios (defined by a task mapping or selection of algorithms, etc.). This approach, although efficient, is restrictive mostly because of its capability of handling a limited set of scenarios. The growing interest for highly adaptive multiprocessor systems capable of taking decisions online (such as task migration for load balancing) based on possibly time-changing criteria challenge these approaches and demand for better flexibility.

The work presented in this paper relies on such an MPSoC system that has been proposed by the authors [1]. 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 the continuous adaptation process that the system undergoes, such as task migrations.

The paper is organized as follows. Next section presents the related work. Section III describes the proposed architecture. Section IV discusses the PID controller and presents the system model. Experimental results are presented in Section V while conclusions are drawn in Section VI.

II. Related Work

Recently, researchers have put focus on adaptation techniques in order to cope with dynamic and unpredictable behaviors that can appear in nowadays embedded systems. This section presents some work that has been conducted in this direction using techniques such as task migration mechanisms and dynamic voltage and frequency scaling.

A. Task Migration Mechanisms

A number of contributions in the literature based on distributed memory systems have used shared memory as a means for enabling task migration [2] [3] [4]. In [3] each core runs a single operating system instance in its logical private memory. Processor cores execute tasks from their private memory and explicitly communicate with each other by means of shared memory. The target platform uses a shared bus as interconnect.

In the case of distributed memory MPSoCs, in [5] we have proposed an adaptive strategy that is responsible for making decisions at run-time. Decisions are taken by processors in a distributed fashion and relate mostly to application performance. In [1] authors address both the benefits brought by task migration mechanisms as well as the associated performance penalty in a purely distributed homogeneous MPSoC architecture. Taking into account the future homogeneous

978-1-4244-9472-9/11/$26.00 ©2011 IEEE

MPSoC systems, scalable architectures with purely distributed memory system are suitable. To the best of our knowledge, our architecture is the only one with a purely distributed memory system which does not rely on shared memory for enabling task migration [1]. Instead, it uses the NoC as a communication link where the tasks are transmitted during the migration process.

B. Dynamic Voltage and Frequency Scaling (DVFS)

DVFS has been a widely applied technique for reducing power consumption in many domains, especially those concerning micro-architectures such as embedded systems. In [6] authors show a technique for minimizing the total power consumption of a Chip Multiprocessor (CMP) while maintaining a target average throughput. The proposed solution relies on a hierarchical framework, which employs core consolidation, coarse-grain dynamic voltage and frequency scaling (DVFS).

Adaptability has been explored under a number of aspects in embedded systems, ranging from adaptive modulation used in the future 3GPP-LTE standard (SDR for software defined radio) [7] to adaptive cores instantiation in dynamically reconfigurable FPGAs [8].

C. Paper Contribution

Most of the proposed strategies are based on a shared memory scheme. The system elects a master node which is responsible for controlling the frequency of each processor in the architecture. Others existing solutions have pre-defined states in which processors change the frequency whenever a given condition is respected. Our approach differs of the others by the following reasons: 1) there is no master in the architecture, so that decisions are taken in a distributed way. Each processing element (PE) in the architecture controls its own frequency; 2) system optimizations are not global since there is no centralized control of the system, making system management more difficult.

This paper presents two major contributions:

1) A novel and purely distributed memory architecture with adaptation capabilities driven by local PID controllers;
2) Analyze and discuss the benefits of using such strategy on a Homogeneous MPSoC architecture running audio/video applications;

III. SHoP Architecture

The key motivations of our approach are scalability and adaptability; the system presented in the rest of this paper is built around a distributed memory/message passing system that provides efficient support for task migration. For these reasons the architecture is named SHoP (**S***elf-adaptive* **Ho***mogeneous* **P***latform*). This system 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. Fig. 1 presents a structural view of the SHoP architecture.

The NPU is built of two main layers, the network layer and the processing layer. The network layer is essentially a compact routing engine based on the Hamiltonian Routing Algorithm [9]. The Network-on-Chip (NoC) used in this work was proposed in [10].

