Just-In-Time Execution Through On-Demand Resource Allocation in HPC Systems

A. Portero, M. Podhoranyi, D. Hrbac
IT4Innovations, National Supercomputer Center
VSB - Technicka univerzita Ostrava 17. listopadu 15/2172
Ostrava-Poruba, The Czech Republic 708 00
antonio.portero@vsb.cz

S. Libutti, G. Massari, W. Fornaciari
DEIB – Politecnico di Milano
Via Ponzio, 34/5
Milano, Italy 20133

ABSTRACT
This article is centred on a mathematical weather forecasting model that must run regularly (i.e. 24/7) on an HPC system. Depending on the environmental conditions, each execution of the model may have a different deadline and a different accuracy requirement. In order to minimize power consumption and heat, we minimize resource allocation as far as the deadlines allow, thus evenly spreading resource usage over time while nonetheless complying with the deadlines. Our work relies on a run-time resource manager that adapts resource allocation to the runtime-variable performance demand of applications. The resource assignment is temperature-aware: the application is dynamically migrated on the coolest cores, and this has a positive impact on the system reliability.

CCS Concepts
• **Computing methodologies → Simulation evaluation**
• **Computing methodologies → Simulation environments**
• **Computing methodologies → Concurrent algorithms.**

Keywords
Parallel execution; HPC; Monte-Carlo simulation; Reliability; Runtime

1. INTRODUCTION
Transistors miniaturization induces increasing power density and higher chip temperatures. This in turn leads to performance degradation, greater device leakage, accelerated chip aging and a significant increase in the system cooling costs. Dynamic thermal management (DTM) addresses thermal hot-spots and temperature variation problems: depending on its current temperature, a processing element (PE) may be accordingly set on a different voltage/frequency configuration.

Our framework is based on the idea of making the application terminate its execution just before the deadline (just-in-time, $_{jit}$). This way, the amount of allocated processing elements is minimized. This in turn allows the resource manager to evenly level power consumption throughout the chip and to dynamically migrate the application on the coolest cores, thus evenly spreading heat and increasing the reliability of the silicon.

ICACS '17, August 10–13, 2017, Jeju Island, Republic of Korea

ACM ISBN 978-1-4503-5284-0/17/08…$15.00
https://doi.org/10.1145/3127942.3127951

2. STATE OF THE ART
Rahimi et al. [1] present a task-scheduling approach that takes into account the measured hardware variability and the system workload to minimize the likelihood of timing errors. Wu et al. [2] propose to employ an OS resident software module to generate power and thermal profiles of the processor. Huang et al. [3] propose an analytical model to estimate the lifetime reliability of multiprocessor platforms when executing periodical tasks. Bolchini et al. [4] present a framework that employs the Monte Carlo (MC) simulation approach to estimate the lifetime reliability of multicore systems. Haghbayan et al. [5] propose a lifetime reliability-aware resource management approach for many-core architectures. The approach is based on a monitor that analyses the aging status of the processing elements and a runtime allocator that suitably maps newly arrived applications on the available resources. Several works also focus on temperature [6, 7]. Ganeshpure et. al [8] propose a solution that employs a temperature prediction scheme to trigger dynamic task rescheduling.

3. RUN-TIME RESOURCE MANAGER
In the context of this work, we employ the HARPA-OS runtime resource manager [9, 10]. HARPA-OS enables the management of multiple applications that compete on the usage of multiple many core computation devices. It also exposes a run-time library [11, 12] that is in charge of: a) synchronizing the execution of applications with the runtime-variable resource allocations; and b) notifying to the resource manager the runtime-variable Quality of Service goals of applications, so that the HARPA-OS scheduling policy, which can be either chosen from a set of predefined ones or implemented from scratch, is able to take into account the feedback coming from applications when computing resource allocations.

We designed and implemented PerDeTemp (PERformance DEgradation TEMPerature), a HARPA-OS scheduling policy that tries to meet the application performance requirements while minimizing resource allocation. When multiple computing resources are available, PerDeTemp employs a multi-objective heuristic to assign to applications only the most healthy and cool cores. Such allocation aims at leveling the power flux over the whole chip, thus mitigating the aging process and avoiding thermal hot-spots.

