An Efficient On-line Task Allocation Algorithm for QoS and Energy Efficiency in Multicore Multimedia Platforms

Francesco Paterna*, Andrea Acquaviva†, Alberto Caprara*,
Francesco Papariello‡, Giuseppe Desoli‡, and Luca Benini*

* DEIS, University of Bologna - Viale Risorgimento 2, 40136 Bologna, Italy
emails: francesco.paterna, alberto.caprara, luca.benini@unibo.it

† DAUIN, Polytechnique of Turin - Corso Duca Degli Abruzzi 24, 10129 Torino, Italy
email: andrea.acquaviva@polito.it

‡ ST Microelectronics - Via Tolomeo 1, 20010 Cornaredo (MI), Italy
emails: francesco.papariello, giuseppe.desoli@st.com

***Abstract*—The impact of variability on sub-45nm CMOS multimedia platforms makes hard to provide application QoS guarantees, as the speed variations across the cores may cause sub-optimal and sample-dependent utilization of the available resources and energy budget. These effects can be compensated by an efficient allocation of the workload at run-time. In the context of multimedia applications, a critical objective is to compensate core speed variability while matching time constraints without impacting the energy consumption.**

In this paper we present a new approach to compute optimal task allocations at run-time. The proposed strategy exploits an efficient and scalable implementation to find on-line the best possible solution in a tightly bounded time. Experimental results demonstrate the effectiveness of compensation both in terms of deadline miss rate and energy savings. Results have been compared with those obtained applying state-of-art techniques on a multithreaded MPEG2 decoder. The validation has been performed on a cycle-accurate virtual prototype of a next-generation industrial multicore platform that has been extended with process variability models.

I. Introduction

Variability in Multiprocessor Systems-on-Chips for multimedia appliances fabricated in sub-45nm CMOS technology impacts performance and energy efficiency [1, 3, 5, 14]. From the target application viewpoint, variability makes hard to provide a QoS guarantee because the speed variations across the cores may prevent an effective exploitation of the platform parallelism. Moreover, energy efficiency is affected by variations' impact on leakage and dynamic power.

Previous researches demonstrate that, by adjusting workload allocation, it is possible to compensate these effects [6, 10–13, 15]. In this context, variation-aware task allocation consists in finding the best allocation to minimizes a given target objective in presence of speed and power consumption variations. However, the main limitation of currently available task allocation strategies is that they cannot either guarantee optimality, or be applied online because of the large overhead to compute the optimal solution or the lack of scalability on the number of tasks.

In this paper we present an efficient and scalable on-line algorithm to compute a solution which allows the frame processing-time constraints to be met while minimizing the energy consumption. An efficient and scalable Linear Programming formulation of the allocation problem finds a candidate solution in linear time. In a second step, a Bin Packing (BP) algorithm is applied to find the best fit of tasks into the core cycle budgets defined by LP. The allocation provided by LP+BP is very close to the optimal one [11]. BP is a crucial step when the number of tasks per kernel is low. However, when the application is partitioned in a large number of similarly sized jobs, the LP step is sufficient to achieve high quality. Finally, a time-bounded branch-and-bound (B&B) exploration step tries to improve the best theoretical solution between those provided by LP+BP and LP. We discovered that B&B is necessary when the application is partitioned in a small number of large sized jobs.

We tested the effectiveness of the proposed strategy on a virtual cycle-accurate prototype of a next-generation industrial platform composed by a general purpose processor and an array of programmable VLIW hardware accelerators, enhanced with variability models and a plug-in to emulate memory mapped runtime monitors. We performed experiments on a MPEG2 decoder ported on the target platform considering a realistic range of parallelism, from 6 to 36 tasks. Results demonstrate that the proposed strategy can be applied frame-

Research activity supported by the European project REALITY FP7-2008-IST-1-216537 and MODERN ENIAC 2008-1 contract n. 21000

978-3-9810801-7-9/DATE11/

by-frame, thus improving the QoS and energy efficiency against state-of-the-art approaches.

