Application-Specific Power-Aware Workload Allocation for Voltage Scalable MPSoC Platforms

Abstract— **In this paper, we address the problem of selecting the optimal number of processing cores and their operating voltage/frequency for a given workload, so to minimize overall system power under application-dependent QoS constraints. Selecting the optimal system configuration is non-trivial, since it depends on task characteristics and system-level interaction effects among the operating cores. For this reason, our QoS-driven methodology for power aware partitioning and frequency selection is based on functional, cycle-accurate simulation on a virtual platform environment. The methodology, being application-specific, was demonstrated on the DES (Data Encryption System) algorithm, representative of a wider class of streaming applications with independent input data frames and regular workload.**

I. Introduction

Many state-of-the-art or envisioned Multi-Processor Systems-on-Chip (MPSoCs) adopt the symmetric multi-processing paradigm [15]. This is due to the evolving micro-architecture of integrated cores, to the extension of their instruction set architecture in the direction of DSPs and to the increasing levels of integration made available by technology scaling. The design and implementation of MPSoCs is characterized by conflicting requirements of the ever increasing demand for higher performance and stringent power budgets. Circuit-level power minimization techniques can be used to address the power concern, including clock gating [18], dynamic voltage and frequency scaling (DVFS) [20] and low voltage design with variable/multiple V_{dd}/V_{th} control [16]. Furthermore, CMOS technology progressively allows an increasing number of voltage and clock domains to be specified on the same chip (see the *voltage islands* concept in [12]).

Lowering supply voltage reduces power quadratically but also results in a performance degradation, which translates into a reduction of the processor operating frequency at which functional correctness is guaranteed. Therefore, voltage scaling is usually associated with frequency scaling and vice versa.

In the new MPSoC domain, the problem of voltage and frequency selection cannot be optimally solved if we consider each processor in isolation. First, tasks running on individual cores are tightly related, since they are often the result of an application partitioning process, based on a specific workload allocation policy which creates task interdependencies.

Second, system-level interaction among the variable-voltage/frequency cores might induce non-trivial effects on global system performance and energy metrics. As an example, system performance is a non-additive metric, but strongly depends on the inter-processor synchronization mechanism and on the interaction on the system bus of the traffic patterns generated by cores running at different speeds.

However, in a parallel computing domain like MPSoCs, workload allocation is another degree of freedom for system power minimization. DFVS and extraction of task level parallelism should be jointly addressed in a global power minimization framework. Here, the trade-off to span is between the number of concurrent processors and the power overhead they introduce in the system, which is a function of their clock speed. For instance, the same application-dependent throughput constraint could be met by means of N processors working at speed X or by sharing the workload among $N+M$ concurrent processors working at reduced speed $X^{'}$.

In this paper, we take a semi-static approach to the frequency/voltage selection problem. Processor configuration is statically determined and set at design time. However, at large granularity idle periods, frequency settings can be updated as a function of changed system conditions. This approach can be applied to application domains characterized by regular and highly predictable workloads, with minimum run-time fluctuations. Baseband processing in wireless modems, encryption engines, digital image filtering and many signal processing functions are examples thereof.

Although we consider a maximum number of available processors, we assume that not all of them must be necessarily allocated to the execution of a given application. We introduce a complete QoS-based methodology that provides the optimal number of processing cores for a given scalable workload and their individual frequency/voltage settings in such a way to minimize system power while meeting application throughput constraints.

Our methodology for processor allocation and frequency/voltage selection is simulation based. We want to overcome the limitations of previous works, which proposed theoretical and highly abstract models without validation on real platforms or on functional, timing accurate MPSoC simulation tools. In contrast, we deployed a virtual platform [21], enhanced with hardware extensions for variable frequency/voltage cores, for developing the allocation and frequency selection methodology and for validating our approach. As such, our methodology is strongly related to the specific workload. We therefore restricted our analysis to the optimization of a parallel, highly scalable DES encryption algorithm, deriving a methodology and drawing conclusions that can be extended to the whole class of applications DES algorithm belongs to, namely streaming applications with uncorrelated input data frames.

This paper is structured as follows. Section 2 reports previous work while the virtual platform environment is described in Section 3. Section 4 and 5 present DES algorithm and problem formulation. Section 6 explains our methodology, whose results are reported in Section 7.

II. Related Work

A survey of techniques for system level power optimization is reported in [1], [2]. The issue of voltage/frequency selection for single-processor systems is a mature research field: many run-time dynamic techniques have been proposed [3], [4], and validation tools [5] and hardware [23] are available.

On the contrary, in the multi-processor system domain, many approaches based on theoretical analysis and abstract simulation have been proposed, but an accurate validation of the effectiveness of these techniques is still in its early stage.

