Decision-theoretic Exploration of Multi-Processor Platforms

Giovanni Beltrame, Dario Bruschi, Donatella Sciuto, Cristina Silvano
Politecnico di Milano
Piazza Leonardo da Vinci 32
20133 – Milano – Italy
{beltrame,sciuto,silvano}@elet.polimi.it, dario.bruschi@gmail.com

ABSTRACT

In this paper, we present an efficient technique to perform design space exploration of a multi-processor platform that minimizes the number of simulations needed to identify the power-performance approximate Pareto curve. Instead of using semi-random search algorithms (like simulated annealing, tabu search, genetic algorithms, etc.), we use domain knowledge derived from the platform architecture to set-up exploration as a decision problem. Each action in the decision-theoretic framework corresponds to a change in the platform parameters. Simulation is performed only when information about the probability of action outcomes becomes insufficient for a decision. The algorithm has been tested with two multi-media industrial applications, namely an MPEG4 encoder and an Ogg-Vorbis decoder. Results show that the exploration of the number of processors and two-level cache size and policy, can be performed with less than 15 simulations with 95% accuracy, increasing the exploration speed by one order of magnitude when compared to traditional operation research techniques.

1. INTRODUCTION

Platform-Based Design [5, 11] has recently emerged as an important design style since it eliminates costly design iterations. In this context, parameterized embedded System-on-Chip (SoC) architectures must be optimally tuned to find the best trade-offs in terms of the selected figures of merit (e.g. energy and delay) for the given class of applications. This tuning process is also called the *Design Space Exploration (DSE)*. The overall goal of the DSE phase consists of optimally configure the parameterized SoC platform in terms of system-level requirements depending on the given application.

In general, the optimization problem involves the minimization (maximization) of *multiple objectives* making the definition of optimality not unique. The solution of *multi-objective* or *constrained* optimization problems consists of finding the points of the Pareto curve [7]. For example, the optimization problem is to minimize the power consumption under a delay constraint or viceversa. The solution of this problem is straightforward if the Pareto curve is available. However, a Pareto curve for a specific platform is available only when all the points in the design space have been explored and characterized in terms of objective functions. This full search approach is often unfeasible due to the cardinality of the design space and to the long simulation time needed for computing the evaluation functions.

In the past, SoC platforms have been explored with classical algorithms (tabu search, simulated annealing, etc.) that do not assume any information regarding the effects of transformations in the platform configuration. Therefore, they heavily rely on simulation as a means for evaluating the metrics of a newly found configuration. If system-level simulation can be performed in reasonable time, these algorithms provide good results, but this is generally not true for Multi-Processor System-on-Chip (MPSoC), for which simulations can be rather time consuming.

This work tries to fill this gap by exploiting domain knowledge provided by the definition of a design platform. The idea is to move the design space exploration complexity from simulation to statistical analysis of parameter transformations. Exploration is modeled as a Markov Decision Process (MDP), and the solution to such MDP corresponds to the sequence of transformations to be applied to the platform to maximize a certain value function. Finding a solution to the problem requires to simulate the system only in particular cases of uncertainty, massively reducing the simulation time needed to perform the exploration of a system, while maintaining near-optimality of the results.

Overall, this work provides four main contributions: the development of an efficient exploration methodology that reduces the simulation time required for exploration; the formalization of platform domain knowledge for MPSoC; the modeling of design space exploration as a decision process; and the development of an efficient solution algorithm for the exploration problem.

This work is structured as follows: Section 2 summarizes classical exploration algorithms for multi-variate analysis; Section 3 introduces the decision model and its solution algorithm; Section 4 describes the application of the methodology to two industrial benchmarks; finally, Section 5 draws some concluding remarks.

2. RELATED WORK

Two main approaches have been proposed in literature for the design space exploration of system architectures: tech-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CODES+ISSS'06, October 22–25, 2006, Seoul, Korea.
Copyright 2006 ACM 1-59593-370-0/06/0010 ...$5.00.

niques that try to reduce the design space size, and techniques that provide exploration heuristics.