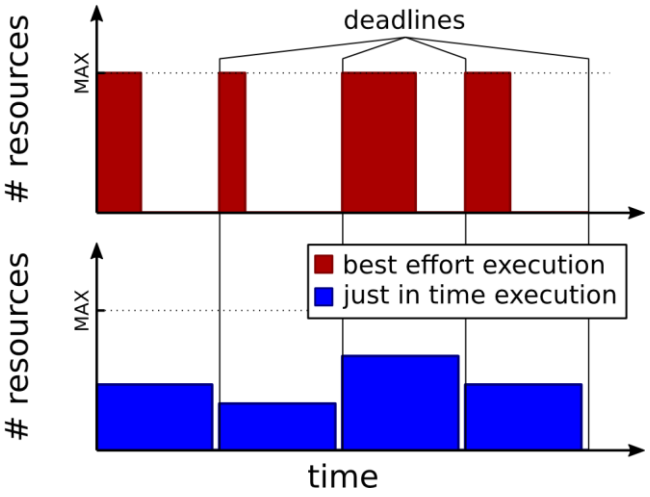

Figure 1. Best-effort vs. JIT scheduling.

Figure 1 shows a comparison between the standard execution of an application and our relaxed, deadline-aware execution. The standard application execution is based on the idea of running the code as fast as possible (best-effort, $_{be}$) using one thread per available processing element. This way, the results are available sooner; however, power consumption is maximized and all the processing elements are stressed. Conversely, our just-in-time execution approach is based on the idea that, since applications are able to dynamically send feedbacks to the resource manager, resource allocation can be made more elastic: it is adjusted over time, so that the runtime-variable performance demand of applications is always complied with, but the execution time of application is always the maximum allowed one (i.e., applications terminate just before their deadline). The resource manager exploits the now-unused resources as a resource pool that can be used in multiple flavors, e.g., to provide cool cores when the next resource allocation will be computed.

The HARPA-OS runtime library, which is linked my managed applications, transparently monitors the applications execution statistics. Among those, one of the most important ones is the average throughput. It is worth noticing that each time HARPA-OS changes the resource allocation of an application, the runtime library re-sets the throughput statistics; hence, the average throughput computed by the runtime library always refers to the current resource allocation. If follows, then, that the average throughput is a very accurate predictor of how the application will behave (i.e., whether the application will terminate or not before the deadline) if the resource allocation remains constant until the application termination.

In order to provide the scheduling policy with a feedback about the current resource allocation, applications use the HARPA-OS run-time library API to retrieve their current execution time and their average throughput. Basing on that values, the applications are able to compare their current throughput, i.e. the average throughput under the current resource allocation, and the ideal throughput, i.e. the throughput that is needed by the application to terminate just before the deadline. This information is periodically sent back to the HARPA-OS as a performance gap:

$$gap_{performance} = \frac{throughput_{current} - throughput_{ideal}}{throughput_{ideal}} \quad (1)$$

Performance gaps greater than 1 mean that the application is executing too fast, which in turn means that the HARPA-OS may seize some of the allocated processing elements and insert them in the pool of empty resources. On the contrary, performance gaps lower than 1 mean that the application needs more resources. In this case, the HARPA-OS takes some of the healthier and cooler processing elements from the unused resources pool and adds them to the set of resources that can be exploited by the application. Finally, performance gaps equal to 1 mean that the application is likely going to terminate just in time. Even in this case, however, the HARPA-OS may decide to change resource allocation, usually to swap the currently allocated set of processing elements with a healthier and cooler one.

4. THE WEATHER FORECASTING MODEL

The application used in this paper is a modular part of the Floreon$^+$ project [13]. The primary objective of the project is to create a platform to support monitoring, modeling, prediction and decision for disaster management. The modularity of the Floreon$^+$ platform allows a simple integration of different thematic areas, regions, and data. The central thematic area of the project is hydrological modeling and prediction.

We have integrated the most computationally demanding module of the Floreon$^+$ platform with HARPA-OS to examine how the HARPA-OS will influence its execution. The selected module models the inaccuracies that affect the Rainfall-runoff (RR) model [14]. It takes as input the precipitation forecast computed by numerical weather prediction models and projects the inaccuracies onto the output of the model by constructing confidence intervals, which are computed using the Monte-Carlo (MC) method.

The confidence intervals accuracy can be positively affected by increasing the number of MC simulations (also referred to as *samples*) and can be determined by estimating the Nash-Sutcliffe model efficiency coefficient [15] between the original simulation output and one of the percentiles selected from the Monte-Carlo results. The percentile simulations describe a possible development of the situation taking possible inaccuracies (along with their probability) into account. For example, the 80% percentile specifies that there is an 80% probability that the real river discharge will be lower or equal to the simulated one. These results can then be propagated further into the flood prediction process, e.g., used as input for hydrodynamic modeling.