In [6] the problem of minimizing power of a multi-processor system using multiple variable supply voltages is modelled as a mixed integer non-linear programming optimization problem. The work in [7] points out that minimizing communication power without considering computation may actually lead to higher energy consumption at the system level. An analytical approach is taken in [8] to assign single optimal voltage to each processor present in an application-specific heterogeneous multi-processor system after allocation and scheduling have been performed. A heuristic to address the problem of energy-efficient voltage scheduling of a hard real-time task graph with precedence constraints for a multi-processor environment is presented in [9], but it is limited to dual voltage systems. The algorithm introduced in [11] targets power utilization and performance of multi-processor systems wherein parameters such as operating voltage, frequency and number of processors can be tuned. The energy-aware scheduling algorithm presented in [17] consists of a design-time phase, which results in a set of Pareto-optimal solutions, and of a run-time phase, that uses them to find a reasonable cycle budget distribution for all of the running threads. The effect of discrete voltage/speed levels on the energy savings for multi-processor systems is investigated in [14], and a new scheme of slack reservation to incorporate voltage/speed adjustment overhead in the scheduling algorithm is also proposed. The approach to energy minimization in variable voltage MPSoCs taken in [13] consists of a two phase framework that integrates task assignment, ordering and voltage selection.

With respect to previous work, our contributions are: 1) the joint solution of processor allocation and frequency/voltage setting; 2) a novel algorithm for efficient construction of the Pareto frontier for selecting the optimal system operating points based on throughput and utilization constraints; 3) validation on a full-system functional and power simulation for a real application case study.

III. Virtual Platform Environment

We carried out our analysis within the framework of the SystemC-based MPARM simulation platform [21]. Figure 1 shows a pictorial overview of the simulated architecture. It consists of a configurable number of 32-bit ARMv7 processors. Each processor core has its own private memory, and a shared memory is used for inter-processor communication. Synchronization among the cores is provided by hardware semaphores implementing the *test-and-set* operation. The system interconnect is a shared bus instance of the STBus interconnect from STMicroelectronics. The software architecture consists of an embedded real-time operating system called RTEMS [24], which natively supports synchronization and inter-task communication primitives.

The virtual platform environment provides power statistics leveraging technology-homogeneous power models made available by STMicroelectronics for a 0.13 μ m technology for ARM cores, caches, on-chip memories and the STBus.

Support for Variable Frequency Cores. The virtual platform has been extended to support different working frequencies for each processor core. For this purpose, additional modules were integrated into the platform, namely a variable clock tree generator, programmable registers and a synchronization module.

The clock tree generator feeds the hardware modules of the platform (processors, buses, memories, etc.) with independent and frequency scaled clock trees. The frequency scaled clock trees are generated by means of frequency dividers (shift counters), whose delay can be configured by users at design-time. A set of programmable registers has been connected to the system bus to let the operating system or a dedicated hardware module (monitoring system status) select the working frequencies. Each one of these registers contains the integer divider of the baseline frequency for each processor.

Scaling the clock frequency of the processors creates a synchronization issue with the system bus, which is assumed to work at the maximum frequency. The processing cores and the bus interface communicate by means of a hand-shaking protocol which assumes the same working frequency at both sides. Therefore, a sychronization module was designed, containing two dual-ported FIFOs wherein data and addresses sent by the bus interface to the processor and vice versa are stored. This module works with a dual clock: one feeding the core side and one feeding the bus interface side. Finally, the module also takes care of properly interfacing processor to bus signals, and a dedicated sub-module is implemented for this purpose. In Figure 1, the hardware extensions for frequency-scaled cores have been shaded. The Maximum STBus frequency of 200MHz was kept as the maximum processing core frequency, to which frequency dividers were applied. The scaling factor for the power supply was derived from [19].

IV. Workload Allocation

A. DES Dataflow

DES performs two main operations on input data, controlled by the key: bit shifting and bit substitution. By doing these operations repeatedly and in a non-linear way, the final result cannot be used to retrieve the original data without the key. DES works on chunks of 64 bits of data at a time. Each 64 bits of data is iterated on from 1 to 16 times (16 is the DES standard). For each iteration, a subset of the key is fed into the encryption block, that performs several different transforms and non-linear substitutions. More details can be found in [22].

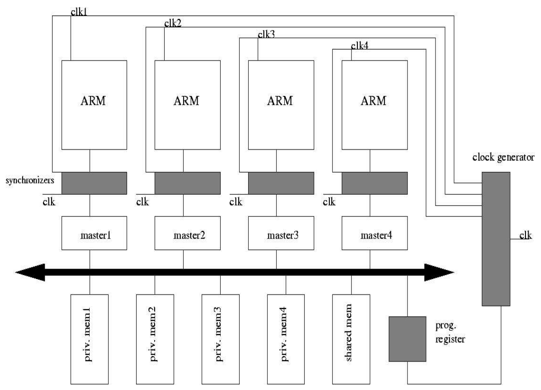

Fig. 1. MPSoC platform with hardware support for frequency scaling.