The former class of techniques tries to limit the exponential increase of design space size by determining relations between architectural configurations, pruning the design space to avoid simulations of those configurations that are certainly non-optimal. In [3] the design space is divided in partitions, limiting the number of possible configurations. Inside each partition, exploration is performed via exhaustive search. The authors also generate a dependency graph between parameters that identifies how modifications on a parameter can require the variation of another. Once the graph is generated, cyclic dependencies identify the design space partitions, and local optimal configurations are identified. Global optimal configurations are computed starting from local ones. This methodology, however, relies on the designer for the generation of the dependency graph that has either to use his experience or evaluate manually the results of simulations.

A different technique is proposed in [2] that reduces the number of configurations from the product of parameters to their sum, guiding the exploration using a sensitivity list. This strategy orders applicable parameter transformations according to their effect on the system metrics. This list is generated from the results of exhaustive simulation on a set of benchmarks, and it can be used on the same class of applications as the benchmarks. Starting from the first element in the list and keeping the other parameters constant, the system is explored exhaustively. Proceeding to the next parameters, the optimal configuration for the considered subspace is found. Although complexity is highly reduced, the exploration results remain sub-optimal, and the results are bound to the initial tuning phase.

Since low-level simulation is often the bottleneck of DSE, [6] introduces hierarchical exploration, lowering the abstraction on smaller partitions of the design space. The initial reduction of the design space is obtained by using component vendor data-sheets or probabilistic analysis, discarding all the configurations that do not meet the design constraints. After a first exploration phase using high-level simulations, the design space is reduced again and the process is repeated until the number of configurations is $N < 10^2$. After this, exploration is continued using more classical methods.

The second class of techniques apply heuristics to the exploration problem. In this context, genetic algorithms have been used for architectural exploration [8, 1], but, although they reduce by 80% the time needed for exploration when compared to exhaustive search, their effectiveness is bound to large sample populations. This means that even if the performance gain increases with the size of the population, simulation times still remain excessive for complex applications. Monte Carlo methods can also be applied to architectural exploration [4], approximating the Pareto function of system metrics. ParetoTabu Search (PTS) and Pareto Simulated Annealing (PSA) belong to this category of function approximation heuristics, and in [7] they are applied to architectural exploration. Both methods start from a set of points that are considered a Pareto curve, and they try to find new points that cover the current set. PTS guides random exploration using tabu lists that prevent directions that will bring to points close to the current set, PSA instead extends gradient search with a technique to get out of local minima, allowing temporarily to move in covered points. Both techniques are much more efficient than exhaustive search, but to reach an accuracy close to 3% they still require $N > 10^3$ simulations.

When considering platform-based design [11], these algorithms do not consider the design constraints imposed by the platform. These constraints provide a useful amount of knowledge that can be used to define better simulation strategies, reducing sensibly the number of required simulations. This work provides the theoretical infrastructure to exploit this information, and introduces an exploration algorithm based on decision theory as outlined in the following.

3. PROPOSED METHODOLOGY

The exploration of the design space of an application platform can be described as a path that leads an initial configuration to a destination configuration with better characteristics. To cover this path, platform parameters are subject to *transformations*. These transformations change the behavior of the platform and the effects are shown in terms of platform *metrics*, e.g. execution time and power consumption. In the following, the results of transformations are considered deterministic (i.e. a platform with a given set of parameters will have a 1-1 correspondence with its metrics) and usually not known a-priori, and can only be evaluated through simulation.

We propose to substitute the simulation phase with a probabilistic estimation of the results of a transformation, modeling the problem as a *Markov Decision Process* [10].

DEFINITION 1. *A Markov Decision Process (MDP) is a t-uple $\langle S, A, T, R \rangle$ where:*

- *S is the set of possible system states*
- *A is a set of possible actions, i.e. parameter transformations*
- *$T : S \times A \longrightarrow \Pi(S)$, the state transition function, is a probability density for a destination state for every state-action pair, i.e. $T(s, a, s')$ is the probability to get in state s' applying a in state s*
- *$R : S \times A \longrightarrow \mathbb{R}$ is the expected reward for each state-action pair, i.e. $R(s, a)$ is the expected reward executing a in state s*

In addition, the Markov property [10] holds: the state transition function depends only on current state and action. Let us assume $P(S_{n+1} = x)$ the probability that next state will be x, we have:

$$P(S_{n+1} = x | S_0, S_1, S_2 ... S_n) = P(S_{n+1} = x | S_n)$$

Solving an MDP means finding the strategy (i.e. the mapping state-action, the best action for each state in the system) that maximizes the expected reward V. This means that the agent starting in state s will execute a series of *steps*, trying to maximize a given metric. Several algorithms exist to solve optimally MDPs, among which *value iteration* [10] is one of the most popular.