4.1 Application Scenarios

Based on the weather, Floreon$^+$ can be subject to different service level requirements. Indeed, the requirements can be translated to the parameters of the uncertainty modeling: a shorter response time in critical situations can, for example, be acquired by decreasing the number of MC samples – which, however, means reducing the precision of the results – or by allocating more computational resources to the application. Depending on the flood emergency situation, we identified three application scenarios that have different requirements. According to their criticality level, we tagged the scenarios as *standard*, *medium*, and *critical*.

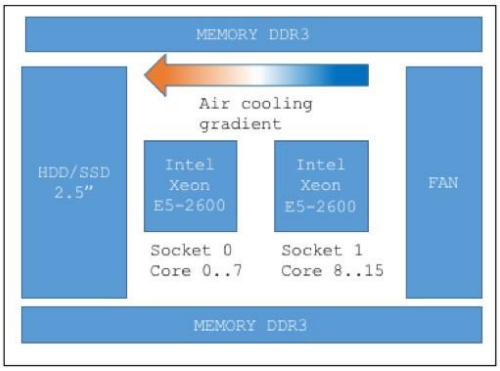

Figure 2. Schema of the blade. Since the fan is on the right of the blade, there is an air cooling gradient between socket 1 and socket 0.

Standard Operation. In this scenario, there is no precipitation, and the flood activity degree is below the critical threshold. In this case, the estimated accuracy can be reduced.

Intermediate Operation. Due to a limited presence of precipitations, the forecast of discharge exceeds a warning threshold. In this case, in order to decrease the uncertainty of the model, the number of MC samples that must be performed by the simulation increases.

Critical Operation. Several days of continuous rain raise the water in rivers or reservoirs. In this situation, more accurate and frequent computations are needed, and results should be provided as soon as possible even if all the computational resources have to be allocated.

5. EXPERIMENTS

In this section, we run the uncertainty module on 16 nodes of an HPC cluster. The module uses a hybrid OpenMP and MPI approach [16, 21, 22] to distribute the computations to multiple nodes. Given that the performance requirements of the application are time-variable (e.g., low when sunny, intermediate/critical when rainy), the HPC center may allocate to the application only some computing nodes and use the remaining ones to execute other applications. Therefore, we performed our experiments in three different configurations: in the first one, we used only one node (i.e., standard operation); in the second, we used two nodes (i.e., intermediate operation); in the third, we used all the cluster (i.e., critical operation).

5.1 Hardware Infrastructure

In our experiments, we use 16 HPC nodes that are connected through InfiniBand. The nodes are part of the chassis described in [17]. Each node is a powerful x86-64 computer equipped with 16 cores (two eight-core Intel Sandy Bridge processors), with 64GB RAM and a local hard drive. The blades in the cluster are the DL510; Figure 2 shows a simplified representation of their air cooling system. Given that the fan is not equally distant from the two sockets, one socket is better cooled than the other one. When the system is idle, the temperature difference between the two sockets is approximately 10 Celsius degrees. The system has several monitors tools installed like power meters, *ganglia* [18] and *likwid* [19].

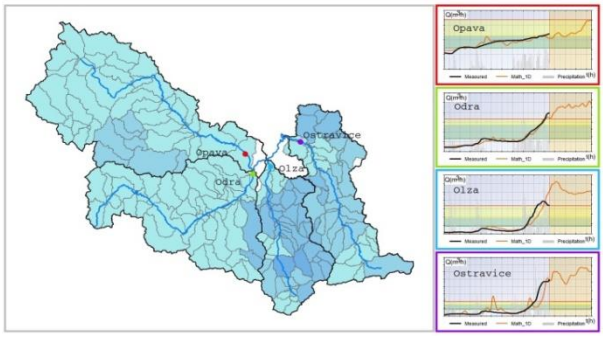

Figure 3. 4 main catchments (left) and outlet hydrographs (right) (black line shows a measured discharge, orange line shows a simulated discharge, X-Axis: time in hours t(h), YAxis: Discharge, cubic meters per hour, Q(m3/h)).