B. Mapping DES on MPSoC Platform

DES algorithm matches the master-slave workload allocation policy. DES encrypts and decrypts data using a 64-bit key. It splits input data into 64-bit chunks and outputs a stream of 64-bit ciphered blocks. Since each input element (called *frame*) is independently encrypted from all others, the algorithm can be easily mapped onto a variable number of cores. Moreover, DES poses a balanced workload to the processors, since the execution time is almost independent on input data. This consideration will be important when we will address core clock scaling later in this paper.

In the parallelized version of DES, we define three kinds of tasks. An initiator task (*producer*) dispatches 64-bit blocks together with a 64-bit key to n calculator tasks (referred as *working* tasks) for encryption. Initiator task and working tasks are allocated to different processing elements (PEs). A buffer in shared memory is used to exchange data. Since frames are uncorrelated from each other, computation of working tasks can be carried out in parallel by running multiple task instances on different slave processors. Here slave processors just need to be independently synchronized with the producer, which alternatively provides input data to all of the slave tasks. Finally, a collector tasks (*consumer*) does exist, which reconstructs an output stream by concatenating the ciphered blocks provided by the working tasks. It is allocated onto another PE and communicates with workers by means of output buffers.

Both input and output buffers are located in shared memory. Each one of them is implemented using two queues, so that one queue can be write-accessed by the producer while the other one is read-locked by the worker. The same holds for the output buffer. Moreover, in case of multiple workers, each one has its own input and output buffer. In brief, the streaming application is mapped onto the platform as a three stage pipeline, where the intermediate stage can be made by an arbitrary number of multiple parallel tasks. The overall system model is described in Figure 2.

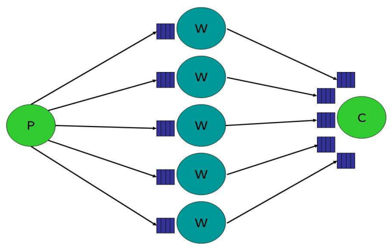

Fig. 2. DES workload allocation policy.

V. Problem Formulation

The ultimate objective of this work is to find the optimal number of parallel workers (and a corresponding number of slave processors) to achieve a given throughput (in frames/sec), and to set the operating frequencies of the cores in the system (producer, consumer and workers) so to minimize overall power consumption. We decided to make producer and consumer work at the same speed, in order to keep the system stable. Moreover, since the workload of the parallel workers is self-balanced due to the intrinsic DES characteristics, we assume they are also running at the same speed.

The optimization problem can be thus formulated as follows. For a given throughput (T), we need to find a couple (N_{WK}, S), where N_{WK} is the number of workers and S is the scaling factor between producer (or consumer) speed and workers' speed. If f_{WK} is the clock frequency of workers and f_{PR} is the clock frequency of producer/consumer, we obtain $f_{WK} = S \cdot f_{PR/CN}$.

Given this formulation, the optimization problem can be solved by searching for those system configurations that minimize the following cost function:

$$P = F(N_{WK}, f_{WK}, f_{PR}, f_{CN}), \tag{1}$$

where P is average power consumption of the whole MPSoC, and the following constraints hold:

$$f_{PR} = f_{CN} = f_{WK}/S; \tag{2}$$

$$N_{WK} \leq N_{MAX}; \tag{3}$$

where N_{MAX} is the maximum number of PEs available in the system, excluding the PEs allocated to producer and consumer tasks. In the following we will change this constraint to handle the case in which not all system PEs can be reserved for DES application.

Intuitively, configurations that minimize the cost function are the ones that minimize system idleness. As a consequence, the optimum scaling factor S, that is a function of N_{WK}, will be the one that balances the execution time of workers and producer/consumer. In general, since the computational effort required by each worker to produce an output frame is much larger than that required by the producer and consumer, S will be larger than one for low N_{WK} and lower than one for high N_{WK}. This is because producer and consumer tasks are essentially memory-bound, while working tasks are CPU-bound. For a memory-bound task, throughput is less sensitive to frequency scaling, since this latter does not affect memory access times, that stay constant. Once S has been found, the absolute speed values are determined by the required throughput. As showed in the experimental result section, the solution to this problem is not trivial. Simple solutions that tune either N_{WK} or core frequencies in isolation are sub-optimal.

