Achieving QoS in NoC-based MPSoCs through Dynamic Frequency Scaling

Guilherme Guindani, Fernando G. Moraes
PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brazil
guilherme.guindani@acad.pucrs.br, fernando.moraes@pucrs.br

Abstract—**The management of Quality-of-Service (QoS) constraints in NoC-based MPSoCs, with dozens of tasks running simultaneously, is still a challenge. Techniques applied at design or run-time to address this issue adopts different QoS metrics. Designers include in their systems monitoring techniques, adapting at run-time the QoS parameters to cope with the required constraints. In order words, MPSoC are able to self-adapt themselves, while executing a given set of applications. Self-adaptation capability is a key feature to meet applications' requirements in dynamic systems. Dynamic Voltage and Frequency Scaling (DVFS) is an adaptation technique frequently used to reduce the overall energy consumption, not coupled to QoS constraints, as throughput or latency. Another example of adaptation technique is task migration, which focus on throughput or latency optimization. The self-adaptation technique proposed in this paper adopts Dynamic Frequency Scaling (DFS) trading-off power consumption and QoS constraints. Each processor running the applications' tasks initially reaches a steady state leading each task to a frequency level that optimizes the communication with neighbor tasks. The goal of the initial state is to reach a trade-off between power consumption and communication throughput. Next, the application performance is monitored to adjust the frequency level of each task according to the QoS parameters. Results show that the proposed self-adaptability scheme can meet the required QoS constraints, by changing the frequency of the PEs running the application tasks.**

Keywords—NoC-Based MPSoCs, QoS monitoring, Adaptability, Energy consumption.

I. INTRODUCTION

MPSoCs with hundreds of processing elements (PEs) interconnected through a network-on-chip (NoC) is a reality in the design of current embedded systems [1]. According to ITRS (http://www.itrs.net/), future MPSoCs will integrate more than 1,000 PEs by 2025, and these systems must contain mechanisms to self-adapt to the demands of both applications and available resources.

Adaptability is built into MPSoCs by employing different techniques, such as priority-based task scheduling, communication priorities, task migration, DVFS and circuit switching. This adaptability is used to cope with the performance degradation while running multiple applications simultaneously. Each of these techniques targets a specific QoS metric, such as communication priorities for throughput and DFVS for energy consumption.

Focusing in energy consumption adaptability, DVFS is the most used technique. It is based on the fact that the frequency has a linear impact on the energy consumption, and the voltage a quadratic impact. Therefore, controlling these two variables it is possible to adapt the CMOS circuit energy consumption. However, the technology scaling, coupled with the increasing manufacturing process variability, may interfere in the design of DVFS techniques. These variations can result in nominally correct DVFS schemes failing to meet frequency or energy targets [2][3].

Dynamic Frequency Scaling (DFS) is an alternative to the DVFS scheme. In DFS, the voltage of the CMOS circuit is kept unchanged, while the frequency changes according to the workload applied to the PE. In [4], a clock gating technique applied to the reference system clock achieved an important energy consumption reduction, with a small increase in the total execution time of applications.

The *goal* of this paper is to propose an adaptability technique using DFS to achieve application QoS constraints, as throughput or latency. The adopted DFS scheme is applied in both PEs and NoC independently, enabling communication flexibility in the network fabric while maintaining the PE with the required frequency. If an application with QoS constraints is disturbed by any other general-purpose application, the DFS scheme is applied to adjust the frequency of the PEs running the application with constraints.

The rest of this paper is organized as follows. Section 2 analyses related work on MPSoC adaptability, through DVFS, aiming QoS applications. Section 3 details the system architecture and DFS scheme used in the proposed work, while Section 4 describes the adaptability scheme using DFS. Section 5 present the scenarios used to evaluate the proposed scheme and results that were produced by these scenarios, section 6 concludes this paper and gives direction for future works.

II. RELATED WORK

Goossens et al. [5] propose a combined energy and real-time task management aiming the composability of the MPSoC elements. The Authors present real-time scheduling algorithms, highlighting their advantages and disadvantages. The Authors briefly explain the composability paradigm applied into the MPSoC, and describe how to build a composable MPSoC energy management. According to the

978-1-4799-1191-2/13/$31.00 ©2013 IEEE

Authors, to establish this management, the MPSoC processing element, DMA and memory should have independent clock signals.

