System and Application Scenarios for Disaster Management Processes, the Rainfall-Runoff Model Case Study

Antoni Portero, Štěpán Kuchař, Radim Vavřík, Martin Golasowski, and Vít Vondrák

VŠB – Technical University of Ostrava, IT4Innovations, Ostrava, Czech Republic
{antonio.portero,stepan.kuchar,radim.vavrik,martin.golasowski, vit.vondrak}@vsb.cz

Abstract. In the future, the silicon technology will continue to reduce following the Moore's law. Device variability is going to increase due to a loss in controllability during silicon chip fabrication. Then, the mean time between failures is also going to decrease. The current methodologies based on error detection and thread re-execution (roll back) can not be enough, when the number of errors increases and arrives to a specific threshold. This dynamic scenario can be very negative if we are executing programs in HPC systems where a correct, accurate and time constrained solution is expected. The objective of this paper is to describe and analyse the needs and constraints of different applications studied in disaster management processes. These applications fall mainly in the domains of the High Performance Computing (HPC). Even if this domain can have differences in terms of computation needs, system form factor and power consumption, it nevertheless shares some commonalities.

Keywords: HPC Systems, disaster management, reliability models.

1 Introduction

Application requirements, power, and technological constraints are driving the architectural convergence of future processors towards heterogeneous many-cores. This development is confronted with variability challenges, mainly the susceptibility to time-dependent variations in silicon devices. Increasing guard-bands to battle variations is not scalable, due to the too large worst-case cost impact for technology nodes around 10 nm. The goal of next generation firmware is to enable next-generation embedded and high-performance heterogeneous many-cores to cost-effectively confront variations by providing Dependable-Performance: correct functionality and timing guarantees throughout the expected lifetime of a platform under thermal, power, and energy constraints. An optimal solution should employ a cross-layer approach. A middle-ware implements a control engine that steers software/hardware knobs based on information from strategically dispersed monitors. This engine relies on technology models to identify/exploit

K. Saeed and V. Snášel (Eds.): CISIM 2014, LNCS 8838, pp. 315–326, 2014.

various types of platform slack - performance, power/energy, thermal, lifetime, and structural (hardware) - to restore timing guarantees and ensure the expected lifetime and time-dependent variations.

Dependable-Performance is critical for embedded applications to provide timing correctness; for high-performance applications, it is paramount to ensure load balancing in parallel phases and fast execution of sequential phases. The lifetime requirement has ramifications on the manufacturing process cost and the number of field-returns. The future firmware novelty must rely in seeking synergies in techniques that have been considered virtually exclusively in the embedded or high-performance domains (worst-case guaranteed partly proactive techniques in embedded, and dynamic best-effort reactive techniques in high-performance). This possible future solutions will demonstrate the benefits of merging concepts from these two domains by evaluating key applications from both segments running on embedded and high-performance platforms. The intent of this paper is to describe the characteristics and the constraints of disaster management (DM) applications for industrial environments. When defining the requirements and their evaluation procedure, a first analysis of the DM applications modules (HW platform, OS and RT engines, monitors and knobs, reliability models) is provided.

From this perspective, this paper focuses on the description of system and application scenario for disaster management processes. The paper is divided in five main sections. Section 2 explains related work and the mitigation tools, basically, two environments for error attenuation. Section 4 provides information about System Scenarios for Disaster Management Processes, section 5 presents application scenarios in HPC environment. Final section 6 shows some results and conclusions, results.

2 Related Work

System scenarios classify system behaviours that are similar from a multi dimensional cost perspective, such as resource requirements, delay, and energy consumption, in such a way that the system can be configured to exploit this cost. At design-time, these scenarios are individually optimized. Mechanisms for predicting the current scenario at run-time and for switching between scenarios are also derived. These are derived from the combination of the behaviour of the application and the application mapping on the system platform.