Handling Bus Effects. In general, we cannot assume that all system resources are available for the target application. This fact has a double effect on the previously defined optimization problem. First of all, not all of the PEs are available, so that equation 3 changes as follows:

$$N_{WK} \leq N_{FREE}; \tag{4}$$

where N_{FREE} is the number of free PEs. Note that our methodology will indicate whether it is power efficient to use all of the N_{FREE} PEs or a lower number of them, together with the proper frequency settings, for the considered workload. Since the bus is partially occupied by interfering traffic generated by applications running on the busy PEs, this will affect the performance of the communication among DES tasks. As a results, the cost function depends on traffic conditions. We can characterize bus traffic by means of two parameters: ρ is the available bus bandwidth, while σ is the average burst size of the interfering traffic. We can then rebuild equation 5 as follows:

$$P = F(N_{WK}, f_{WK}, f_{PR}, f_{CN}, \sigma, \rho). \tag{5}$$

VI. QoS-Driven Optimization Strategy

Our power optimization framework consists of a two step process. First, we statically perform a smart exploration to find Pareto-optimal configurations that minimize the cost function in a power/throughput design space. We perform this exploration by varying traffic parameters (σ, ρ) in a discrete range of possible values. Then, we store these configurations in a three-dimensional look-up table indicized by σ, ρ and N_{FREE} that will be used at run-time in a semi-static way to maintain QoS under varied traffic conditions.

A. Design-time Smart Design Space Exploration

We explore the throughput/power design space from higher to lower throughput configurations, in an attempt to find the Pareto curve. The smart exploration algorithm has to be repeated for an increasing numbers of workers. The method tries to find the optimal scaling factor between producer/consumer and workers frequency (a configuration) that leads to minimum idleness in processing cores operation. This is achieved through successive frequency scaling steps. For each explored configuration in the design space, a simulation run is performed to compute the corresponding level of power dissipation of the system.

The methodology starts by simulating the configuration where all of the PEs (both workers and producer/consumer) run at the maximum speed, then derives two other configurations by scaling PR/CN or WK frequency. From these new configurations, we generate other configurations using the same scaling rule. Clearly, since we scale the frequency of PEs, at each step we move to lower throughput regions. However, in order to reduce the number of simulation runs, we defined an optimized version of the algorithm that allows to reduce the number of design points to be explored.

The smart exploration is based on the following intuitions. First, a configuration is said to be ”dominated” if there is at least another configuration that provides a higher or equal throughput with lower power. Configurations that turn out to be non-dominated once the smart search procedure completes are those belonging to the Pareto curve. In practice, a higher throughput with respect to the one provided by a dominated configuration can be provided at a lower power cost. Second, it is worth observing that dominated configurations cannot generate (using the scaling rule explained before) non-dominated configurations that cannot be obtained by other non-dominated configurations already found. Hence, when we generate two new configurations, we always check if one of them is dominated. If this is true, the dominated configuration is discarded and thus all other configurations derived from it are not explored.

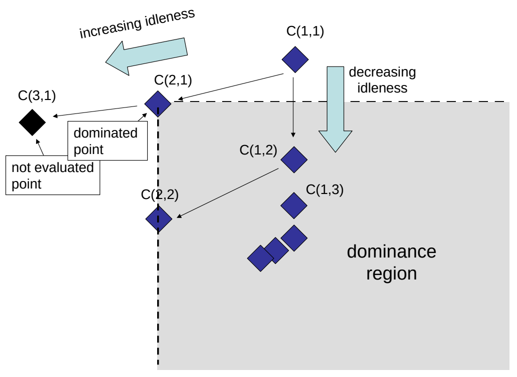

Fig. 3. Example of smart exploration.

The observation can be justified as follows (refer to Figure 3). Let us first point out that derived configurations can have lower or equal throughput w.r.t. their parent configurations, being obtained through frequency scaling. Then, let us indicate with (i, j) the scaling factor of PR/CN and WK respectively with respect to maximum system frequency. Assume now to generate two new configurations C(1,2) and C(2,1) from the starting one C(1,1). If a configuration is dominated, for example C(2,1), this means that its power consumption is higher and the throughput is lower than another one already explored, namely C(1,2). In practice, C(2,1) corresponds to a less efficient operating point in which the amount of power wasted in idle cycles is increased w.r.t. C(1,1). Since in C(2,1) we have scaled the PR/CN frequency, we deduce that the performance bottleneck in C(1,1) was represented by the producer/consumer, and that by moving to C(2,1) we have made the pipeline even more unbalanced. In contrast, C(1,2) reduces workers frequency, and goes in the direction of balancing the pipeline and minimizing idleness. Suppose now to generate two additional configurations from the dominated point C(2,1). Since we are further decreasing PR/CN frequency, we are moving away from the optimal ratio between PR/CN and WK speed, increasing the power wasted by WKs in idle cycles.

These considerations led to the smart exploration algorithm described in Figure 4. A ”Pareto list” stores configurations that pass the dominance check and are therefore Pareto points. A generic loop of this algorithm works as follows. From each configuration C(i,j) we generate two new configurations C(i+1,j) and C(i,j+1), and select the one with highest throughput, which we call C_{CURR} (current configuration). The other one is stored into a temporary list sorted for decreasing throughput only if it passes a dominance check against C_{CURR}, against all previously stored configurations in the temporary list and the ones that are already in the Pareto list. Even though it passes the dominance check, this point cannot be put in the Pareto list since there could be other dominating configurations with intermediate throughput values generated by C_{CURR}.