Mansouri et al. [6] propose an adjustment policy on the DVFS scheme running on the MPSoC local nodes. The Authors developed a mathematical model of the application energy consumption, using a consensus algorithm to adjust the DVFS scheme of each local node. Each node gathers information from its neighbors to reach a consensus and adapt its own DVFS. The paper shows results with 87% power dissipation improvements when compared to the worst-case scenario.

Garg et al. [7] propose a DVFS control policy applied to the Voltage and Frequency Islands (VFI) concept. The VFI divide the MPSoC in different voltage and frequency regions, which are controlled by a DVFS manager. The proposed DVFS control policy, named Custom Feedback (CF), is between the Fully-Centralized (FC) and Fully-Decentralized (FD) policies. The FC uses global MPSoC information to control the VFIs DVFS, and the FD uses neighboring VFI information to control the local DVFS. The proposed CF policy uses both local VFI and neighboring information to control its own DVFS. The amount of information collected from the VFI neighborhood, or feedback, is parameterizable and implies on the performance of the DVFS controlling scheme. The Authors shows that CF has an 8% improvement when compared to FC, and 17% when compared to FD controlling schemes.

Fattah et al. [8] propose a hierarchical monitoring scheme, dividing the MPSoC in regions, named clusters. In the proposal, the clustering is application oriented (a cluster consists of a group of PEs that execute tasks of a given application). Each cluster is managed by a mid-level manager, named application manager. A top-level manager - system manager, is responsible for managing the system as a whole.

Takase et al. [9] propose an optimization flow, at design time, for applications running in a Multi-Purpose Processor (MPP). The Authors start the optimization flow by simulating the application on a cycle-accurate simulator, which enables to profile the application running on the MPP. From these profiles, the Authors optimize the energy and the performance of the application by exploring DVFS and reconfiguration schemes on the MPP. Next, hardware modification (DVFS/reconfiguration) checkpoints are automatically inserted into the application assembly code. Another optimization is in the task allocation, where it is evaluated if the task is going to be allocated in the main memory or in the cache memory. The inter-task dependencies and deadlines is also optimized, verifying energy and time constraints. The Authors shows that, by using their framework it is possible to reduce the MPP energy consumption up to 49%.

The evaluated related work are mainly focused on the DVFS scheme to address the global energy consumption of an MPSoC, however none of them treat the energy consumption as a QoS parameter of an application. Reducing the energy consumption of the MPSoC could influence other application QoS parameters such as throughput and latency, or in the real-time parameters of other applications.

The proposed self-adaptable DFS scheme can manage the application QoS constraints while optimizing the frequency of general-purpose tasks (best-effort applications). The use of such scheme can optimize the overall MPSoC energy consumption.

III. System Architecture

The reference MPSoC [10] is a homogeneous NoC-based MPSoC. Each PE contains the following modules: (*i*) a 32-bit Plasma processor (MIPS-like architecture); (*ii*) a local memory (RAM); (*iii*) a DMA module, responsible for transferring the task object code to the memory and messages to/from the NoC from/to the local memory; (*iv*) a network interface (NI). The Hermes NoC is used to interconnect the PEs.

Fig. 1 presents the PE architecture, with the DFS controller. A GALS interface is inserted between the router and the PE. This GALS interface uses a bisynchronous FIFO [11], and two-flop synchronizers in the control signals. The microkernel (operating system of the processor) monitors the CPU utilization and communication queue occupancy, storing them in memory-mapped registers (*not_scheduled*, *pipe_ocup*, *req_msg* signals in Fig. 1). Based on this information, the DFS controller can take decisions and switch the frequency of the processor dynamically.

It is important to highlight that each individual clock generation module receives the system clock as reference, generating a new clock from it. The benefit of such approach is that the global clock has to feed only the clock generation modules, reducing significantly the global clock load, and hence simplifying the clock tree generation and its power consumption, which is responsible for, at least, 40% of the power consumption in MPSoCs. The global system clock is defined in the following sections as reference frequency.