II. Related work

We focus on minimizing energy under performance constraints, while several related works only focus on performance. In [6] the main purpose is to maximize performance only. Energy minimization is not considered also in [12, 15]. In [12] the task allocation is exploited to stochastically minimize the execution time of variability-affected multicore platforms. In [15] a scheduling approach, which assumes that the task executions are statistical, has been presented where the objective is to mitigate the impact of variations in a multicore platform.

In [13] authors propose different variation-aware task scheduling techniques aimed to different power/performance objectives. They consider independent tasks whose number is bounded by the number of the cores in the platform. The techniques differ on presence/absence of a DVFS support and uniform/non uniform frequency distribution among the cores. Without DVFS, a ranking approach is used. Choosing to minimize the power consumption, the authors assume to know the dynamic power consumption for each task to map the most consuming task on the lowest-power cores. To do that the algorithm ranks the task form that at maximum power consumption to that at minimum power consumption, whilst the cores are ranked from that at minimum power consumption to that at maximum power consumption, then the current task is mapped on the first available core.

In this work we assume that per-core voltage setting is not available, because we focus on embedded platforms featuring small processing elements where overheads of power switches and voltage regulators is not affordable [10]. We assume DFS and shutdown support only and we set each core at its own maximum supported clock frequency. As shown in the experimental result section, the proposed policy outperforms the ranking ones presented in [13], both in terms of QoS and energy efficiency for a realistic multimedia application.

In [11] authors studied an off-line ILP-based policy and an on-line one based on Linear Programming and Bin Packing. LP+BP can run on-line using a set of LP pre-solved stored into a look-up-table. In this work we give the following novel contributions: i) We propose a scalable and efficient policy that exploits three different techniques to find an on-line solution that improves the LP+BP one in most of the cases; ii) We provide a scalable implementation of the LP solution; iii) We applied and characterized the proposed policy on a realistic MPEG2 benchmark.

III. Platform and Variability Model

A. Variability model

To model variability at the system level, we start from the characterization of the variation effects on circuit delay, dynamic energy and leakage power [9] which have been obtained by the Variability-Aware Modeling tool (VAM) [7]. VAM starts from gate-level netlist and detailed library characterization to build statistical models at core level for these quantities. We configure the variability status of the core i through its own value in the tri-dimensional space where a variability configuration is characterized by a certain longest path delay, leakage power, and dynamic power: $(lpd^i, P^i_{lkg}, P^i_{dyn})$; these quantities have been chosen because of their clock frequency independence. To build a variability affected platform, we start from an N-core platform with N nominal homogeneous cores of the 32-nm CMOS technology platform. We then extract a variability configuration value from the global range, which models the die-to-die effect. Finally, we extract N values from the local range that models the within-die effect and will be centered onto the global value previously extracted. These values will be associated to the N cores of the platform. In Table I we show the ranges used for the experiments in this paper.

	global variability		local variability	
	min	max	min	max
lpd	-17,70%	22,87%	-15,34%	16,63%
Plkg	-17,58%	16,90%	-21,18%	19,46%
Pdyn	-16,17%	18,75%	-11,59%	11,91%

TABLE I
Global and local variability ranges for the platform target (Section III-B) in 32nm CMOS. LPD: Longest Path Delay; Plkg: Leakage Power; Pdyn: Dynamic Power.

B. Platform model

In our experiments we used a cycle accurate model of the xSTream multicore platform provided by ST Microelectronics. This platform is meant to be addressing the needs of data-flow dominated, highly computational intensive tasks, typical of many embedded products. It is composed by a General-purpose Processing Element (GPE) acting as host processor and a number of programmable accelerators acting as streaming engine (or fabric). As GPE is used an ST231 processor, which is an embedded media processor derived from the Lx technology platform [2], and as streaming engine an array of xPE processors. xPEs execute instruction fetches from local memories instead of caches, a great simplification at the pipeline forefront. Local memory is also used for wide data access. The system has a global memory containing the program running on the GPE and its data. GPE, xPE array, and global memory are connected through a crossbar.

IV. Variability-aware run-time workload allocation

In this section we describe in details the on-line task allocation policy. The policy is based on the following assumptions. We assume to work on an application composed by sets of independent tasks (i.e. on a frame-by-frame basis) which must be executed by a deadline (i.e. the frame time). We assume to know, for each task set, the cycles needed by the execution of each task, as well as the information about speed and power (i.e. maximum clock frequency, dynamic power, leakage power) of each core[1].