These scenarios are used to reduce the system cost by exploiting information about what can happen at run-time to make better design decisions at design-time, and to exploit the time-varying behaviour at run-time. While use-case scenarios classify the application's behaviour based on the different ways the system can be used in its over-all context, system scenarios classify the behaviour based on the multi-dimensional cost trade-off during the implementation trajectory [1]. The following sections present system scenarios based on the mapping of an HPC application. The application is stressed with injection of faults and a trade-off between number of mitigated errors and overhead of the used mechanics (Operating System [2] and run-time [3]) is presented.

3 Mitigation Tools

Transient errors are a major concern for the reliable operation of modern digital systems. A new environment is needed for performance evaluation of a software mitigation technique on an industrial grade in many-core computation platform. Error mitigation is performed by re-computing erroneous data in a demand-driven manner with minimal hardware support. Monitoring the runtime performance [3] application is performed illustrating the mitigation overhead without violating the application real time constraints.

3.1 Runtime Environment – RTE

The runtime specifies a firmware that detects transient errors in the executed application. A possible way to detect transient errors is using parity detectors or the use of costly Error-correcting code (ECC) in memories. When a transient error is detected in the execution of a thread, a re-execution of such thread can happen. There are small cache memories where data and instructions are loaded without transient errors. The idea behind this runtime is the fast fine-grain faulty threads re-execution. The runtime daemon has to detect the execution threads with errors in micro seconds. The current state of the art detects the errors at program level and the re-execution time is in the order of milliseconds. In case the number of errors is high, the fine grained mitigation has to provide better performance in terms of executed time and power consumption. In the case when the runtime detects permanent errors in the infrastructure, the re-execution of the threads is performed in other healthy parts of the system. We define the runtime environment as HARPA RTE [4].

3.2 Operating System with RTRM

The HARPA framework [2],[4] is the core of a highly modular and extensible run-time resource manager which provides support for an easy integration and management of multiple applications competing on the usage of one (or more) shared MIMD many-core computation devices. The framework design, which exposes different plug in interfaces, provides support for pluggable policies for both resource scheduling and the management of applications coordination and reconfiguration.

Applications integrated with this framework get a suitable instrumentation to support Design-Space-Exploration (DSE) techniques, which could be used to profile application behaviours to either optimize them at design time or support the identification of optimal QoS requirements goals as well as their run-time monitoring. Suitable platform abstraction layers, built on top of Linux kernel interfaces, allow an easy porting of the framework to different platforms and its integration with specific execution environments. We define this operating System as HARPA OS [4]. The Operating System [2] with an efficient Run-Time Resource Manager (RTRM) exploits a Design-Time Exploration (DSE), which performs an optimal quantization of the configuration space of run-time tunable

applications, identifying a set of configurations. The configuration of a run-time tunable application is defined by a set of parameters. Some of them could impact the application behaviour (for example the calibration of Rainfall-Runoff (RR) models can produce different accuracy for different situations) while other have direct impact on the amount of required resources. For example, the amount of Monte Carlo iterations in uncertainty modelling and the time between batches of simulations lead to different requirements for allocating system resources.

3.3 Error Mitigation Environment

Different combinations of tools produce different trade-offs between the amount of errors mitigated and the computation overhead. The overhead can be measured in terms of extra execution time and power consumption compared with the same system without HARPA. The HPC application without any error mitigation tool can then be used as the baseline scenario to which all other scenarios will be compared. When hardware errors are injected into the platform [5], the results of the simulations are going to be incorrect with a high percentage of confidence. Figure 1 shows an example of the expected results. Where a methodology schema, trade-off between quantity of errors mitigated versus computational overhead and energy versus execution time is shown. The confidence thresholds are provided by the number of injected errors that the system is able to support; below the threshold the framework is able to run the application without providing wrong results with a high percentage of confidence. This confidence threshold can be different for each system scenario and also their computation overhead. When the number of injected errors is higher than this threshold, the framework is not able to provide exact results and/or violates the time constraints. The threshold without error attenuation tools is relatively low compared to the situation when the runtime daemon, the Operating System or both are running on the platform at the same time. We expect an attenuation of the errors which are injected by the model of faults. There is the trade-off between the amount of errors that the platform can support and the overhead in terms of computation. The computation overhead means more lines of executed code, more re-execution of threads, and more re-mapping of threads in healthy parts of the system. This overhead leads to the increase in execution time and power consumption, but the additional code can be used to support better balancing of the workload and a better management of the temperature. Following sections describe different scenarios from the baseline scenario, baseline with error injection, roll-back case with error injection when HARPA runtime is used, the HARPA Operating System case for error mitigation and finally using both HARPA tools (Operating System and runtime) at the same time.