3.1 Platform exploration as an MDP

Design space exploration of a platform can be seen as the movement of two points in two spaces: the parameter space and the metrics space. Actions determine a movement in

the parameter space, modifying the current system configuration. This movement is deterministic and discrete, according to platform constraints. This produces a corresponding movement in the metrics space that determines the results of the action. The choice of the action depends uniquely on the corresponding movement vector in the metrics space.

It is then possible to model the platform exploration problem as an MDP:

- S is the set of all possible configuration-metrics tuples.
- A is the set of applicable actions, i.e. transformations to the platform parameters.
- $T : S \times A \longrightarrow P_i(S)$ is the probability density over the metrics space applying transformation a in state (configuration-metric tuple) s.
- $R : S \times A \longrightarrow \mathbb{R}$ is the reward associated with each state-action pair, i.e. the difference between the metric in the starting state s and in the destination state of action a. In the following we use TE^{α} in the $\{T, E\}$ metrics space, with T the execution time of an application, and E its energy consumption.

The Markov property holds: the effect of a transformation depends only on the configuration to which it is applied and not on past actions that brought to the given configuration.

3.1.1 Movement vectors

In the case of complete uncertainty of the effects of actions, simulation is the only way to determine the corresponding position in the metrics space of the current configuration. However, platform-based design limits the number of parameters and provides the knowledge base to estimate the effects of actions, without requiring simulation. In particular, being able to determine a subspace of movement in the metrics space, corresponding to a transformation in the parameters space, allows us to use simulation only for uncertainty conditions. We call *movement vectors* the model used to determine this subspace.

Let us consider the set $P = \{p_1, p_2, ..., p_n\}$ of platform parameters where p_k can be for example the number of processors, cache size, number of threads, etc. To each p_k it is possible to apply a transformation τ that modifies its value as in [2], i.e. it can increase or decrease its value, modifying the behavior of the target architecture.

Definition 2. *Given a platform configuration $P = \{p_1, \cdots p_k, \cdots p_n\}$, a transformation $\tau(p_k, \Delta)$ produces a configuration $P' = \{p_1, \cdots p'_k, \cdots p_n\}$ where $p'_k = p_k + \Delta$ with $\Delta \in \mathbb{Q}$.*

To evaluate the effect of a transformation on the architecture, we either compute or estimate the variations on a set of reference metrics. For this purpose, we define a set of metrics $M\{m_1, .., m_i\}$. In the following, for the sake of simplicity, we limit the search to a bi-dimensional metrics space $M \triangleq \{T, E\}$, where T is the execution time of the application and E represents its energy consumption.

Definition 3. *A movement vector is an interval in the metrics space corresponding to a vector in the parameter space, and it is defined as:*

$$\langle f_1(\tau(p_k, \Delta)), f_2(\tau(p_k, \Delta)), ..., f_i(\tau(p_k, \Delta)) \rangle$$

where $i = |M|$, and $f_1, f_2, ..., f_i$ functions that determine the range of effects of the transformation τ on $\forall m_k \in M$.

Applying movement vectors limits the destination metrics subspace of a transformation of the parameters.

To determine movement vectors for a given parameter, it is sufficient to determine the minimum and maximum variations in the metrics space for a given transformation. Let us consider an increase of the number of processors (np) by one in the $\{E, T\}$ metrics space. In the worst case, the addition of a processor would have no effect on execution time:

$$f_T(np') = f_T(np)$$

In the best case, the new CPU will get a substantial share of the executed instructions, and the load would be evenly balanced:

$$f_T(np') = \frac{I \cdot CPI}{f \times L \times \frac{np}{np'}}$$

where I is the number of instructions executed on average by each processor, L the average load per processor and CPI the average number of clock cycles per instruction, and f the processor frequency. Energy consumption can be determined consequently, as a cost associated with executed instructions. We defined movement vectors (omitted for the sake of brevity) for many platform characteristics, among which cache size, number of interconnection links, number of threads, etc.