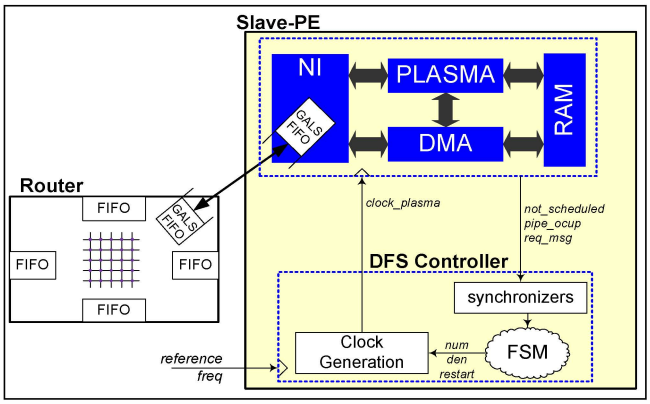

Fig. 1 - Router-PE GALS interface and the DFS controller responsible for generating the PE frequencies [4].

A. Dynamic Frequency Scaling

The DFS scheme adopted was originally proposed in [4], which changes the frequency with a fixed voltage source. The DFS is applied to the PE and the router independently.

The clock generation module is instantiated at each PE and each router. This module uses as input the reference frequency. The principle of the clock generation process is to

generate a new frequency by simply omitting selected cycles of the reference clock. There are two configuration signals, *num* and *den*, to generate a new frequency. For example, to obtain 75% of the original frequency, *num* is set to 3 and *den* to 4. With such configuration, 3 out of 4 clock cycles are propagated to the clock output signal. When the frequency needs to be changed, the clock is stopped by asserting the restart signal. Finally, releasing the restart signal, the new frequency is available to the PE or router. The proposed module is glitch free by construction.

The PE DFS controller computes the communication load and CPU utilization level according to values provided by the microkernel. The key parameter to control the PE frequency is the communication load. Each PE contains a global vector in the local memory, named pipe, controlled by the microkernel, which stores the messages to be sent for the remote PEs. As applications are modeled as task graphs, monitoring the pipe occupation enables to adjust the data flows by regulating the PE frequencies.

In addition, the PE DFS controller implements a communication mechanism to balance power consumption and performance. It takes into account that a consumer processor must receive the data in a frequency that is not inferior to its operating frequency. Thus, if the producer processor is operating at higher frequency than the consumer processor, the message can be sent at a lower frequency to save power. On the other hand, if the producer is operating at a lower frequency than the consumer, its frequency should be temporally increased to not penalize the consumer performance.

The DFS controller integrated at each router is responsible for defining the router frequency. Each input buffer obtains its frequency from the received packets, which carries in the header field the frequency value that it needs to be transmitted. With this information, the NoC frequency controller is able to switch the router frequency for each traffic being routed. Also, the controller can identify the router activity and put the router in a low power mode in case of inactivity, saving a significant amount of energy. When two or more traffic flows are passing through router, the controller selects the higher frequency between them, avoiding performance loss in the network.

IV. Adaptability Using DFS

To meet application QoS requirements, different techniques can be used at design time and at run-time [12]. In MPSoC architectures, with several applications disputing a limited amount of resources, failure to meet the application QoS constraints can be a constant.

As illustrated in Fig. 2, applications are modeled as task graphs, with one or more initial tasks, *n* computational tasks, and an output task. At design time, a profile of the application is obtained to define the QoS constraints. At execution time the output task is monitored, verifying the actual performance against the defined constraints.

Tasks communicate through message passing, using MPI-like send/receive primitives. Non-blocking *sends* insert messages in the communication queues (as illustrated in Fig. 2), while blocking *receives* read from the communication queues. Such communication scheme reduces the overall NoC traffic, since a message is only injected into the network when it is required. If the communication queues are placed at the receiver side, messages could block the NoC when the queues become completed filled.

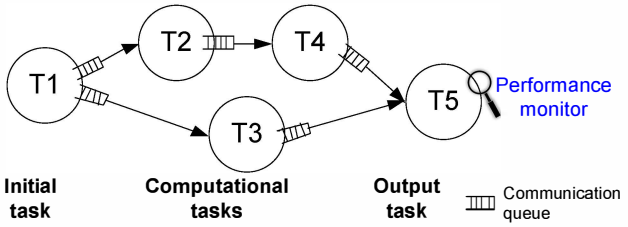

Fig. 2 – Application modeled as a task graph, being the final task monitored for QoS purposes.