5.2 Catchments Simulation

The experiment monitors the run-time behavior of 4 concurrent instances of the uncertainty module. Each of these instances models the RR uncertainty for a different catchment of the Moravian Silesian region: the Opava, Odra, Ostravice and Olza catchments (see Fig. 3). The catchments are ordered according to the impact in case of flooding (the lower the index, the higher is the importance):

- C_1: *Ostravice* - Functional urban areas with high population density and industrial areas in floodplain zones.
- C_2: *Olza* - Flood sensitive zones in urban areas.
- C_3: *Odra* - Mountains in the upper part of the catchment can cause significant runoff. Less exposed urban areas.
- C_4: *Opava* - Soils with low infiltration capacity.

Each catchment is simulated independently, and individual instances do not interact with each other.

6. RESULTS

Table 1 presents the set of experiments performed in the cluster. The experiments tagged α refer to the single node scenario, while those tagged with β and γ respectively refer to the dual node and entire cluster scenarios.

One node configurations. Figures 4.a.1, 4.b.1, 4.c.1 and 4.d.1 show the number of cores allocated during the 10 minutes execution for each catchment, while Figures 4.a.2, 4.b.2, 4.c.2 and 4.d.2 show the monitored performance gap (дp equations 1) (as already mentioned, the closer to 100% is the gap, the better is the performance).

Table 1. For each experiment, number of Monte Carlo samples to be performed by the instance that models each catchment

	Thousand of MC samples to perform			
Experiment	C_1	C_2	C_3	C_4
α_1	1.5	1.5	1.5	1.5
α_2	1.5	3.5	5.0	7.0
α_3	7.0	7.0	7.0	7.0
β_1	3.5	3.5	3.5	3.5
β_2	7.0	7.0	7.0	7.0
β_3	3.5	7.0	12.0	15.0
γ	80.0 between all catchments			

As shown by the experimental results, the number of allocated resources gets higher as the number of samples of MC to be performed increases. The allocation of resources is satisfactory for $_{\alpha}1$ and $_{\alpha}2$ scenarios, but not for $_{\alpha}3$. In Figure 4.a.2, we can observe that the performance gap is below 100%, meaning that the resources required for this experiment are not enough. In this situation, a second node should be allocated.

Two nodes. Similarly, Figure 5 shows the results for the dual-node configurations. In this case, $_{\beta}2$ has an equal number of samples to operate than $_{\alpha}3$, and, since in this case we have two computing nodes at our disposal, the computation is performed without issues. However, in this case, the resources are again not enough for all the scenarios: Figure 5.b.2 shows that the performance gap of $_{\beta}3$ is below 100%.

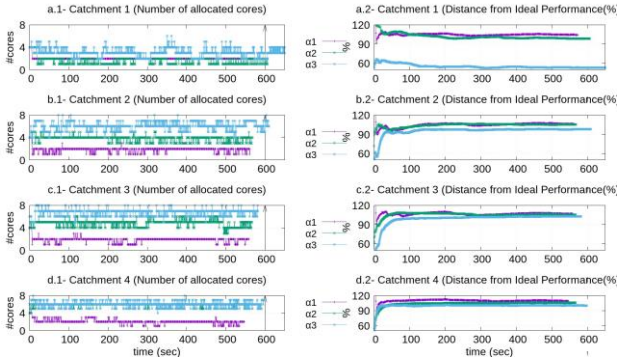

Figure 4. Standard Operation: Light or not precipitation, low water level, (α experiments).

Multi-node results. In the last experiment of this set, we executed the $_{\gamma}$ scenario on 16 nodes with a requirement 80K MC samples. This time, all the instances terminated just-in-time, i.e., exactly at the deadline. Using the $_{likwid}$ power monitor, we monitored the power consumption in both the $_{jit}$ and the $_{be}$ configurations. Whereas just-in-time execution leads to a consumption of maximum 100W, the best-effort execution leads to a peak of 160W per node. Therefore, we saved around 44% in maximum power consumption, while nonetheless complying with the deadline.

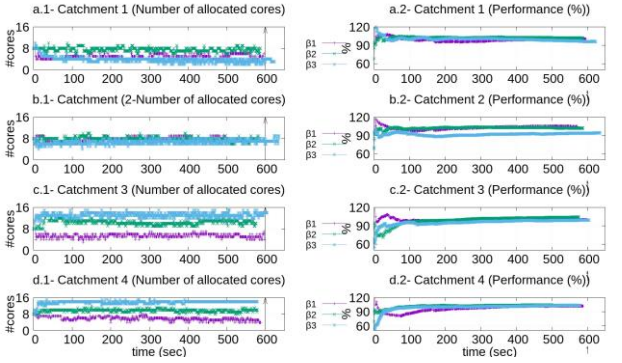

Figure 5. Intermediate Operation: Intermediate execution, medium precipitation or water level warning threshold exceeded (β experiments).