We first outline how the various algorithms are combined, and then we outline each step in details. To find the best

[1]We set each core at its maximum supported clock frequency as explained above.

allocation in a bounded time, the policy exploits two types of solutions starting from a Linear Programming (LP) formulation of the problem (detailed below) that determines the amount of cycles to be allocated to each core such that the deadline is met with minimum energy consumption. Then, a BP step is applied to fit the tasks to be mapped in the bins determined by LP (i.e. LP+BP). Since each task is characterized by its cycle budget, in general the task does not fit exactly the bins, leading to a sub-optimal solution. Increasing the number of tasks (i.e. their granularity becomes finer for a given frame processing), we can approximate the task size by assuming that all tasks have the same cycle budget. In this way the only LP step is enough to reach a high quality. These two solutions are not optimal. However, we exploit them to try to explore all the possible allocations in a bounded amount of time. To achieve this target, we formulated the problem of finding the optimal allocation using a time-bounded Branch & Bound (B&B) algorithm.

A. LP+BP problem formulation

Here we summarize the LP formulation proposed in [11] and present a novel closed-form expression to reduce the time needed find the solution. We do not describe the BP step presented in [11] for the sake of conciseness, because this step is a quite straightforward extension of the best-fit algorithm, and it is not time-consuming.

Each core *i* is characterized by its power in active state (i.e. dynamic power plus active static power $P_{dynAi} + P_{lkgAi}$) and in idle state (i.e. idle static power P_{lkgIi}). For each core i running at frequency f_{cki} we express its active time as T_{Ai} and idle time T_{Ii} as follows:

$$T_{Ai} = \frac{C_{Ai}}{f_{cki}} \quad T_{Ii} = \frac{C_{Ii}}{f_{cki}} \tag{1}$$

where C_{Ai} is the number of cycles spent in *activity* state while C_{Ii} is the number of cycles spent in *idle* state. The LP step starts from the total number of cycles of all the tasks (we called it as K). The goal of the LP is to assign a cycle budget to each core disregarding task granularity. We express the energy consumption as follows:

$$E_{TOT} = \sum_{i=1}^{N} \left[\frac{(P_{dynAi} + P_{lkgAi})\, C_{Ai}}{f_{cki}} + \frac{P_{lkgIi} C_{Ii}}{f_{cki}} \right] \tag{2}$$

Referring to (2) the following vector of $2N$ coefficients R is considered:

$$R = (\frac{P_{dynA1} + P_{lkgA1}}{f_{ck1}}, \frac{P_{lkgI1}}{f_{ck1}}, \ldots, \frac{P_{dynAN} + P_{lkgAN}}{f_{ckN}}, \frac{P_{lkgIN}}{f_{ckN}}) \tag{3}$$

and the vector x of $2N$ real variables that will be then rounded up to the closest integer:

$$x = (C_{A1}, C_{I1}, \ldots, C_{AN}, C_{IN}) \tag{4}$$

The ***LP*** minimization problem can be expressed as:

$$\begin{cases} \min_x R \cdot x^T \\ \frac{C_{Ai} + C_{Ii}}{f_{cki}} = \frac{C_{Aj} + C_{Ij}}{f_{ckj}} \quad \forall i,j : 1 \ldots N, i \neq j \\ \sum_{i=1}^{N} C_{Ai} = K \\ \frac{C_{A1} + C_{I1}}{f_{ck1}} \leq T \end{cases} \tag{5}$$

The first constraint concerns the sum of idle and active times that must be equal for all the cores. The second one concerns the total number of cycles while the third one the maximum execution time T.

A closed-form solution of the LP. The above outlined LP formulation features some properties that simplify its solution. The main observation is that these properties reduce the set of possible optimal LP solutions: they are characterized by a number of cores fully active, a number of cores that are fully idle and at most one core characterized by an incomplete utilization (i.e. only one core is used for a fraction of the frame time). In what follows we provide the mathematical formulation of the closed form solution of the LP.