3.1.2 States

Since it is not possible to know the effects of an action without simulation, each state is identified by the $\langle P, M \rangle$ tuple, i.e. the configuration in the parameter space P and by an estimation of the metrics M. For each configuration, several states are defined, corresponding to possible metrics values. Since the metrics space is inherently continuous and grows exponentially, it is necessary to (a) make it discrete and (b) reduce the number of states without affecting accuracy.

Given the subspace identified by the movement vector associated with an action, we partition it according to a given accuracy and consider the centroid of the partition as a state of the MDP. This partitioning introduces an error given by the difference between the actual value of the metrics and the centroid of the partition that best approximates it. Increasing the number of partitions we can reduce the error, but it increases the number of states and corresponds to an exponential increase in the solution time of the MDP [10]. In this work we propose a variable partitioning strategy that generates new states only if required to obtain a given accuracy. We propose two ways to reduce the number of states forming the MDP:

- We determine the λ *distance* in the metrics space, that is the minimum distance between two separate points, implicitly determining the number of partitions associated with each movement vector. This allows us to minimize the number of states for a given accuracy, and the precision of the proposed algorithm can be increased by arbitrarily reducing λ.
- We consider a *event horizon* l, that represents the number of projection steps considered in the value-iteration algorithm, determining the number of reachable states. This number is $|A|^{l+1}$ and can be tuned accordingly to trade-off accuracy and exploration speed.

3.1.3 Actions

Actions are also a dominant factor for the complexity of MDPs [10], so we adopted a strategy to reduce the number of applicable actions in each state. Each action is a transformation to the platform parameters, and in the following the terms are considered interchangeable. Each parameter can be modified by dual transformations:

$$\tau^+ = \tau(k, \Delta) \quad \text{and} \quad \tau^- = \tau(k, -\Delta)$$

The execution of τ^+ is made void by the execution of τ^-, but since we are operating on probability subspaces, the subsequent execution of τ^+ and τ^- introduces an estimation error. To remove this error, we keep a list of executed actions and dual actions are forbidden in all subsequent steps. Dual actions are allowed again only when backtracking from non promising solution paths (as outlined in Section 3.2).

3.1.4 Transition functions

With the partitioning of the metrics space, $T(s, a, s')$ represents the probability of reaching a partition with a centroid s' from state s applying action a. This information is not easily determined before simulation, and therefore the initial distribution is uniform on the intervals specified by the movement vector associated with a.

Supposing that we know the actual destination point of a (as opposed to the destination probability distribution), we can *correct* the probability distribution $T(s, a)$.

DEFINITION 4. *We define the* virtual sample number w_i *for each partition* $d_{i,a,s}$ *associated with a state-action tuple* $\langle s, a\rangle$ *as an index of the probability of ending in* $d_{i,a,s}$ *with respect to all the other partitions* $d_{j,a,s}$ *with* $j \neq i$.

Using virtual sample numbers, transition functions for each state-action pair are computed adding $c \geq 1$ virtual samples to each partition $d_{i,a,s}$ when building the decision tree associated with the MDP. This defines a more realistic T as the algorithm goes on with the exploration, while maintaining the Markov property. If the number of partitions associated with a tuple $\langle s, a\rangle$ varies during exploration, the virtual samples are redistributed.

Dual actions enforce a particular relation between their probability distributions: the more that action is ineffective, the more effective will be its dual. This means that $T(s, a)$ implies a complementary $T(s, a^{-1})$ with a^{-1} the dual of a. If a has developed a certain T_a during exploration, the reactivation of a^{-1} automatically derives its $T_{a^{-1}}$ by mirroring T_a. This creates a knowledge base on a^{-1} even if it was never applied to the system configuration.

3.2 Exploration Algorithm

The algorithm consists of four steps: generation of the decision tree, computation of the strategy, application of the transformations, restart and optimality verification.