6.1 Reliability

Figures 6 and 7 present the heat-map of a single node when running an uncertainty module instance (12K RR MC samples) using a justin-time and a best effort configuration, respectively. We obtained the heat-maps by using the ganglia monitoring tool. In both figures, the X axis presents time and the Y axis represents the cores IDs. The *Diff per Core* (see left part of the figures) is the difference in temperature per each core c_i ($0<i<16$). The *Diff per Timestep* (lower part of the figures) is the difference in temperature among all cores given a timestep. The *Median* (upper part) provides the median temperature per timestep. As can be seen in Figure 6 (best effort execution with cpufreq performance governor), there is a hotspot in socket 0, which, as already shown in Figure 2, is the socket that is further from the fan. Conversely, Figure 7 shows the execution with HARPA-OS-PerDeTemp. In this case, there are not hotspots and, thanks to the temperature-aware tasks migration performed by PerDeTemp, the temperature is perfectly distributed throughout all the available processing elements.

The presence of hotspots is known to have negative effects on the Mean Time Between Failures of the system (MTBF), and the aging acceleration factor depends on the difference (Δ) of temperature. According to the MTBF estimation presented in [20], running the application in a HARPA-OS-PerDeTemp configuration improves the reliability of the system from 17% to 43% in case of bad cooling (socket 0). With a better cooling (socket 1), the MTBF for the besteffort relatively grows, but it is still 11% to 30% better if we manage the system with *PerDeTemp* ($_{jit}$) instead of using the best-effort ($_{be}$) policy.

7. CONCLUSIONS

In this paper, we presented a framework that, based on the runtimevariable computational resources demand of applications, strives to minimize the resource usage of applications while making them comply with their deadlines. We call this practice *just-in-time execution*, since the applications termination is always as near as possible to the deadline. The framework is able to use the pool of unused resources in multiple fashions, e.g., to swap allocated faulty processing elements with healthier ones, or to periodically assign to applications cool processing elements, hence evenly leveling the generated heat throughput the chip even in case of asymmetrical cooling.

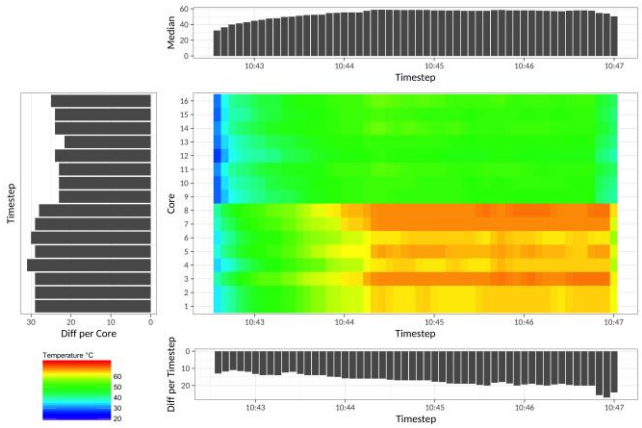

Figure 6. Shows the heat map in the case of best-effort performance GNU/Linux governor (12K in total).

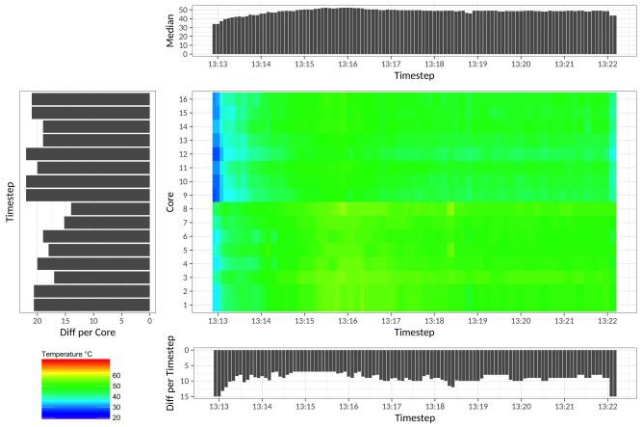

Figure 7. Shows the heat map in case of HARPA Perdetemp runtime with GNU/Linux performance governor (12K in total).