When a given application starts its execution, the DFS scheme proposed in [4] is used to lower the overall application energy consumption. The DFS main goal is to put all communication queues with an average occupancy, such that all PEs have data to consume when requested.

The proposed self-adaptation technique can be seen as a closed control loop, as depicted in Fig. 3. Two monitors are used: (*i*) hardware monitors to register the QoS parameters; (*ii*) software monitors, implemented in the microkernel, to evaluate QoS violations. The hardware monitor can be used without the software monitor, to profile a given application QoS parameter. Both share the same evaluation window (parameterizable value) when used together.

The QoS *Evaluation* module combines the use of the two monitors, a *steady state* monitor and a *QoS monitor*. This module, implemented in the microkernel, verifies the severity of the QoS violations, and if it is necessary switch from the original DFS policy to the one controlled by the microkernel (activating the *adaptability mechanism*).

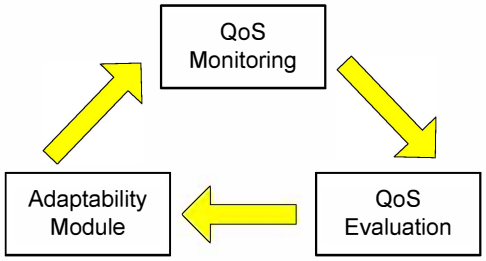

Fig. 3 – Conceptual adaptability scheme.

The monitored application is, in our approach, the application output task. The QoS Evaluation module detects when the frequency reaches a steady state. This steady state is achieved when the PE frequency does not change for a given period (parameterizable value). At this point, the clock generation module of the DFS Controller starts to be managed by the microkernel of the processor, and the QoS monitor starts to evaluate the number of the QoS constraints violations (In the present implementation, throughput is the monitored performance parameter).

The evaluation window and the QoS constraints are defined at the application level, through a system calls. For example:

```
Enable_Monitoring(<task_ID>, <nb#cycles>)
QoS_Set_Throughput(<task_ID>, <nb#bits>)
```

Where: **task_ID**, corresponds to the identification of the monitored task; **nb#cycles** corresponds to the parametrizable evaluation window; **nb#bits** corresponds to the expected number of bits to be received in the monitoring period.

The QoS constraints may not be reached in the following situations: (*i*) the performance obtained at the steady state is inferior to the specified constraint; (*ii*) disturbing traffic interferes with the monitored application, reducing its performance. Also, the performance may be superior to the specified, enabling a frequency reduction.

The QoS *evaluation module*, using the information generated by the monitors, may decide to increase the frequency, if QoS violations are detected, or decrease it, if the performance is superior to a given threshold. This behavior guarantees the application QoS compliance while maintaining a low energy consumption profile.

It is important to note that each application task may be executing with distinct frequencies when the steady state is reached. To avoid a central manager responsible to keep the state of each PE, a distributed approach was adopted. The principle is to use a back-propagate message, ordering an adaptation in the PE frequency. If the QoS evaluation module decides to modify the frequency, the *adaptability module* back-propagate a message to the tasks sending data to the current task, up to the initial task. For example, in Fig. 2 task T5 back-propagate to tasks T4 and T3. If a given task receives two messages to modify the frequency, as T1 in Fig. 2, it discards one of these messages.

When a given PE receives a frequency change message, it verifies if the message should be discarded or not, as previously explained. Then, the following actions are executed: (*i*) the new frequency is transmitted to the DFS Controller; (*ii*) the PE frequency changes; (*iii*) the microkernel create frequency change messages to all PEs generating data to the current PE, injecting them in the NoC.

V. RESULTS

Two synthetic applications are used to evaluate the proposed adaptability scheme: a 6-stage pipeline (*Pipe*) and a producer-consumer (PC_{app}). The computation time of each task is emulated by a loop.

Five test scenarios were simulated:

1. *Pipe* application running alone in the MPSoC. It is used to create a throughput profile of the application, and the QoS constraint value.
2. *Pipe* application running togheter with one PC_{app} (tasks D10 and D11), without adaptivitiy.
3. *Pipe* application running togheter with one PC_{app} (tasks D10 and D11), with monitoring and DFS adaptability.
4. *Pipe* application running togheter with three PC_{app} (tasks D10-D11, D21-D22, D31-D32), without adaptivitiy.
5. *Pipe* application running togheter with three PC_{app} (tasks D10-D11, D21-D22, D31-D32), with monitoring and DFS adaptability.