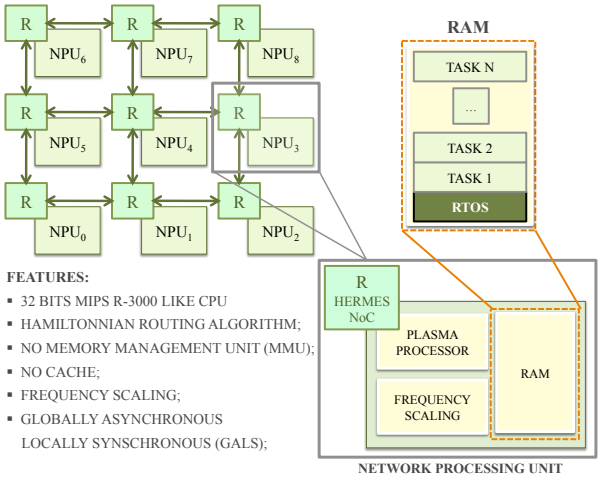

Fig. 1: Structural View of the SHoP Architecture

IV. System Model

A proportional-integral-derivative controller (PID controller) is a generic control-loop feedback mechanism (controller) widely used in industrial control systems. A PID controller calculates an *error* value as the difference between a measured process variable and a desired *setpoint*. The controller attempts to minimize the error by adjusting the process control inputs. In the absence of knowledge of the underlying process, a PID controller is a suitable controller [11]. However, for best performance, the PID parameters used must be tuned according to the nature of the process to be regulated.

In this paper we propose the usage of a PID controller for adjusting the appropriated frequency of the processors at the same time as deadline miss ratio is reduced. The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$ as the controller output, the final form of the PID algorithm is:

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \quad (1)$$

1) *Proportional gain,* K_p: larger values typically mean faster response since the larger the error, the larger the proportional term compensation. *2) Integral gain,* K_i*:* larger values imply steady state errors are eliminated more quickly. *3) Derivative gain,* K_d*:* larger values decrease overshoot, but slow down transient response and may lead to instability due to signal noise amplification in the differentiation of the error. Fig. 2 summarizes a traditional PID controller.

Most applications in embedded systems are based on soft-real time constraints. Actual architectures have to be capable of adapting to avoid situations where deadlines are missed. In the proposed strategy, applications requirements are taken into account aiming to provide QoS (Quality-of-Service). As entry point (*setpoint* in Fig. 2), the system is fed with the

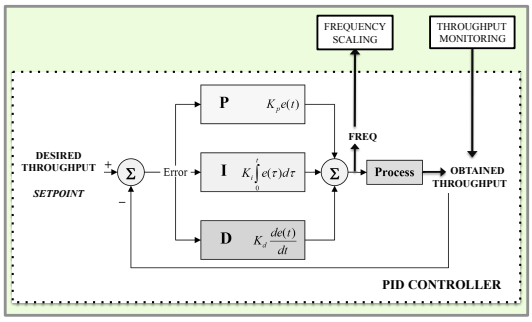

Fig. 2: PID Controller

application requirements, e.g. the minimal throughput the application requires to ensure the functionality of the system in a reliable way.

The system then calculates an *error* value which is obtained by the difference between the *desired* and *obtained* throughput. As output of the PID controller a frequency value is indicated. This value is sent to the *frequency scaling* module which will be responsible for scaling up and down the frequency of the processor to cope with application requirements. The procedure is then repeated and the *obtained* throughput gradually gets closer to the *desired* throughput. This is explained by the fact that after each iteration the *error* value is reduced assuming that the values of P, I and D have been correctly chosen. Fig. 3 presents an abstract system model of the proposed strategy for each *Network Processing Element (NPU)* in the architecture.

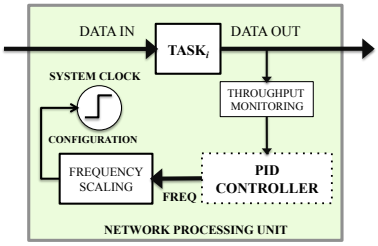

Fig. 3: System Model

Basically each running application is composed of one or multiple tasks. Task are monitored by a throughput monitoring that is responsible for calculating tasks performance in a *non-intrusive mode*. This information is passed to the PID controller which will be responsible for choosing the appropriated frequency in order to speed up or slow down processing. This strategy can be suitable for power saving in embedded systems.

Fig. 4 illustrates an overview of the proposed approach. As previously mentioned, there is one PID controller devoted to each task in the system that must ensure soft-real time constraints. In this example there is one task per NPU, so one PID controller for each processor is required. In the case that there are multiple tasks in the same NPU, we could build a system with multiple PID controllers in the same NPU, each one being responsible for contributing as a factor that will be added to the final result.