8. ACKNOWLEDGEMENTS

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project fiIT4Innovations excellence in science - LQ1602fi and by the European Union FP-7 program through the HARPA project (grant no. 612069).

9. REFERENCES

[1] Rahimi, A., Cesarini, D., Marongiu, A., Gupta, R. K., & Benini, L. 2015. Task scheduling strategies to mitigate hardware variability in embedded shared memory clusters. In *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*.

[2] Wu, W., Jin, L., Yang, J., Liu, P., & Tan, S. X. D. 2008. Efficient power modeling and software thermal sensing for runtime temperature monitoring. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):25:1–25:29, May 2008.

[3] Huang, L., Yuan, F., & Xu, Q. 2009. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms, DATE ’09.

[4] Bolchini, C., Carminati, M., Gribaudo, M., & Miele, A. 2014. A lightweight and open-source framework for the lifetime estimation of multicore systems. In *IEEE 32nd International Conference on Computer Design (ICCD)*.

[5] Haghbayan, M. H., et al. A Lifetime-Aware Runtime Mapping Approach for Many-core Systems in the Dark Silicon Era, DATE 2016.

[6] Hartman, A. S., & Thomas, D. E. 2012. Lifetime improvement through runtime wear-based task mapping. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS’12, pages 13–22, New York, NY, USA, 2012. ACM.

[7] Oh, D., Kim, N. S., Chen, C. C. P., Davoodi, A., & Hu, Y. H. 2010. Runtime temperature-based power estimation for optimizing throughput of thermal-constrained multi-core processors. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, ASPDAC ’10, pages 593–599, Piscataway, NJ, USA, 2010. IEEE Press.

[8] Ganeshpure, K., & Kundu, S. 2014. Performance-driven dynamic thermal management of mpsoc based on task rescheduling. *ACM Trans. Des. Autom. Electron. Syst.*, 19(2):11:1–11:33, March 2014.

[9] Harpa harnessing performance variability fp7 project, http://www.harpaproject.eu, 2013.

[10] Massari, G., Libutti, S., Portero, A., Vavrik, R., Kuchar, S., Vondrak, V., Borghese, L., Fornaciari, W. 2015. Harnessing Performance Variability: A HPC-oriented Application Scenario, *Euromicro Conference on Digital System Design (DSD) 2015* Funchal, Madeira, Portugal.

[11] Portero, A., Kuchař, Š., Vavřík, R., Golasowski, M., & Vondrá, V. 2014. System and Application Scenarios for Disaster Management Processes, the Rainfall-Runoff Model Case Study. *CISIM* 2014 pp. 315-326.

[12] Bellasi, P., Massari, G., & Fornaciari, W. 2012. A RTMR proposal for multi/many-core platforms and reconfigurable applications. *ReCoSoC 2012*.

[13] Martinovic, J., Kuchar, S., Vondrak, I., Vondrak, V., Nir, B., & Unucka, J. 2010. Multiple Scenarios Computing In The Flood Prediction System FLOREON. *ECMS 2010* pp.182-188.

[14] Golasowski, M., et al. Uncertainty modelling in Rainfall-Runoff simulations based on parallel Monte Carlo method, NNW 2015.

[15] Nash, J. E., and Sutcliffe, J. V. 1970. River flow forecasting through conceptual models part I A discussion of principles, *Journal of Hydrology*, 1970, 10 (3), pp. 282-290.

[16] Portero et al. Using an adaptive and time predictable runtime system for poweraware HPC-oriented applications. *IGSC* 2016

[17] Sliva R., Stanek F. Best Practice Guide Anselm, Bull Extreme Computing at IT4Innovations PRACE, 2013.

[18] Sacerdoti, F. D., Katz, M. J., Massie, M. L., & Culler, D. E. 2003. Wide area cluster monitoring with Ganglia. In *CLUSTER* volume 3, pages 289–289, 2003.

[19] Treibig, J., Hager, G., & Wellein, G. 2010. A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW)*, pages 207–216, 2010, IEEE.

[20] Ellerman, P. 2012. Calculating Reliability using FIT & MTTF: Arrhenius HTOL Model. In *MICROSEMI, Tech. Rep.*, 2012.

[21] Mpi: A message-passing interface standard version 3.0, 2012.

[22] Openmp: Application program interface, version 4.0, 2013.