4 System Scenarios for Disaster Management Processes: Rainfall-Runoff Model Case

We can simulate different system scenarios to obtain different trade-offs. The first scenario is baseline or normal execution. This execution provides the correct

values after a period of time and without any hardware injected error. But what happens if we start to inject errors to the normal execution. Then, without any attenuation tool the system will crash. We have to run the HARPA tools to attenuate these errors and still obtain correct results. The following system scenarios will be used for evaluating individual requirements of applications in the HARPA our environment.

4.1 Methodology

The starting point is a parallel code of the DM application running in multiple processes and threads [6,7]. A hybrid OpenMP and MPI [8] is used to deploy the computation to an HPC cluster. The cluster contains a set of x86-64 multi core computers, connected in a fat tree organization, infiniband technology network. The DM application runs under a specific real time operating system and a runtime engine (suggested structure shown in Fig. 1). Both tools manage the workload of the application and help in the attenuation of hardware errors.

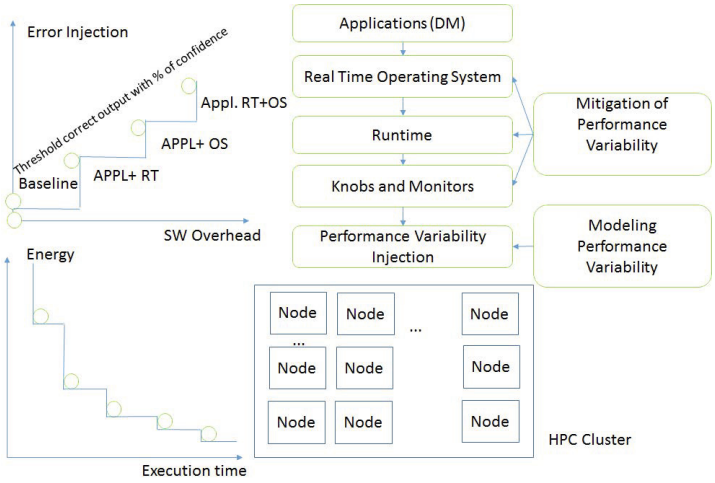

Fig. 1. Methodology Schema, Trade-off: Quantity of errors mitigated vs computational overhead and Energy vs Execution time

The *Real Time Operating System* is in charge of managing the resources. It can benefit from the performance counters in order to take advantage of both workload execution time and energy consumption. The response time of the operating system has to be lower than one second.

The *Runtime Engine engine* actuates the knobs to bias the execution flow as desired, based on the state of the system and the performance (timing / throughput) requirements of the application. It is the runtime engine that implements the various control strategies, aiming to provide dependable-performance in the

presence of (highly) unreliable time-dependent variations. The goal is to exploit different manifestations of what we call as platform slack (i.e., slack in performance, power, energy, temperature, lifetime, and structures/components), in order to ascertain timing guarantees throughout the lifetime of the device (in spite of time-dependent variability) and maintain the expected lifetime of the system. The *runtime engine* must respond to the different knobs, monitors and operating system request in order of one microsecond.

The *knobs and monitors* in the platform allow to modify and control parts of the system like temperature sensors, program counters and provide useful information to the upper layers (RT OS and RTE) about where to map the workload more efficiently. Finally, *Performance variability injection* is a *daemon* that simulates the system as it would be built in a technology of 10nms. This submicron technology has a higher variability and then much more hardware errors appears during the application required execution time. The trade-off between the RTOS and RT engine is that produces an overhead (Fig. 1 up left) in the software, but this new environment will on the other hand enable a higher level of error mitigation. At the design time, we are able to detect which points are optimal in terms of energy vs execution time (Pareto Points[9]). The multi-threaded solution has not to be run on the HPC cluster at the best-effort. A set of intermediate solutions can provide similar solution with lower power consumption and at the same time finish before the pre-established deadline.