The strategy consists in deciding controller parameters on a task basis. To this purpose, a simulation of the MPSoC

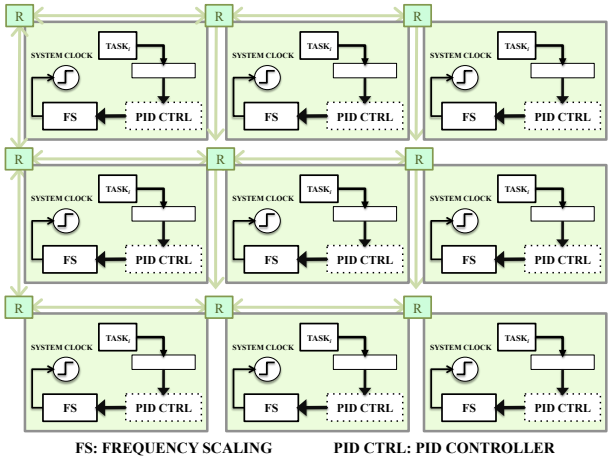

Fig. 4: Distributed PID Controllers

system is executed in order to obtain the step response. Fig. 5 shows the cycle-accurate simulation results and the first order extracted model that is used for the process (Fig. 2). Based on that high-level model, a number of different configurations of controllers can be explored, each of which exhibits different features such as speed, overshoot, static error.

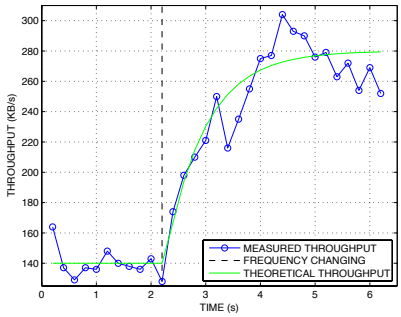

Fig. 5: Obtained Throughput *vs* Theoretical Throughput

The values of P, I and D have been chosen to increase the reactivity of the system. Fig. 6 presents the PID response according to P, I and D values.

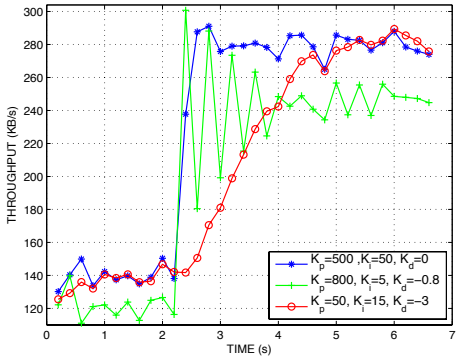

Fig. 6: PID Controller Response

Assuming that the *setpoint* of the system is around 260KB/s we can observe that in the first case where $P = 500, I = 50$ and $D = 0$, the system converges to the *setpoint throughput* rapidly. As result we observe an overshoot in terms of performance. In the second scenario where $P = 800, I = 5$ and $D = -0.8$, due the fact that the value of P is much bigger than

I, we can also observe an *overshoot* in terms of performance. The system throughput presents a high oscillation due to the small value of I. At least, when $P = 50, I = 15$ and $D = -3$ we see that system throughput increases slowly. This is explained by the fact that the value of P is very small and then the convergence time is longer. Based on this information we have chosen to use the first controller, because it converges to a stable system relatively fast.

V. Case Studies and Results

Fig. 7 illustrates the validation scenario. The system executes three applications simultaneously (ADPCM, FIR and MJPEG decoder). There are also two tasks (Split) and (Join), responsible for splitting and joining packets respectively. To analyze the reactiveness of the PID controller, two disturbing tasks, P_1 and P_2, are inserted into the system, and executed simultaneously with the benchmark application.