The first step is the generation of the graph $G = \{N, E\}$ that describes the behavior of the MDP. In general, this is a bidirectional cyclic graph with a finite number of states. A node $n \in N$ in the graph represents a state of the system, and in particular a tuple $\langle P, M\rangle$ where P is a point in the parameters space and M is a point in the metrics space. This means that even if two states correspond to the same configuration P, they may have different metrics estimations, and are therefore considered separately. This property and the fact that dual actions are not enabled, frees G from cyclic structures, simplifying the problem. An edge $e \in E$ is defined as:

$$e = \langle n_i, n_j, a, T(n_i, a, n_j)\rangle$$

and it represents the transition probability (given by the transition function $T(\cdot)$) from node n_i to node n_j when applying action a. At the start of the exploration, the initial configuration is chosen randomly and its metrics are determined via simulation, building the root node n_0. Starting from n_0, all possible actions are applied, identifying all possible configurations that differ from n_0 by one parameter. For each configuration, movement vectors identify the destination subspace in the metric space. This is partitioned according to the maximum accuracy defined for the exploration (see Section 3.1.2) using λ. The centroid associated with each partition, together with the respective configuration, generates a new child node, and the difference between parent and child metric defines the reward associated with each state-action pair. Last, for each node n_i, $T_i(n_i, a, n_k)$ is generated by adding a fixed number c of samples to the parent distribution.

The decision tree is progressively built by iterating on the newly generated nodes breadth-first, until either it is not possible to apply any other action on the leaf nodes or the l-th level is reached, our event horizon.

The second step computes the strategy and consists of a modified version of the value iteration algorithm[10], and it is described by Algorithm 1. This allows the computation

Algorithm 1 The strategy evaluation algorithm

```
initialize V(s) = 0
repeat
  for all s ∈ S do
    for all a ∈ A do
      Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s')V(s')
    end for
    V(s) := max_a Q(s,a)
  end for
until policy converges
```

of the expected average value for the execution of an action, weighing the value of child nodes by the probability of reaching them from parent nodes, starting from leaf nodes, associated with their immediate reward only. To take into account estimation and simulation error and to avoid fluctuations around an optimal value, we define a minimal reward for l steps. Considering estimation in l steps, the l actions will be executed only if the cumulative reward is greater than the minimal value (e.g. TE^α decreases at least 1%). This also guarantees the convergence of the algorithm, due to the diminishing returns of actions applied to the system. This phase ends with the determination of the best sequence of k actions to be performed from the initial state.

The actions identified in the previous stage are applied to the system and their results are evaluated by simulation only when strictly necessary. If the strategy suggests action a, three conditions are possible:

1. a brings in only one state. The action is deterministic within the λ accuracy chosen for the algorithm, and no simulation is necessary.
2. a produces more than one state, and each of them is mapped to the same action by the strategy. This

means that whichever state the system will end into, the same next action will be chosen, and therefore it is not useful to verify it by simulation.

3. a produces more than one state, and they are mapped to two or more different actions. Only in this case it is important to discriminate in which state the system will actually get, and simulations are performed.

Every time an uncertainty condition arises, the algorithm is stopped and simulation is performed, fixing its position in the metrics space.

The application of the strategy goes on until either the event horizon of l steps is reached or an optimal configuration is found (when the algorithm converges). In the former case, the algorithm is restarted considering the last optimal configuration as the initial node. However, all the collected information (probability distributions, forbidden actions, etc.) are kept and used in the subsequent runs of the algorithm.

In the latter case, the algorithm verifies it is not in a local minimum, and does so by restarting the algorithm, but also re-enabling the forbidden actions. This enables the algorithm to backtrack and find more promising paths towards the solution or to remove actions that were wrongly estimated as effective. If no state in excess of the minimum reward is found, the exploration is terminated and the current configuration is returned as the optimal.

3.3 Pareto search

The algorithm works with utility functions that have the utility property [10]. To have the algorithm generate an approximate Pareto curve it must be possible to explore in a multi-variate environment. This is obtained by repeated applications of the algorithm, changing the reward function at every application so that it covers the whole span of the metric space of interest. During the exploration, all points simulated by the algorithm are kept if and only if they are not Pareto-covered by other points. At the end of the scan of the design space, the remaining points constitute the estimation of the Pareto curve. As an example, in the experiments detailed in the following, we explored using the metric TE^{α} (see Section 3.1) with $\alpha = \{0, 1, 2\}$.

4. EXPERIMENTAL RESULTS

The proposed methodology has been verified on two industrial benchmarks: an MPEG4 encoder and an Ogg-Vorbis decoder. Both applications have been run on the StepNP simulation platform [9]. This platform is formed by a configurable number of ARM processors with private instruction and data caches connected via an STBus interconnection channel. The platform also features a shared second-level cache, connected to an external memory through a bus. In particular, exploration was performed on:

- Number of processors: this number is essential to reach the timing constraints of the application without drawing excessive power. The bounds for the system are represented by a minimum of 2 cores and a maximum of 8.