Description of Baseline or Normal Execution. It is based on the execution of the HPC application with parallel libraries but without error injection and without any HARPA configuration. This is the default scenario (standard operation of the system before utilization of any HARPA engine). This execution with all the values defines our baseline solution.

Description of Normal Case with Error Injection. In this case, hardware errors are injected to the platforms (HPC system and embedded platform for comparison if possible) while running the application on it. The system operation description is a model of errors used to emulate hardware errors in the platforms under study. The framework has to model errors in the memories, errors in the core CPUs and in the buses. Monitoring of the application when errors are injected: When the number of injected hardware errors increases then the percentage of incorrect solution also increases. Thus, a supervision of the platforms has to be considered under this scenario to find out the threshold of injected errors that still leads to a correct solution provided by the system.

Description of HARPA Runtime: Rollback Case with Errors Injection. In this scenario, errors are injected to the application, but the runtime HARPA daemon is able to detect hardware errors. This runtime daemon also rollbacks the execution of each faulty thread and re-executes it rollback and re-execution of the faulty threads increase the global execution time of the application. There is a threshold level of errors that are acceptable in the system before the required execution time of the program exceeds the threshold. The description of

system operation is related with the HARPA runtime daemon that is in charge of rollback and re-execution of faulty threads. It is also possible to change the clock frequency to the maximum in the part of the system (cores) and during re-execution. System becomes more tolerant in comparison to the system scenarios without rollback. When more errors are injected, more mitigation has to be performed by the daemon. There is a threshold where the output solution provided by the system is still correct (Fig. 1).

Description of HARPA Operating System Case. In this case, application is running in the HARPA OS with the RTRM environment that manages the workload and resource allocation for error mitigation in the system. When errors are injected into the application, the HARPA environment identifies and manages them, decreasing the number of re-executions in the platform. An improvement of this scenario is using procrastination scheduling technique, where task execution is delayed to maximize the duration of idle intervals. This technique [10] has been proposed to minimize leakage energy drain. We will address dynamic slack retrieval techniques under procrastination scheduling to minimize the static and dynamic energy consumption. The description of system operation in this scenario is similar to the baseline scenario with error injection but Run-time Resource Management of the HARPA OS is used as a middleware when the application is running. This middleware allocates the workload of the system at runtime and it works in two steps:

1. design time, a suitable Design Space Exploration (DSE) activity identifies a set of resource requirements that are worth to be considered at runtime, namely Application Working Modes (AWM), and a set of application specific parameters defining different QoS levels, namely Operating Points (OP).
2. run-time, resource management is enforced in a hierarchical and distributed way: For each running application, the RTRM assigns the most promising AWM, as a result of a system-wide multi-objective optimization. The application can optionally perform a QoS fine tuning by switching among its OPs. Platform specific mechanisms (e.g. DVFS) are exploited to avoid risky conditions. Delaying task execution and maximizing idle periods (i.e. decreasing global power consumption) [10].

Description of HARPA OS + HARPA Runtime with Dynamic Slack Case. This new scenario is the combination of the rollback with error injection scenario with the features from HARPA OS. HARPA OS serves as a base for running the application in this extended environment with the rollback daemon that re-executes faulty threads with maximum clock frequency and with dynamic slack. In previous sections, we showed the need to work in different working points to align computation with resources. Fig. 2 shows a node of x86-86 cores with memory and accelerators. With OpenMP *pragmas*, we can decide the resources used to compute the HPC application. There are diverse Pareto

points [9] from a sequential execution to an execution with all resources available (16 cores and the accelerators). In some cases, resources do not have to be available due to temporal or permanent hardware errors (greyed out in Fig. 3). The attenuation tools are able to detect this situation and remap the application in a new Pareto point from a new Pareto curve.