We call:

$$D_{Ai} = \frac{P_{dynAi} + P_{lkgAi}}{f_{cki}} \quad D_{Ii} = \frac{P_{lkgIi}}{f_{cki}} \tag{6}$$

We can rewrite (2) like this:

$$E_{TOT} = \sum_{i=1}^{N} D_{Ai} C_{Ai} + \sum_{i=1}^{N} D_{Ii} C_{Ii} \tag{7}$$

We can introduce an additional variable t expressing the execution time, replacing the first constraint in (5) with:

$$\frac{C_{Ai} + C_{Ii}}{f_{cki}} = t \quad \forall i : 1 \ldots N \tag{8}$$

and the third one with $t \leq T$. Since

$$C_{Ii} = f_{cki} t - C_{Ai} \tag{9}$$

we can rewrite (7) like this:

$$E_{TOT} = \sum_{i=1}^{N} (D_{Ai} - D_{Ii})\, C_{Ai} + t \sum_{i=1}^{N} D_{Ii} f_{cki} \tag{10}$$

We now define:

$$\begin{array}{l} x_i = C_{Ai}/f_{cki} \\ p_i = (D_{Ai} - D_{Ii})\, f_{cki} = P_{dynAi} + P_{lkgAi} - P_{lkgIi} \\ q = \sum_{i=1}^{N} D_{Ii} f_{cki} = \sum_{i=1}^{N} P_{lkgIi} \end{array} \tag{11}$$

and rewrite the LP formulation as follows:

$$\begin{cases} \min_x \sum_{i=1}^{N} p_i x_i + qt \\ \sum_{i=1}^{N} f_{cki} x_i = K \\ 0 \leq x_i \leq t \leq T \quad \forall i : 1 \ldots N \end{cases} \tag{12}$$

Note that x_i expresses the activity time of core i.

We assume that LP (12) has a feasible solution, which is easily seen to hold if and only if $\sum_{i=1}^{N} f_{cki} T \geq K$. Moreover, we assume that the cores are ordered by increasing values of $\frac{p_i}{f_{cki}} = D_{Ai} - D_{Ii} = \frac{P_{dynAi} + P_{lkgAi} - P_{lkgIi}}{f_{cki}}$, i.e. the values of the cores: $\frac{p_1}{f_{ck1}} \leq \frac{p_2}{f_{ck2}} \leq \cdots \leq \frac{p_N}{f_{ckN}}$ are sorted and represent the penalty in energy for using a cycle of the core. There is an optimal solution of (12) in which either (a) there exists a core s such that either cores $1 \ldots s$ are always active during the execution time and cores $s+1 \ldots N$ are always idle, or (b) the execution time is equal to T, and there exists a core r such that cores $1 \ldots r-1$ are always active during the execution time, core r is partly active and partly idle, and cores $r+1 \ldots N$ are always idle. In fact, the proposition gives a closed-form expression of the optimal solution depending on the specific values of K and T.

B. LP formulation for tasks of equal size

In this section we discuss how to solve the simplified version of the problem in which all tasks have the same number of cycles. The solution computed by this simplified-LP formulation can be better than the LP+BP for a large number of tasks.

In this simplified formulation, we have to decide only the number of tasks to assign to each core since, considering each task given by H cycles, we have that C_{Ai}, the activity cycles of the core i, is given by $C_{Ai} = H \times M_i$, where M_i is the number of tasks assigned to core i.

Using this formulation, the constraint $\sum_{i=1}^{N} C_{Ai} = K$ becomes $\sum_{i=1}^{N} M_i = M$, where M is the number of the tasks. Solving (2) we have for each core a number of tasks which is not an integer value in general. In order to obtain the optimal integer solution firstly we round down the LP solution (removing the fractional part) and assign to each core the resulting number of tasks. Then, we assign the remaining tasks adding them one-by-one to the cores starting from the first core according to the ranking of the closed-form expression. It can be shown that the solution obtained in this way is optimal.

C. Branch & Bound problem formulation

Our analysis operates on a C-ary tree, where C is the number of cores, which represents all the possible allocations of tasks to cores. Each node $N_{j,i}$ of the tree represents the mapping of a task j on a core i. The maximum depth of the tree represents the number of tasks in the program. Operating on a given level j of the tree corresponds to deciding of the mapping of the task j. Given an arbitrary ordering of tasks [2], to explore all the possible mappings we build C trees, each having the first task mapped on a different core. Each node at a generic level $j-1$ has C successors to explore all the possible allocations for the task j.