- Instruction cache: the size ranges from 256 byte to 64K byte, with direct mapped policy.

- Data cache: the size ranges from 256 byte to 64K byte, direct mapped and 2- to 8-way caches are considered.

The design space is then formed of 1890 possible configurations. Exploration of the configurations is based on two metrics: the application execution time, and its energy consumption. All the explorations have been run on a P4 2.66GHz with 512MB of RAM. The proposed methodology has been compared with two industry-standard algorithms for exploration: Pareto Tabu Search (PTS), and Pareto Simulated Annealing (PSA) both with the same number of simulations and with a setup to obtain the same accuracy.

4.1 Comparison: same simulation effort

The exploration for the MPEG4 benchmark with a 5% error required 15 simulations, while Ogg-Vorbis required 11. This leads to two approximate Pareto curves of 7 and 6 points for MPEG4 and Ogg-Vorbis, respectively. Comparing this result with PTS and PSA, using approximately the same number of simulations (15) leads to the results shown in Figure 1(a) and Figure 1(c). PTA15 and PTS15 found less approximate Pareto points, and most of the points they found are covered by the MDP approximate Pareto curve.

More formally, our results show that MDP covers entirely the approximate Pareto curve generated by PSA in both applications, and covers 60% and 50% of the curves generated by PTS for MPEG4 and Ogg-Vorbis, respectively.

4.2 Comparison: same accuracy

To increase the robustness fo the results, We also compared MDP with PTS and PSA tuning them for the same accuracy level (allowed error 5%, convergence at 3%), requiring ~ 100 simulations for the classical algorithms. The results depicted in Figure 1(b) and Figure 1(d) show very good superposition of the curves with the one found by MDP, with minor strays due to the given accuracy level. This proves that MDP can obtain results very similar (in the confidence range) to those of PTS and PSA, using significantly less simulations (15 for MPEG4 and 11 for Ogg-Vorbis).

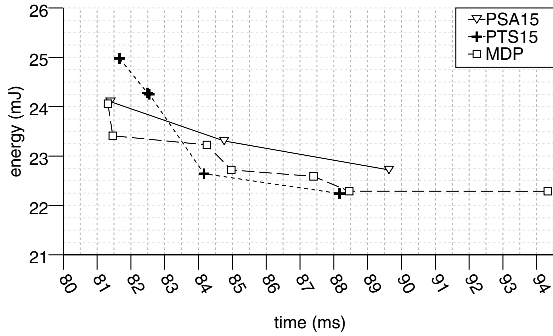
4.3 Algorithm performance

Last, the algorithm exploration effort was estimated for different λ distances and applications, and the results are reported in Table 1 The number of nodes and the execution time grow exponentially as the allowed error decreases [1], as expected. Nevertheless, the tree search times are orders of magnitude smaller than system-level simulation ($\sim$ 30 minutes for MPEG4, $\sim$ 9 minutes for Ogg-Vorbis), confirming the validity of the approach even for very high accuracies. Overall, the algorithm brings an exploration time save of more than 82% as shown by Figure 2, reporting the normalized exploration times for MDP, PTS and PSA

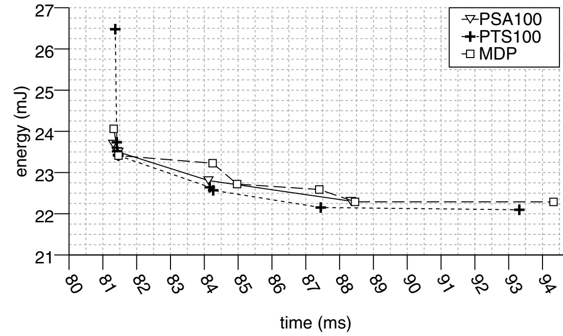
Table 1: Algorithm execution effort

Application	error [%]	nodes	exec. time [s]
MPEG4	1,25	19737	166
	2,5	3162	38
	5	1119	25
Ogg-Vorbis	2,2	43152	316
	4,4	6796	58
	8,8	1651	28