Fig. 4 illustrates the task mapping. The arrows indicate the flows generated by the applications' tasks. It is important to observe the disturbing induced by the PC_{app} applications.

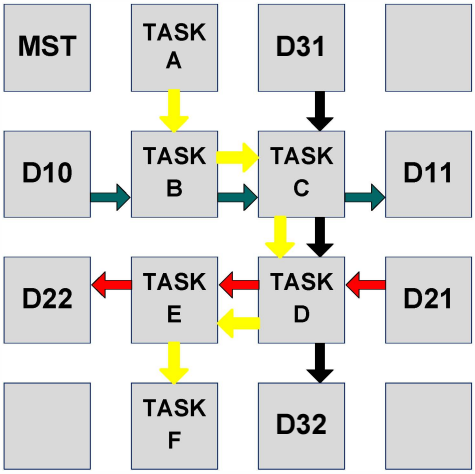

Fig. 4 – Task mapping. Tasks A to F correspond to the *pipeline* application. Flows D10→D11, D21→D22 and D31→D32 disturb the *pipeline* application (MST → manager processor)

Fig. 5 illustrates the behavior of Scenario 1, for profiling purposes. The X-axis corresponds to the simulation time, measured in clock cycles. The left Y-axis corresponds to the throughput, in bits per 10,000 clock cycles[1], and the right Y-axis to the PE frequency, as a percentage of the reference frequency.

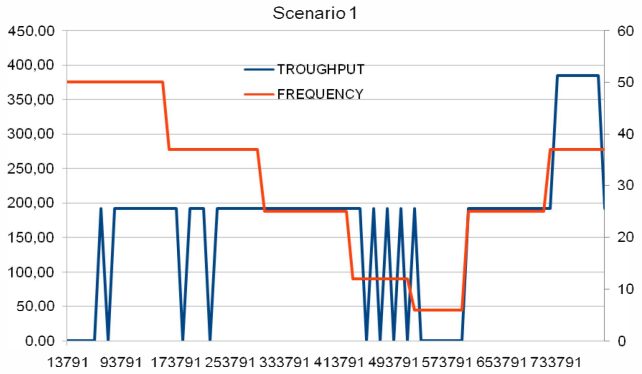

Fig. 5 - Scenario 1, application *pipe* without disturbing tasks, used for profiling purposes.

[1] 10,000 clock cycles is the monitors evaluation window. 200 bits per 10,000 clock cycles, with a frequency of 100 MHz, correspond to a throughput of 2 Gbps.

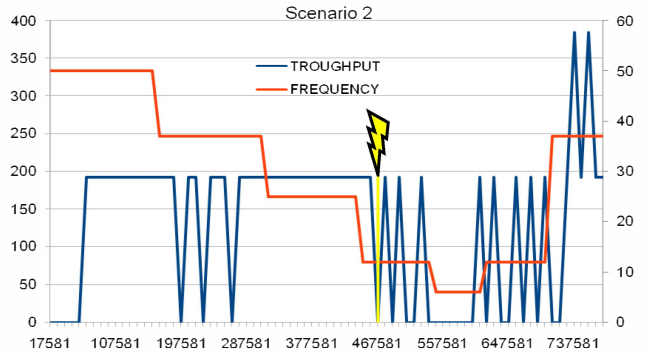

Fig. 6 – Scenario 2, *pipe* application running together with one PC_{app}, without adaptivitiy.

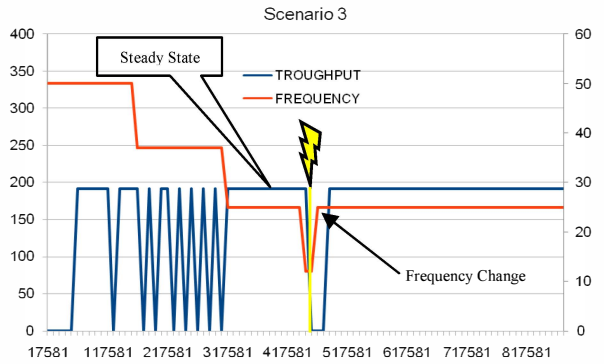

Fig. 7 - Scenario 3, *pipe* application running together with one PC_{app}, with adaptivitiy.