A solution for our problem consists in finding a *path* from the *root* node to a *leaf* node. Each solution represents a mapping for which we can compute the energy consumption and the execution time. The *best* solution will be the one which requires minimum energy while meeting the deadline. Due to the constrained computational and memory resources we adopt a Branch & Bound search algorithm.

When a node is considered for allocation, the sum of the energy and the execution time for the entire path leading to that node is computed. If the deadline constraint is not met or the energy is higher than that of the *best* solution so far, the search space beneath the node is pruned. The search starts again from the predecessor. In case a valid path until a leaf node is found, the energy spent with the current solution is compared against that of the *best* solution. The best between the solutions provided by LP+BP and simplified-LP is initialized as first best solution for comparison.

V. Parallel MPEG2 decoder benchmark

A description of the MPEG2 decoder can be found in [4, 8]. The task graph is composed of three parts: a control part which scans the current frame, a slice decoding, and an Inverse Discrete Cosine Transform (IDCT). The scan of the current frame is performed by the GPE, the slice decoding and the IDCT can be parallelized and executed on a generic number of xPEs. The slice decoding and the IDCT have been divided in independent tasks whose number can be equal or greater than the xPE number.

To take into account stall cycles[3], we rearranged the formulation shown in Section IV-A. We used the formulation presented in Section IV-A by adding to the parameters regarding activity cycles the contribution due to stall cycles. Referring to (2), and considering r as the ratio between stall cycles and activity cycles, which is assumed constant for each core, we adjusted the first term of (13) as follows:

$$\frac{(P_{dynAi} + P_{lkgAi})\, C_{Ai} + (P_{dynSi} + P_{lkgSi})\, C_{Si}}{f_{cki}} = \\ = \frac{(P_{dynAi} + P_{lkgAi} + (P_{dynSi} + P_{lkgSi})\, r)\, C_{Ai}}{f_{cki}} \tag{13}$$

In this way, we could adapt the LP formulation by simply adjusting the coefficients of the $C_{Ai}s$ variables. Regarding the time we adjusted the deadline in the second constraint of (12) as follows in (14).

$$x_i = \frac{C_{Ai}}{f_{cki}} \leq \frac{t}{1+r} \leq \frac{T}{1+r} \tag{14}$$

In this way, the algorithm can produce the C_{Ai} cycles budget for each core i. Likewise, we give to the algorithm which solves the BP problem the same adjusted time constraint shown in (14) as input.

VI. Experimental results

We report results for all solutions computed by LP+BP, simplified-LP (also called just LP) and B&B in terms of energy consumption and deadline (i.e. frame) miss rate. We compared these results with: Rank Frequency (RF), Rank Power (RP) [13] and a Round-Robin inspection (RR) of cores (i.e. no adopted policy).

A. Set-up

We refer to an 8-core xSTream virtual platform whose cores are affected by variability in terms of maximum supported clock frequency, dynamic power, and leakage power. Since we want to analyze the policies improvement both in terms of global variations and local variations, we extracted 5 different platforms according as described in Section III-A[4].

To compare the results across platforms, the metrics adopted are the average energy consumption among the platforms for each task-allocation technique and the functional yield defined as how many platforms are able to decode the videoclip with zero deadlines misses. In practice, the functional yield is 100% when all the deadlines are met for all the platforms. Since each extracted platform has its own energy spread due to the variability (i.e. spread in terms of power divided by frequency among the cores of the platform) for the sake of comparison we normalized the energy consumption of each frame decoding with respect to a range of energies given defined by the minimum and maximum possible values, regardless the performance: $E = (E - E_{min})/(E_{max} - E_{min})$. The minimum

[2] Tasks are independent and then the order is not important.

[3] Stall cycles are estimated during the profile step as shown in Section VI-A.