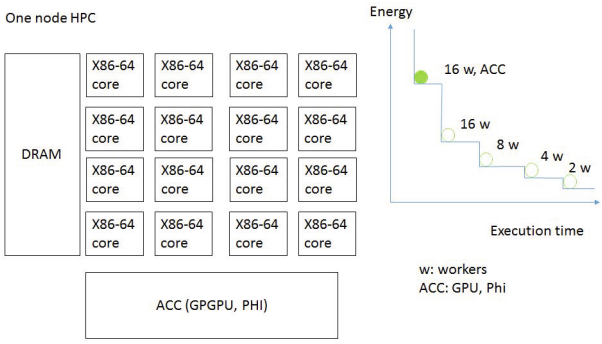

Fig. 2. HPC Node Schema without errors injection and Pareto curves (Energy vs Execution time)

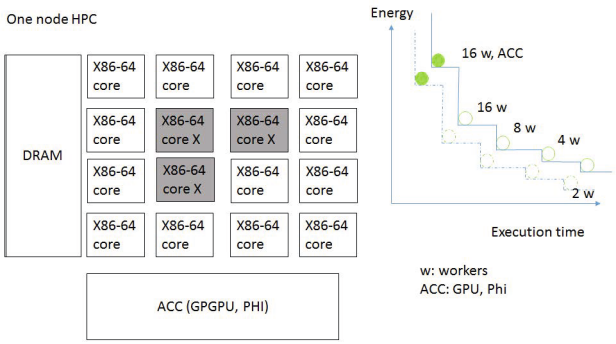

Fig. 3. HPC Node Schema with errors injection and Pareto curves (Energy vs Execution time)

5 Application Scenarios

Application scenarios describe different triggers and states of the application that influence the system responsiveness and operation (e.g. critical flooding level, critical state of patient's health, voice & data, etc.). Based on these scenarios, the system can be in different states with different service level requirements.

5.1 Critical Modules and Their Operation in HPC Environment. Case: Rainfall-Runoff Modelling

One of the applications that are used to develop, test and validate the HARPA architecture is the Floreon+ system [11,12]. It is an automatized flood prediction system for the Moravian-Silesian region in Czech Republic developed at IT4Innovations National Supercomputing Centre. One type of flood prediction models that we are working with are rainfall-runoff (RR) models. Hydrometeorological data for these models are collected from a network of gages and by methods of remote sensing (e.g. radar estimation of precipitation rates) together with the precipitation forecast. Meteorological inputs include the amount of precipitation in particular, air temperature and data about snow pack (thickness and water equivalent) during winter seasons. Hydrological inputs cover data on discharge volumes and water levels from the hydrologic gages. RR models compute how the precipitation over a specified area influences the stream flow over a specified period of time. These results is described by a function of discharge change over time (also called as hydrograph, example in Fig. 4). The complexity of RR models comes with a wide range of problems, one of them being a static setting of model parameters. This setting is done by hydrologists during the creation of model schematization for each modelled geographical area. This process cannot correctly support automatic runs in the system that have to behave accordingly to dynamic changes in weather and current state of the river and its catchment areas. The correct setting of model parameters can be done by inverse modelling and calibration methods, but these methods require repeated executions of the model with different parameters that converge to optimal configuration of these parameters in the model for current situation [12]. Taking these specifics into account, we identified two main application scenarios that support the different workload of the system based on the flood emergency situation.

Standard Operation. In this scenario, weather is favourable and the flood warning level is below the critical threshold. Here, the computation can be relaxed; some kind errors and deviations can be allowed. The system should only use as much power as needed for standard operation; one automatic batch of simulations only has to be finished before the next batch starts. The results do not have to be available as soon as possible, so no excess use of resources is needed. Fig. 4 shows a possible working point (energy vs execution time) where only 8 cores in one node of the HPC cluster are used for the standard operation. It is not necessary to use the accelerators of the system (GPU, and/ or Intel Phi). But resource allocation can be much larger in an emergency operation.