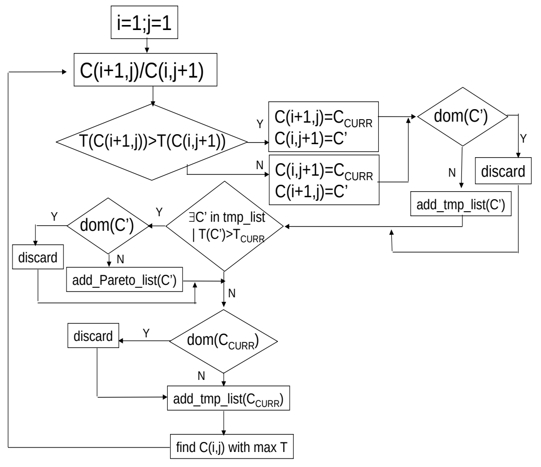

Fig. 4. Flow diagram of the smart exploration algorithm.

Before examining C_{CURR}, the temporary list is searched in order to find if there are stored configurations C' that need to be examined first because they have a higher throughput than C_{CURR}. For each one of them, we perform a dominance check and eventually add them in the Pareto list. Finally, we perform the dominance test on C_{CURR}. If two configurations in the Pareto list have the same throughput, the one with the lowest power replaces the other one. New configurations are generated starting from the last one added to the Pareto list.

At the end of this step, we obtain a Pareto curve, and the procedure has to be repeated for a different number of workers. Finally, we obtain an overall Pareto curve showing the optimal number of workers and the operating frequency of the cores for a given throughput. In general, the overall Pareto curve (PAR^O) is obtained by the composition of all Pareto curves with $N_{WK} \leq N_{FREE}$. Smart exploration is also performed for different traffic conditions, thus allowing semi-static resource allocation based on the levels of bus traffic.

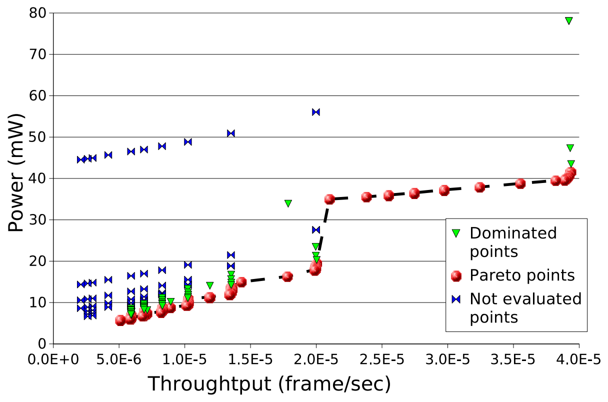

Fig. 5. Pareto curve with one worker.

B. QoS-Oriented Semi-Static Workload Allocation

Once Pareto-optimal configurations have been statically determined in the previous step, they will be stored in a three-dimensional look-up table having traffic parameters and N_{FREE} as indexes. The table returns the overall Pareto curve for N_{FREE} maximum processors, which gives the optimal configuration for a given throughput constraint.

At run-time, system conditions might change as an effect of events occurring at a large time granularity, such as freed PEs or newly admitted applications in the system or abrupt changes of bus traffic. In this case, the amount of resources allocated for our application must be recomputed by looking at the table. Traffic parameters and number of free cores might be retrieved from the environment (i.e. an operating system) or through monitoring. Since we do not store in the table all possible values of σ, ρ but only a discrete set, it is possible that run-time values of traffic parameters do not belong to this set. In this case, Pareto-optimal points must be obtained through interpolation of stored configurations.

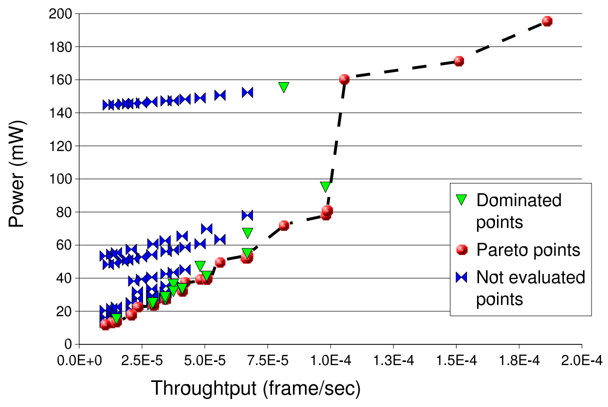

Fig. 6. Pareto curve with five workers.