[4] We applied VAM onto the xPE netlist and we obtained TableI.

value E_{min} is intended as the energy consumed using only the core with the lowest energy consumption per cycle (i.e. power divided by frequency). The maximum value E_{min} is intended as the energy consumed by the core with highest energy per cycle. The results we report are based on the execution of a videoclip of 25 frames with a frame rate of 25 frame-per-second (i.e. 1 second long) and a resolution of 720×576. We repeated the tests using task sets of variable size (N). In particular, for each frame and both for the slice decoding and for the IDCT we had N independent tasks to allocate onto the cores. For each frame the three sequential steps (control part, slice decoding, IDCT) need to be performed within $40msec$ according to the required frame rate[5]. A profiling step is performed to collect the task information needed by the policy. In particular we store the maximum value of the ratio between stall and activity cycles both for the slice decoding and the IDCT for both slice decoding and the IDCT kernels.

B. Results

Quality of Service - QoS

Regarding QoS, we analyzed the deadline miss rate averaged across the platforms caused by the various policies and we reported the results in Table II. In case of 6 tasks (less than the number of cores), no policy can meet all deadlines for all platforms. It must be observed that RF, which tries to maximize the performance, performs better than the LP+BP solution, but it is worse than the solution that B&B is able to find. Note that anyway the energy provided by RF is higher than both LP+BP and B&B as we will show later.

On the other hand RR, RP, LP show higher deadline miss rates. For what concern LP (i.e. simplified-LP), this was expected because of the small number of tasks, which makes the equal-size approximation very weak. However, LP improves the deadline miss rate of one order of magnitude for 12 and 24 tasks and reaches 0 deadline miss rate for 36 tasks. In general, all the policies perform better when increasing the number of tasks, but RR is unstable as for 12 tasks its deadline miss rate is larger than 0 while it is not for 8 and 24 tasks. RP and RF respect all time constraints from 8 tasks as B&B and LP+BP but, as shown later, the latter provide lower energy consumption.

Regarding the functional yield (i.e. how many platforms can decode the videoclip with 0 deadline miss rate), we report results in Table III. LP+BP, B&B, and RF have always the same yield, and from 8 tasks we can use all the platforms (100% yield). The LP yield increases very quickly from 12 tasks to 24 tasks, and finally reaches the 100% of the yield for 36 tasks.

Energy Efficiency

While QoS and yield for the proposed policies against RF are comparable, this is not the case for energy consumption. Looking at Table IV comparing the energy related to the 0 deadline miss rate values, LP, LP+BP, and B&B solutions always consume less energy than RF and RP. B&B is the best, but LP and LP+BP provide energy consumption closer to B&B when the number of tasks increases.

We plotted in Figure 1 the percentage difference between the energies of LP and LP+BP and the energy of B&B (i.e. $\frac{E - E_{B\&B}}{E_{B\&B}}$, where E is the energy for LP+BP or LP). The negative values indicate that LP or LP+BP consume less than B&B but in these cases the deadline miss rate is higher than B&B. We highlight this performance penalty in the plot using a red segment for LP and a green dashed segment for LP+BP.

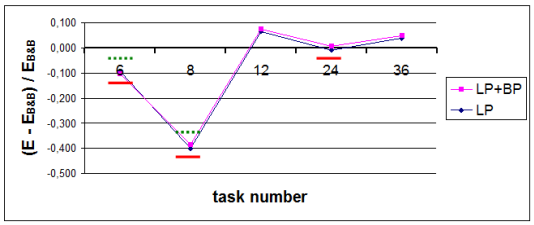

Fig. 1. Energy difference against the B&B energy. When the difference is negative the policy has a higher deadline miss rate than B&B (red segment indicates this for LP, and the green dashed segment is related to this event for LP+BP).

To summarize our findings, we can consider the energy spent for 0 deadline miss rate, which is a desirable case. Only LP leads to lower energy consumption than B&B but its deadline miss rate definitely worse, being not always 0. Moreover, the proposed policy shows better scalability as the energy consumption decreases while increasing the number of task (i.e. the parallelism). RF and RP, instead, have an opposite (less scalable) behavior.