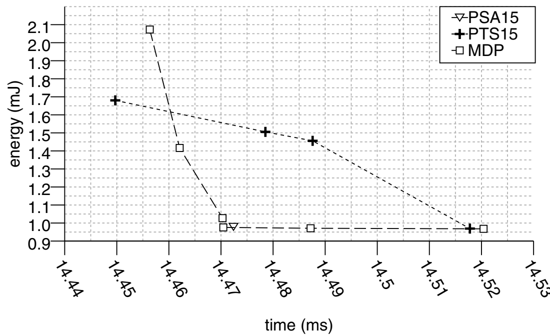
[1]Using different λ, accuracies in percentage may differ depending on the application.



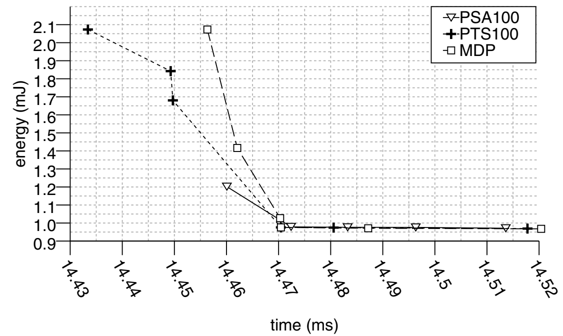
(a) MPEG4: MDP, PSA15 and PTS15



(b) MPEG4: MDP, PSA100 and PTS100



(c) Ogg-Vorbis: MDP, PSA15 and PTS15



(d) Ogg-Vorbis: MDP, PSA100 and PTS100

Figure 1: Approximate Pareto curves for the MPEG4 encoder and the Ogg-Vorbis decoder

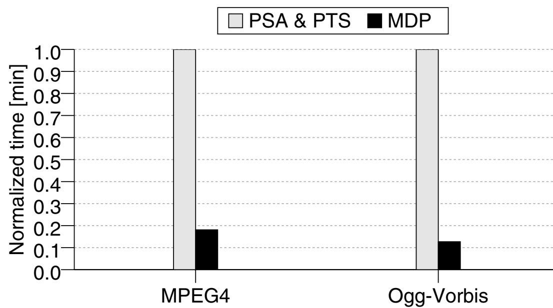


Figure 2: Normalized exploration time for 95% accuracy with PSA, PTS and MDP

5. CONCLUSIONS

This paper presented an applicatio of decision theory, and in particular of Markov Decision Processes, to architectural multi-variate exploration for platform-based design. The proposed algorithms uses domain knowledge to reduce the number of simulations needed for exploration up to an order of magnitude, defining "movement vectors" in the metrics space for parameter transformations. Results show an excellent overall exploration speedup (> 82%) and very good accuracy of the discovered pareto curve. Future work includes the definition of more movement vectors, and the combination of pseudo-random techniques to MDP to use the best characteristics from both worlds.

6. REFERENCES

[1] R. P. Dick and N. K. Jha. MOGAC: A multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedde systems. *IEEE transactions on CAD*, 17(10):920–935, 1998.

[2] W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria. A sensitivity-based design space exploration methodology for embedded systems. *Design Automation for Embedded Systems*, 7(1):7–33, 2002.

[3] T. Givargis, F. Vahid, and J. Henkel. System-level exploration for pareto-optimal configuration in parametrized system-on-a-chip. *IEEE Transactions on VLSI*, 10(4):416–422, August 2002.

[4] T. D. Givargis and F. Vahid. Platune: A tuning framework for system-on-a-chip platforms. *IEEE Transactions On CAD*, 21:1317–1327, 2002.

[5] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions On CAD*, 19:1523–1543, 2000.

[6] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *LCTES/SCOPES '02*, pages 18–27, 2002.

[7] G. Palermo, C. Silvano, and V. Zaccaria. Multi-objective design space exploration of embedded systems. *Journal of Embedded Computing*, 1, 2006.

[8] M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. In *CODES'02*, 2002.

[9] P. G. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A system-level exploration platform for network processors. *IEEE Design and Test of Computers*, 1:2–11, 2002.

[10] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[11] A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, and M. Sgroi. Benefits and challenges for platform-based design. In *DAC '04*, pages 409–414, 2004.