VII. Results

In this section we first show power/performance Pareto curves obtained through smart design space exploration that are used to fill in the configuration table described in previous section. We consider power contributions of all system components. Then, we compare our results with those provided by alternative approaches.

A. Pareto Optimal Configurations

Pareto-optimal configurations are shown in Figure 5 for one worker (PAR^1) and in Figure 6 for 5 workers (PAR^5). Both figures outline the effectiveness of the smart design space exploration process. It is evident that a large number of configurations have not been evaluated, thus cutting down on simulation runs. Points that have been discarded because they are dominated are also shown. Let us observe the one worker case in Figure 5. The algorithm starts from the upper rightmost point corresponding to $f_{PR/CN}$ = f_{WK} = f_{MAX}. It immediately discards the upper points in the plot and moves down vertically until it finds the first point of the Pareto curve. We observed that the discarded point correspond to f_{WK} scaling. This means that in the starting configuration the single worker is the bottleneck and by scaling down f_{WK} we make the system more unbalanced. In contrast, by scaling $f_{PR/CN}$, we get a reduction of power consumption with constant throughput until PR/CN become the bottlenecks. This corresponds to a Pareto optimal configuration, and the relative scaling factor between $f_{PR/CN}$ and f_{WK} frequencies minimizes the idleness. All configurations with same throughput but higher power are then discarded. Other points that we obtain by scaling PR/CN from here increase the idleness, but they are still Pareto optimal since they are not dominated by other configurations. The reason for this is the discrete number of available frequencies.

With a larger number of workers (greater than four), Pareto curves become similar to Figure 6, where a case with 5 workers is represented. Here, PR/CN are the bottlenecks in the starting configuration. Although we correctly scale workers frequency to balance the system, the scaling granularity is so coarse (scaling of 5 workers at a time) that after one scaling step the workers become the bottleneck. Therefore the identification of Pareto points here is much less intuitive than in Figure 5.

The overall Pareto curve ($PAR^O(N_{FREE})$) for a maximum number of available workers is obtained by comparing the Pareto-optimal configurations for the different numbers of workers. The curve is shown in Figure 7 for $N_{FREE} = 5$ workers. We can observe that using all of the workers is not always the most power-efficient solution.

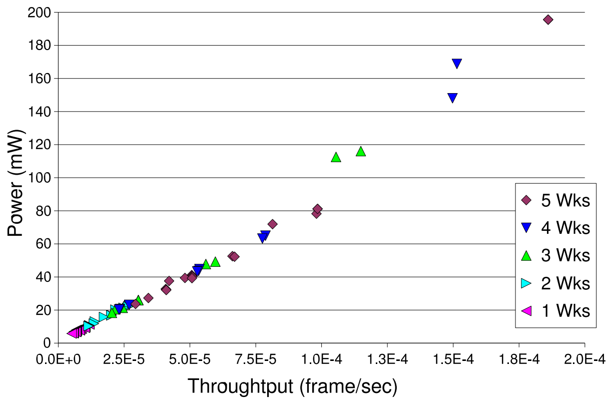

Fig. 7. Overall Pareto curve with five available workers.

Our analysis showed that for a larger number of workers, second-order effects come into play. We in fact observed an increase of the achievable throughput until the number of workers is less or equal to eight, which is the last point with no diminishing returns. In fact, with more than eight WKs the bus saturates when high throughputs have to be delivered, as detailed in Figure 8. For this reason, when considering the overall Pareto curve $PAR^O(N_{MAX})$ with $N_{FREE} \geq 9$, configurations that do not use all of the workers are more efficient.

Interfering traffic effects are shown in Figures 9 and 10. All the points correspond to configurations wherein producer, consumer and workers work at the same speed, which is maximum on the rightmost part of the plot, and scaled as we move toward lower throughput values. Figure 9 highlights the impact of reduced available bus bandwidth on DES performance for the 1 worker case. The throughput theoretically achievable by configurations on the rightmost part cannot be actually provided because of a larger impact of bus contention. This effect is not observed in the leftmost part, since there is a lower frequency of bus accesses to deliver lower throughput and since the processors are working at a lower speed than the bus.

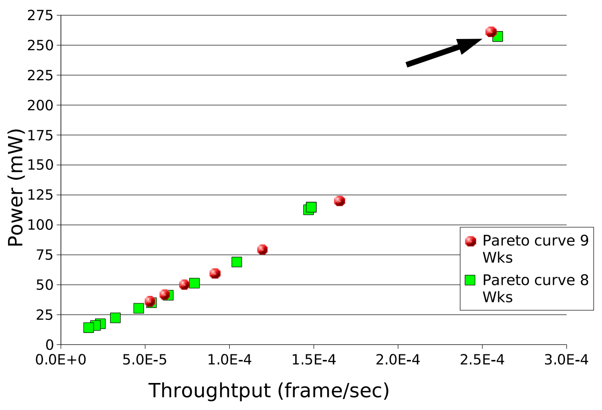

Fig. 8. Bus saturation effect.