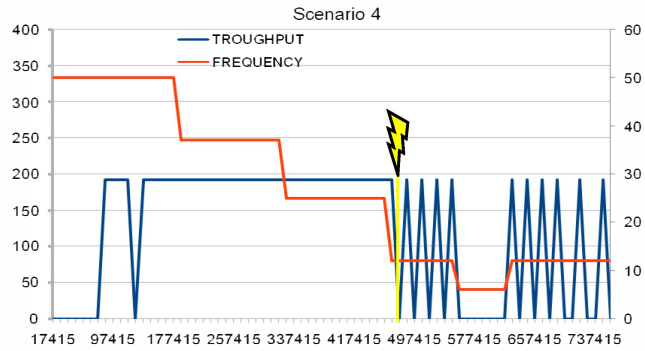

Fig. 8 - Scenario 4, *pipe* application running together with two PC_{app}, without adaptivitiy.

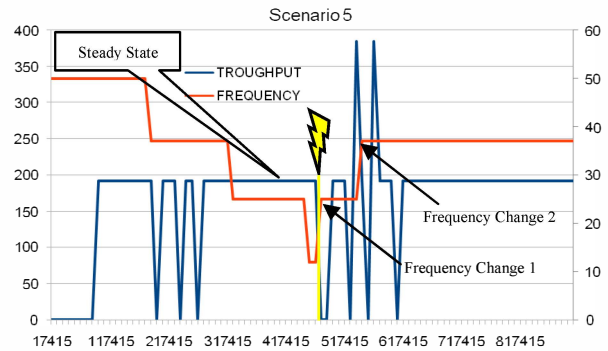

Fig. 9 - Scenario 5, *pipe* application running together with two PC_{app}, with adaptivitiy.

It is important to follow in the graph of Fig. 5 the frequency changes, and the corresponding throughput. The DFS controller set the initial frequency to 50% of the reference frequency. The frequency continuously decreases, with the goal to keep the communication pipes with an average occupancy. At a given point the frequency is too low and the communication queues begin to get empty. At this moment, the DFS Controller reacts and increases the frequency to reestablish nominal queue occupancy. At the end of the simulation, there is no more data to be consumed, leading the DFS Controller to increase the frequency. From this scenario, it is possible to fix the throughput constraint at 1.92 Gbps.

In scenario 2 the disturbing traffic starts at 470,000 clock cycles, as highlight in Fig. 6. The effect of the disturbing traffic is observed after the minimal frequency value. In Fig. 5 the throughput is reestablished, while in Fig. 6 the throughput oscillates and not reaches the defined constraint.

It would be expected a similar behavior between Fig. 6 and Fig. 7 in the beginning of the simulation, up to the start of the disturbing traffic. The observed difference in the throughput comes from the additional processing load of the monitors, which evaluate when the application reaches a steady state.

The *pipe* application reaches a steady state around 420 Kcycles, and the DFS scheme starts to be controlled by the microkernel. The *pipe* application starts to be disturbed at the same time as the scenario 2, around 470 Kcycles. The QoS module verifies that there is an increase of the QoS violations, and then fires the DFS adaptability process switches from the original DFS policy to the new policy which change the PE frequency and sends a back-propagated frequency change message. After the frequency change (depicted in Fig. 7), the application throughput remains stable and in compliance with QoS constraints (1.92 Gbps).

In scenario 4 the *pipe* application is disturbed by 3 other applications around 497 Kcycles, as highlighted in Fig. 8. As in Fig. 6 the throughput oscillates after the minimal frequency value, and not reaches the defined constraint.

In the scenario 5 (Fig. 9), the *pipe* application reaches a steady state around 450 Kcycles, and the DFS begins to be controlled by the microkernel. The *pipe* application starts to be disturbed at the same moment, around 497 K cycles. The QoS module verifies that there is an increase of the QoS violations, and then fires the DFS adaptability process which changes the PE frequency. After the first frequency change (highlighted in Fig. 9), the application throughput remains unstable and the QoS module continues to register QoS violations. This is due to the disturbing traffic. After a few more QoS violations, the QoS module fires a second DFS adaptability, which again changes the PE frequency. After the second frequency change (highlighted in Fig. 9), the application throughput reaches the QoS constraints (1.92 Gbps).