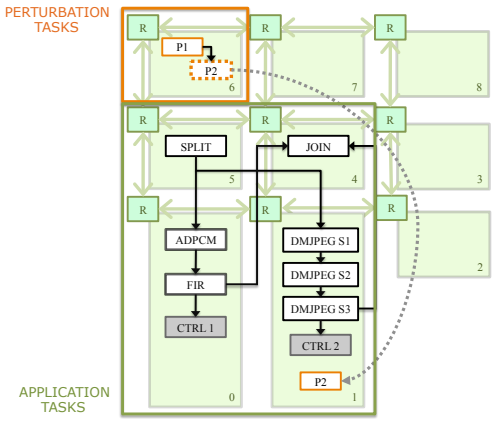

Fig. 7: Task Migration With PID Controller

Three different scenarios (S_1, S_2 and S_3) have been created. In all three scenarios the application processes 4 images of 804 bytes of size each. Fig. 8 presents the obtained throughput for such scenarios. In S_1 there is no perturbation in the system the obtained average throughput is around 125KB/s. In S_2 one of the two perturbation task (P_2) is then migrated to the NPU where all three tasks of the MJPEG application are currently running. The task migration occurs at 0.73s. We can observe that after P_2 has been migrated, the throughput of the system decreases downto 80KB/s. In the third scenario we use the proposed PID controller in order to try to compensate the throughput shoot as fast as possible. We can observe that right after the migration, system throughput starts going down and then the controller increases the frequency of the processor to reach the minimal throughput, defined as *setpoint*. We can clearly see that by using PID controller the system is capable of adapting itself to situations where perturbations might appear. As final result, the system gets stable and in average the obtained throughput after regulation is around 120KB/s, representing a gain of almost 50% compared to the scenario without PID controller.

VI. Conclusions

In this paper we have proposed a strategy for better exploiting architecture resources in a purely distributed memory

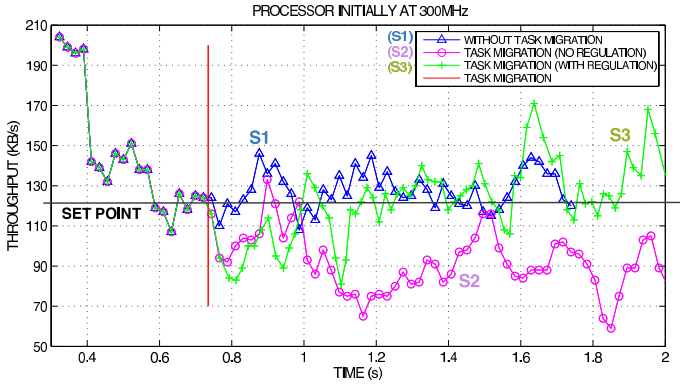

Fig. 8: Task Migration Impact

homogeneous MPSoC architecture. It presents promising results regarding to the adaptability of the system. We have demonstrated the efficiency of the proposed PID controller by presenting three different scenarios. For validating our approach we have implemented a multi-threaded version of the MJPEG decoder together with an ADPCM and FIR application which exchanges message using a message passing interface (MPI). Results shown that by using the proposed strategy, obtained throughput is very close to the expected throughput in scenarios where perturbations are not considered. In scenarios where perturbations are taken into account, we have shown that the system is capable of reacting to the throughput change by means of tuning processor frequency at run-time.

References

[1] G. Marchesan Almeida and Rémi Busseuil et al. Evaluating the impact of task migration in multi-processor systems-on-chip. In *SBCCI'10: Proceedings of the 23rd symposium on Integrated circuits and system design*, pages 73–78, São Paulo, Brazil, 2010. ACM Computer Society.

[2] S. Bertozzi and A Acquaviva. Supporting task migration in multi-processor systems-on-chip: A feasibility study. In *DATE '06. Proceedings*, volume 1, pages 1–6, 2006.

[3] A. Acquaviva, A. Alimonda, S. Carta, and M. Pittau. Assessing task migration impact on embedded soft real-time streaming multimedia applications. *EURASIP J. Embedded Syst.*, 2008:1–15, 2008.

[4] M. Pittau, A. Alimonda, and S. Impact of task migration on streaming multimedia for embedded multiprocessors: A quantitative evaluation. In *ESTImedia*, pages 59–64. IEEE, 2007.

[5] G. Marchesan Almeida and G. Sassatelli et al. An adaptive message passing mpsoc framework. *International Journal of Reconfigurable Computing*, Volume October, 2009.

[6] Mohammad G. and Ehsan P. Minimizing the power consumption of a chip multiprocessor under an average throughput constraint. In *International Symposium on Quality Electronic Design*. IEEE, 2010.

[7] F. Clermidy and Lemaire. An open and reconfigurable platform for 4g telecommunication: Concepts and application. In *DSD '09: Proceedings of the 2009 12th Euromicro Conference on Digital System Design*, pages 449–456, Washington, DC, USA, 2009. IEEE Computer Society.

[8] D. Puschini and F. Clermidy. Dynamic and Distributed Frequency Assignment for Energy and Latency Constrained MP-SoC. In *DATE'09*, pages 1564–1567, Nice, France, 04 2009.

[9] X. Lin, P. K. McKinley, and L. M. Ni. Deadlock-free multicast wormhole routing in 2-d mesh multicomputers. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):793–804, 1994.

[10] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, 38(1):69–93, 2004.

[11] Stuart Bennett. *A History of Control Engineering, 1800-1930*. Institution of Electrical Engineers, Stevenage, UK, UK, 1979.