Figure 10 instead shows the impact of average burst size of the interfering traffic (σ), keeping the interfering bandwidth ρ constant. We show that the impact on throughput is larger for smaller but more frequent bursts, and that configurations providing high throughput values are more sensitive.

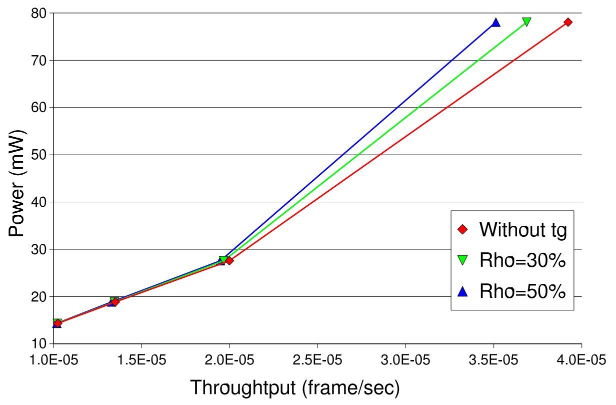

Fig. 9. Effect of interfering traffic bandwidth. ρ is the bandwidth occupancy of interfering traffic with respect to maximum bus bandwidth.

B. Efficiency Comparison

In Figure 11 we compare our optimal solutions for a 2 workers case ($N_{FREE} = 2$) with an alternative policy which always uses the maximum number of available workers and scales down all processor frequencies to get a lower throughput. With our power-aware methodology, we can cut down power by 30% for high throughput values, since we are able to reduce idleness. For lower throughput values the savings are smaller but still our configurations are more power efficient.

The effectiveness of the power-aware allocation has been also compared with a policy with no voltage/frequency scaling. In this case, a lower throughput can be achieved by employing a lower number of processors. Results are reported in Figure 12 for $N_{FREE} = 5$. The comparison highlights that using our strategy allows to save 50% of power for lower throughput values, since we again scale frequencies to reduce idleness.

VIII. Conclusion

We presented a QoS-driven methodology for optimal allocation and frequency selection. Our methodology is based on functional simulation and full system power estimation. It is demonstrated on the DES algorithm, representative of a wider class of streaming applications with

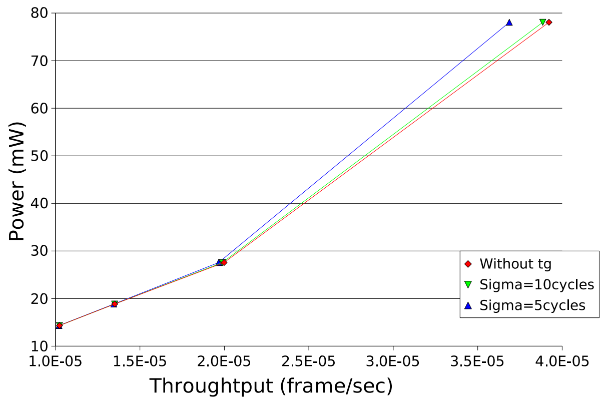

Fig. 10. Effect of burst size of interfering traffic.

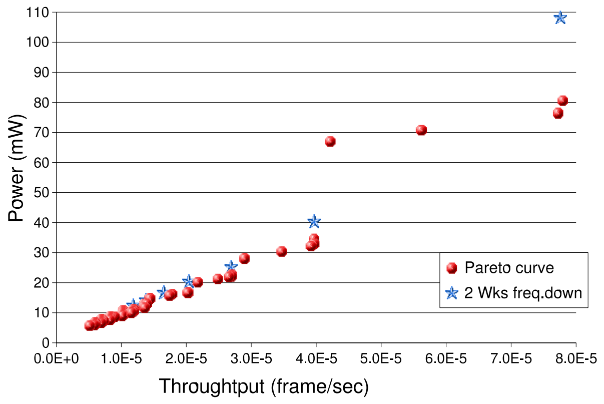

Fig. 11. Comparison of power-aware allocation strategy with no workers tuning policy.

independent input data frames and regular workloads. We have showed the savings in terms of needed simulation runs and the efficiency with respect to alternative approaches.

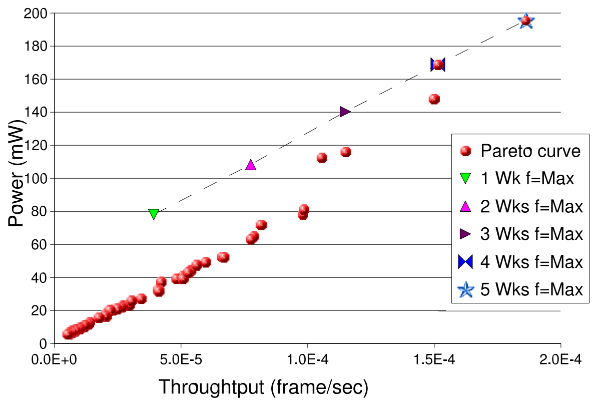

Fig. 12. Comparison of power-aware allocation strategy with no frequency scaling policy.