Emergency Operation. Several days of continuous rain raise the water in rivers or a very heavy rainfall on a small area creates new free-flowing streams. These conditions are signalled by the discharge volume exceeding the flood emergency thresholds or precipitation amount exceeding the flash flood emergency thresholds. Much more accurate and frequent computations are needed in this

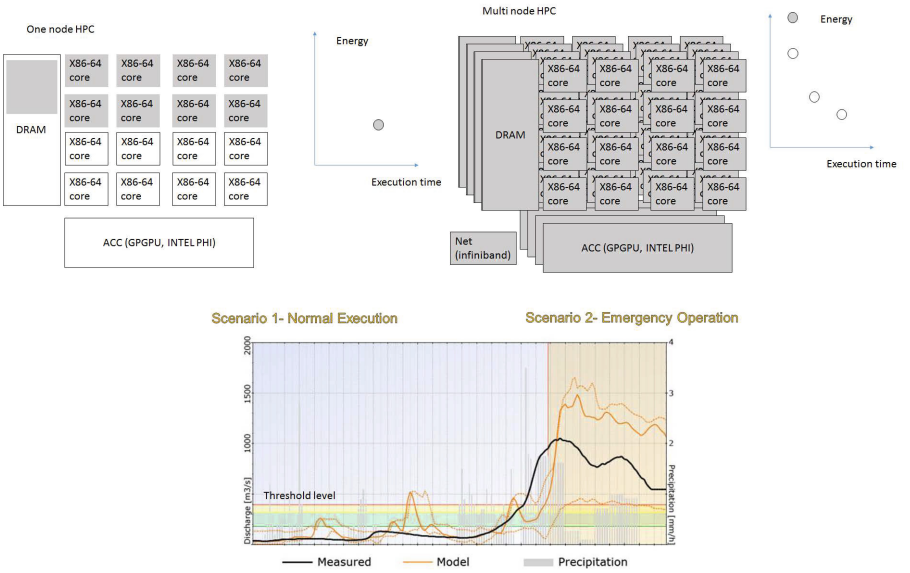

Fig. 4. Scenarios: Normal execution and Emergency Operation

scenario and results should be provided as soon as possible even if excess resources have to be allocated. The Fig. 4 shows real results from a real case [11], where the black curve presents the measured discharge and the orange curve shows the simulated discharge. The threshold level for switching to the emergency scenario depends on the location of the measurement station and is defined by flood activity degrees at the station that are specified by local catchment authorities. The figure also shows that several simulated values at the middle of the simulation exceed the threshold level for a short period while the measured values do not reach the threshold at all.

6 Preliminary Experiments and Conclusions

We have executed several experiments of the baseline scenario (see section 4) to see how the standard runs of RR models perform. Experiments were done on a RR schematization of Ostravice river basin (Czech Republic) with our in-house developed RR model [11] that was parallelized by OpenMP [6] and its calibration parallelized by MPI [7]. The measured input data used for this model were provided by the measurement gages operated by the Czech Odra basin management office [13] and the Medard model [14] was used for precipitation forecast. This configuration serves as the basis for further experimentation with other introduced system scenarios. All experiments were performed on the Anselm supercomputer operated by IT4Innovations. This cluster contains 209 computational nodes where each node is a x86-64 computer, equipped with 16 cores, at least 64GB RAM, and local hard drive. These nodes are interlinked by high

speed InfiniBand (3600MB/s) and Ethernet networks. All nodes share a Lustre parallel file system with a throughput of 6 GB/s. The durations of experimental runs of the baseline scenario models are shown in Table 1.

Table 1. Durations of Experimental Runs of the Baseline Scenario Models

OMP threads per process	Simulation without calibration	Simulation with calibration				
		1 MPI process	2 MPI processes	4 MPI processes	8 MPI processes	16 MPI processes
1	0.5316s	691.5s	404.5s	196.7s	96.5s	59.3s
2	0.3854s	492.4s	288.6s	140.6s	70.0s	46.1s
4	0.2747s	339.5s	200.2s	98.3s	50.2s	33.6s
8	0.2609s	321.1s	185.2s	93.2s	48.6s	47.1s
16	0.3746s	488.2s	307.2s	234.1s	132.3s	143.1s