	RR	LP	LP+BP	B&B	RF	RP
6 tasks	0,296	0,328	0,112	0,096	0,104	0,304
8 tasks	0,000	0,384	0,000	0,000	0,000	0,000
12 tasks	0,040	0,056	0,000	0,000	0,000	0,000
24 tasks	0,000	0,008	0,000	0,000	0,000	0,000
36 tasks	0,000	0,000	0,000	0,000	0,000	0,000

TABLE II
DEADLINE MISS RATE PER POLICY; VALUES CAN BE WITHIN 0 AND 1. THE VALUES ARE THE AVERAGE AMONG THE 5 DIFFERENT EXTRACTED PLATFORMS.

	RR	LP	LP+BP	B&B	RF	RP
6 tasks	0%	0%	20%	20%	20%	0%
8 tasks	100%	0%	100%	100%	100%	100%
12 tasks	60%	40%	100%	100%	100%	100%
24 tasks	100%	80%	100%	100%	100%	100%
36 tasks	100%	100%	100%	100%	100%	100%

TABLE III
FUNCTIONAL YIELD IN PERCENTAGE: HOW MANY PLATFORMS CAN DECODE THE VIDEOCLIP WITH 0 DEADLINE MISS RATE.

	RR	LP	LP+BP	B&B	RF	RP
6 tasks	0,874	0,617	0,673	0,679	0,669	0,682
8 tasks	0,810	0,425	0,721	0,709	0,783	0,784
12 tasks	0,785	0,650	0,615	0,611	0,770	0,789
24 tasks	0,793	0,565	0,579	0,570	0,792	0,800
36 tasks	0,792	0,580	0,565	0,559	0,788	0,789

TABLE IV
ENERGY CONSUMPTION PER POLICY. VALUES ARE NORMALIZED AND CAN ASSUME VALUES WITHIN 0 AND 1. THE VALUES ARE THE AVERAGE AMONG THE 5 DIFFERENT EXTRACTED PLATFORMS.

Policy execution time overhead

The algorithms for computing the various solutions are executed on the ST231 host core clocked at 900MHz to find the allocation for the slice decoding and IDCT kernel tasks. The decoding performed by the xPEs is overlapped with the task

[5]We extract the deadline T for the slice decoding and the IDCT for each frame from $40msec$ in according to their own needed cycles and taking into account the time for the control part.

allocation performed by the ST231 for the next frame. To be feasible on-line, the policy must be executed in approximately one third of the frame decoding time (remember that frame decoding is composed by three parts as explained in the section on the MPEG2), that is 12,000μsec at the considered frame rate.

By looking at Table V, where execution time results are reported, it can be observed that B&B is in general more time consuming. In many cases, B&B does not exploit all the available time because in some cases it can find the optimal solution in a short time compared to the frame execution time. This arises in case of few tasks. For instance, the average needed time for 6 tasks is shorter than 10,000μsec. B&B can find the optimal solution improving those of LP+BP and LP (see Table II). By increasing the number of tasks B&B hardly finds the optimal solution. However B&B still improve the LP+BP or the LP solution in some cases. Focusing on RF and RP, they show poor scalability also in terms of execution time, since it increases of 100μsec in a linear way with the number of tasks. Overall, the proposed policy provides the best results in terms of energy and QoS and it can be implemented on-line during frame decoding.

	LP	LP+BP	B&B	RF	RP
6 tasks	205	256	6779	265	287
8 tasks	227	206	9203	348	372
12 tasks	221	262	10389	415	441
24 tasks	215	280	10871	630	655
36 tasks	216	354	11452	850	876

TABLE V
TIME TO EXECUTE THE ALGORITHMS AT RUN-TIME. THE ALGORITHMS ARE EXECUTED ONTO THE HOST PROCESSOR OF XSTREAM WHICH IS AN ST231 AT 900MHZ. THE VALUES ARE IN μsec AND THE AVERAGE AMONG THE 5 DIFFERENT EXTRACTED PLATFORMS.