REFERENCES

[1] L. Benini, A. Bogliolo, and G. De Micheli. “A survey of design techniques for system-level dynamic power management”. *IEEE Trans. on VLSI Systems*, pages 299–316, June 2000.

[2] N. Jha. “Low power system scheduling and synthesis”. *IEEE/ACM Conf. on CAD*, pages 259–263, 2001.

[3] W. Kwon and T. Kim. “Optimal voltage allocation techniques for dynamically variable voltage processors”. *IEEE Trans. on VLSI Systems*, pages 125–130, June 2003.

[4] P. Pillai and K. Shin. “Real-time dynamic voltage scaling for low-power embedded operating systems”. *ACM SIGOPS 01*, pages 89–102, October 2001.

[5] A. Iyer and D. Marculescu. “Power effi ciency of voltage scaling in multiple clocks, multiple voltage cores”. *Int. Symposium of Computer Architecture*, pages 158–168, May 2002.

[6] L. Leung, C. Tsui, and W. Ki. “Simultaneous task allocation, scheduling and voltage assignment for multiple-processors-core systems using mixed integer nonlinear programming”. *ISCAS03*, V:309–312, May 2003.

[7] J. Liu, P. Chou, and N. Bagherzadeh. “Communication speed selection for embedded systems with networked voltage-scalable processors”. *CODES02*, pages 169–174, May 2002.

[8] A. Rae and S. Parameswaran. “Voltage reduction of application-specifi c heterogeneous multiprocessor systems for power minimisation”. *ASP-DAC*, pages 147–152, January 2000.

[9] D. Roychowdhury, I. Koren, C. Krishna, and L. Y.H. “A voltage scheduling heuristic for real-time task graphs”. *Int. Conf. on Dependable Systems and Networks*, pages 741–750, June 2003.

[10] D. Bertozzi and L. Benini. “Battery lifetime optimization for energy-aware circuits ”. *Low Power Electronics Design*, edited by C.Piguet, pages –, 2004.

[11] J. Suh, D. Kang, and S. Crago. “Dynamic power management of multiprocessor systems”. *Int. Parallel and Distributed Processing Symp.*, pages 97–104, April 2002.

[12] D. Lackey, P. Zuchowski, D. Bedhar, T.R. aand Stout, S. Gould, and J. Cohn. “Managing power and performance for systems-on-chip designs using voltage islands ”. *Int. Conf. on CAD*, pages 195–202, November 2002.

[13] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. *DAC03*, pages 183–187, 2003.

[14] D. Zhu, R. Melhem, and B. Childers. “Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems”. *IEEE Trans. on Parallel and Distributed Systems*, 14:686–700, July 2003.

[15] D. Pham et al. “The design and implementation of a fi rst generation CELL processor”. *IEEE/ACM ISSCC*, pp.184–186, 2005. July 2003.

[16] I. Hyunsik, T. Inukai, H. Gomyo, T. Hiramoto, and T. Sakurai. “VTC-MOS characteristics and its optimum conditions predicted by a compact analytical model”. *ISLPED01*, pages 123–128, August 2001.

[17] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins. “Energy-aware runtime scheduling for embedded-multiprocessor SOCs”. *IEEE Design and Test of Computers*, pages 46–58, 2001.

[18] L. Benini, P. Siegel, and G. De Micheli. “Automatic synthesis of gated clocks for power reduction in sequential circuits”. *IEEE Design and Test of Computers*, pages 32–40, December 1994.

[19] K. J. Nowka et al. “A 32-bit PowerPC System-on-a-Chip with support for dynamic voltage scaling and dynamic frequency scaling”. *IEEE JSSC* Vol. 37, no. 11, pp. 1441–1447, Nov. 2002.

[20] G. Qu. “What is the limit of energy saving by dynamic voltage scaling? ”. *IEEE/ACM Int. Conf. on Computer Aided Design*, pages 560–563, 2001.

[21] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, R. Zafalon, Analyzing On-Chip Communication in a MPSoC Environment, Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 2004, Paris, France, Feb 16-20, 2004, pp. 752-757 Vol. 2

[22] Federal Information Processing Standards Publication 46-2, ”“Announcing the Standard for DATA ENCRYPTION STANDARD (DES)””, `http://www.itl.nist.gov/fipspubs/fip46-2.htm`, Dec. 1993.

[23] Intel XScale technology, http://www.intel.com/design/intelxscale/

[24] RTEMS Home Page, http://www.rtems.com