These scenarios show that the proposed DFS self-adaptable scheme can manage the QoS constraints of a given application, without a central manager, ensuring scalability to the proposal.

VI. Conclusions and Future Work

MPSoCs are commonplace today, as in cell phones and media centers. As an example, new Samsung state-of-the-art mobile devices are embedded with 8-cores CPU and a dedicated GPU. For these types of devices, performance and energy consumption must be managed at run-time.

This paper presented a QoS adaptability mechanism, based on a DFS technique. The proposal is an alternative to very complex DVFS techniques associated to QoS schemes. The main features of the proposed solution include: (*i*) the overall energy consumption reduction enabled by the DFS in the whole MPSoC; (*ii*) the adaptation of the application's tasks frequencies, at run-time, by monitoring QoS constraints in the application's output task, (*iii*) possibility to run the application on a expected QoS range (minimal and maximal limit values).

Results show the effectiveness of the approach. However, the works must be extended in several directions: (*i*) include other monitored performance parameters, such as latency and jitter; (*ii*) evaluate the proposed method when several tasks execute in the same PE; (*iii*) evaluate the proposal with real benchmarks; (*iv*) associate other adaptability techniques, such as task migration, enabling to meet QoS constraints when the DFS reach its limits.

Acknowledgments

Fernando Moraes is supported by CNPq project 302625/2012-7, and CAPES projects CAPES-COFECUB 708/11 and AEX 8418/13-6.

References

[1] Howard, J.; Dighe, S.; Hoskote, Y. “A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS”. In: ISSCC, pp.108-109, 2010.

[2] Herbert, S.; Marculescu, Dx. "Variation-aware dynamic voltage/frequency scaling". In: HPCA, pp. 301-312, 2009.

[3] Garg, S.; Marculescu, D.; Marculescu, R.; Ogras, U."Technology-driven limits on DVFS controllability of multiple voltage-frequency island designs: A system-level perspective". In: DAC, pp. 818-821, 2009.

[4] da Rosa, T. R.; Larrea, V.; Calazans, N.; Moraes, F. G. “Power consumption reduction in MPSoCs through DFS.” In: SBCCI, pp. 1-6, 2012.

[5] Goossens, K.; Molnos, A.; Ambrose, J.A.; Nelson, A.; Stefan, R.; Cotofana, S. "A composable, energy-managed, real-time MPSOC platform," OPTIM, pp.870,876, 2010.

[6] Mansouri, I.; Clermidy, F.; Benoit, P.; Torres, L., "A run-time distributed cooperative approach to optimize power consumption in MPSoCs," SOC Conference (SOCC), pp.25,30, 2010.

[7] Garg, Siddharth; Marculescu, D.; Marculescu, R., "Custom Feedback control: Enabling truly scalable on-chip power management for MPSoCs," ISLPED, pp.425,430, 2010.

[8] Fattah, M.; Daneshtalab, M.; Liljeberg, P.; Plosila, J. “Exploration of MPSoC Monitoring and Management Systems”. In: ReCoSoC, pp. 1-3, 2011.

[9] Takase, H.; Gang Zeng; Gauthier, L.; Kawashima, H.; Atsumi, N.; Tatematsu, T.; Kobayashi, Y.; Kohara, S.; Koshiro, T.; Ishihara, T.; Tomiyama, H.; Takada, H., "An integrated optimization framework for reducing the energy consumption of embedded real-time applications," ISLPED, pp.271,276, 2011.

[10] Carara, E.A.; de Oliveira, R.P.; Calazans, N.L.V.; Moraes, F.G. "HeMPS - a framework for NoC-based MPSoC generation". In: ISCAS, pp. 1345-1348, 2009.

[11] Chelcea, T.; Nowick, S. "A low latency FIFO for mixed-clock systems". In: Computer Society Workshop on VLSI, pp. 119-126, 2000.

[12] Mello, A.; Tedesco, L.; Calazans, N.; Moraes, F. "Evaluation of Current QoS Mechanisms in Networks on Chip". In: SoC, 2006, pp. 115-118.