Cross-Platform variability compensation

To highlight the impact of the proposed policy on the variability compensation, we report the energy and performance for each variability affected platform in Figure 2. In this scatter graph, on the Y axis we find the normalized energy consumption presented in Section VI-A, while on the X axis the average execution time per frame is reported. This

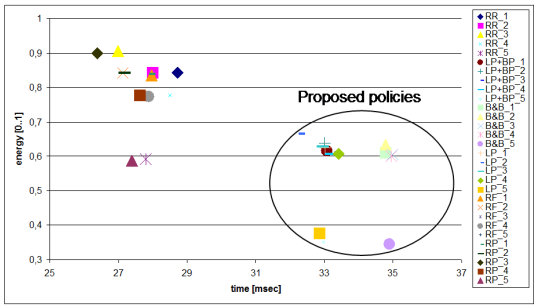

Fig. 2. Along the Y axis the normalized energy, the values are between 0 e 1. Along the X axis the average execution time. 36-task case. The numbers on the labels identify the platforms.

plots highlights that the proposed policy which uses LP+BP, LP and B&B solutions grouped on the right side of the plot, is more predictable in terms of decoding time across the various platforms, actually compensating the platform-by-platform variations. Indeed, their performance is close to 35$msec$ which is shorter than the deadline of 40$msec$. Note also that the decoding time is in general larger than RF, because the proposed policy exploits all the available time to minimize energy consumption. The lowest energy consumption is provided by the B&B solution, located on the right-bottom part of the plot.

VII. CONCLUSION

In this work we presented an on-line policy to compute the allocation of tasks in variability-affected multicore platforms. The policy has been implemented on top of a cycle accurate virtual prototype of an industrial MPSoC and applied to a realistic MPEG2 benchmark ported to the platform. We reported results about policy overhead to demonstrate its applicability at runtime and we demonstrated its effectiveness in terms of QoS and energy efficiency with respect to state-of-the-art approaches.

REFERENCES

[1] Y. Cao and C. McAndrew, "Mosfet modeling for 45nm and beyond," in *IEEE, Proceedings of the Conference on International Conference on Computer-Aided Design*, 2007, pp. 638–643.

[2] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homewood, "Lx: a technology platform for customizable vliw embedded processing." in *Proceedings of the Conference on International Symposium on Computer Architecture*, 2000, pp. 203–213.

[3] E. Flamand, "Strategic directions toward multicore application specific computing," in *IEEE, Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 1266–1266.

[4] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An introduction to MPEG-2*. London, UK, UK: Chapman & Hall, Ltd., 1996.

[5] S. Herbert and D. Marculescu, "Characterizing chip-multiprocessor variability-tolerance," in *ACM, Proceedings of the Conference on Design Automation Conference*, 2008, pp. 313–318.

[6] S. Hong, S. Narayanan, and M. Kandemir, "Process variation aware thread mapping for chip multiprocessors," in *IEEE, Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 821–826.

[7] "Vam - variability aware modeling," IMEC, http://www.imec.be/ScientificReport/SR2007/html/1384291.html.

[8] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. Legall, Eds., *MPEG Video Compression Standard*. London, UK, UK: Chapman & Hall, Ltd., 1996.

[9] A. Papanicolaou, M. Miranda, P. Marchal, B. Dierickx, and F. Catthoor, "At tape-out: Can system yield in terms of timing/energy specifications be predicted?" *IEEE, Proceedings of the Conference on Custom Integrated Circuits Conference*, pp. 773–778, 2007.

[10] F. Paterna, A. Acquaviva, A. Caprara, F. Papariello, G. Desoli, and L. Benini, "Variability-tolerant run-time workload allocation for mpsoc energy minimization under real-time constraints," in *ACM, Proceedings of the Conference on Computing Frontiers*, 2010, pp. 109–110.

[11] F. Paterna, A. Acquaviva, F. Papariello, G. Desoli, and L. Benini, "Variability-tolerant workload allocation for mpsoc energy minimization under real-time constraints." in *IEEE, Proceedings of the Workshop on Embedded Systems for Real-Time Multimedia*, 2009, pp. 134–142.

[12] L. Singhal and E. Bozorgzadeh, "Process variation aware system-level task allocation using stochastic ordering of delay distributions," in *IEEE, Proceedings of the Conference on Computer-Aided Design*, 2008, pp. 570 –574.

[13] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 363–374, June 2008.

[14] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," *IEEE/ACM, International Symposium on Microarchitecture*, pp. 129–140, 2008.

[15] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task allocation and scheduling for mpsoc," in *IEEE, Proceedings of the Conference on Computer-Aided Design*, 2007, pp. 598–603.