There are several conclusions in the results concerning the parallelization. The first one is the fact that durations for 16 OMP threads per process are significantly higher than for 8 threads. This shows that the overhead of creating and managing more threads along with the NUMA architecture for each node of the cluster (each node contains 2 independent processors, each with 8 cores) hinders the performance and scalability of the algorithm. Another interesting result is that the gain from 4 to 8 threads is very small and it is not enough to cover the overhead for configurations with more MPI processes. These results show that parallelization of the calibration process is very important for our operational execution of RR models, because there are 4 main basins in the Moravian-Silesian region and we plan to run 3 different RR models for each of these basins automatically every hour. This could not be done without parallelization as it would take more than 2 hours (4 x 3 x 11.5 = 138 minutes if all models would take the same time) to run one batch of simulations, without even considering error mitigation practices introduced in this paper.

Acknowledgements. This article was supported by Operational Programme Education for Competitiveness and co-financed by the European Social Fund within the framework of the project New creative teams in priorities of scientific research, reg. no. CZ.1.07/2.3.00/30.0055, by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by the project Large infrastructures for research, development and innovation of Ministry of Education, Youth and Sports of Czech Republic with reg. no. LM2011033, and by 7 th EU framework programme project no. FP7-612069 HARPA - Harnessing Performance Variability.

References

1. Gheorgita, S.V., Palkovic, M., Hamers, J., Vandecappelle, A., Mamagkakis, S., Basten, T., Eeckhout, L., Corporaal, H., Catthoor, F., Vandeputte, F., De Bosschere, K.: System scenario based design of dynamic embedded systems. Transactions on Design Automation of Electronic Systems (ToDAES) 14(1), Article No. 3 (January 2009)
2. Bellasi, P., Massari, G., Fornaciari, W.: A rtrm proposal for multi/many-core platforms and reconfigurable applications. In: ReCoSoC (2012)
3. Rodopoulos, D., Papanikolaou, A., Catthoor, F., Soudris, D.: Software mitigation of transient errors on the single-chip cloud computer. In: Workshop on Silicon Errors in Logic - System Effects, SELSE (2012)
4. Harpa harnessing performance variability fp7 project (2013), `http://www.harpa-project.eu`
5. Broekaert, F., Sassi, F., Kuchar, S., Portero, A.: D.5.1.- requirements analysis and specification of the project application domain. In: FP7-612069, FP7-ICT-2013-10 (2014)
6. Openmp: Application program interface, version 4.0 (July 2013)
7. Mpi: A message-passing interface standard version 3.0 (2012)
8. Portero, A., Scionti, A., Yu, Z., Faraboschi, P., Concatto, C., Carro, L., Garbade, A., Weis, S., Ungerer, T., Giorgi, R.: Simulating the future kilo-x86-64 core processors and their infrastructure. In: SpringSim(ANSS) (2012)
9. Portero, A., Talavera, G., Moreno, M., Catthoor, F., Carrabina, J.: Methodology for energy-flexibility space exploration and mapping of multimedia applications to single-processor platform styles. IEEE Transactions on Circuits and Systems for Video Technology 8(21), 1027–1039 (2011)
10. Jejurikar, R., Gupta, R.: Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In: ACM (ed.) Proceedings of the 42nd Annual Design Automation Conference DAC 2005, Anaheim, California, USA, pp. 111–116. ACM, New York (2005)
11. Martinovic, J., Kuchar, S., Vondrak, I., Vondrak, V., Nir, B., Unucka, J.: Multiple scenarios computing in the flood prediction system floreon. In: ECMS 2010, pp. 182–188 (2010)
12. Vavrik, R., Theuer, M., Golasowski, M., Kuchar, S., Podhoranyi, M., Vondrak, V.: Automatic calibration of rainfall-runoff models and its parallelization strategie. In: Proceedings of International Conference of Numerical Analysis and Applied Mathematics ICNAAM, Rhodes, Greece (in print, 2014)
13. Czech odra basin management office (August 2014), `http://www.pod.cz`
14. Medard model (August 2014), `http://www.medard-online.cz/index.php`