Parallel Computing: Technology Trends
I. Foster et al. (Eds.)

doi:10.3233/APC200061

Prediction-Based Partitions Evaluation Algorithm for Resource Allocation

Anna PUPYKINA [a,1], Giovanni AGOSTA [a]
[a] *Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy*

Abstract. Resource allocation is a well-known problem, with a large number of research contributions towards efficient utilisation of the massive hardware parallelism using various exact and heuristic approaches. We address the problem of optimising resources usage on deeply heterogeneous platforms in the context of HPC systems running multiple applications with different quality of service levels. Our approach manages the partitioning within a single heterogeneous node aiming at serving as many critical applications as possible while leaving to the upper levels of runtime resource management the decision to preempt resources or to launch the critical application on a different node. We investigate predictive allocation algorithms, allowing to serve up to 20% more high priority requests when using a moving average or machine learning prediction model vs baseline without prediction.

Keywords. NUMA Shared Memory, Resource management, Prediction, Memory management, High Performance Computing

1. Introduction

The push towards Exascale supercomputers is leading to increasingly heterogeneous High Performance Computing (HPC) architectures, characterised by the coupling of accelerators to the more traditional HPC cores. Such future HPC architectures integrating different kinds of hardware accelerators, such as general-purpose graphics processing units (GPGPUs) and reconfigurable computing resources (e.g. Field Programmable Gate Arrays, FPGA), can be classified as *deeply heterogeneous architectures* [1]. At the same time, the new classes of applications, such as real-time high-performance applications, are emerging that require Quality of Service (QoS) guarantees. The typical practice of reserving a subset of the supercomputer to a single application becomes less attractive, leading to the exploration of cloud technologies in the context [2]. So that, a viable scenario is that of multiple applications, with different QoS levels, coexisting on the same deeply heterogeneous HPC infrastructure and sharing accelerators. For this scenario to be successful in practice, resources need to be allocated with a vision that includes both the application requirements and the current and future state of the overall system.

Thus, resource utilisation prediction can be employed to forecast the future state of the cluster based on statistical information on past behaviour. Then the prediction can be used to guide the response to the resource allocation request in the best way to ful-

[1]Corresponding Author: Anna Pupykina, Politecnico di Milano, Via Ponzio 34/5, 20133 Milan, Italy; E-mail: anna.pupykina@polimi.it.

fil the QoS requirements of the applications while optimising the use of resources (primarily, processing elements and memory) in a system-wide perspective. Recent studies in resource utilisation prediction are directed to resource optimisation in cloud architectures [3], on the other hand, research related to the prediction in HPC is mostly focused on the predicting execution time and queue waiting time. More specific prediction for HPC application is presented in [4]. A grammar-based approach for modelling and predicting the I/O behaviour of HPC applications allows to recognise when future I/O operations will occur (i.e., predict the interarrival time between I/O requests), as well as where and how much data will be accessed. [5] gives an in-depth survey of the most recent state-of-art memory management techniques for HPC and Cloud Computing which are used on the different layers of hardware/software stack.

In this work, we focus on the *memory-centric prediction-based partitions evaluation* algorithm for resource allocation on deeply heterogeneous Non-Uniform Memory Access (NUMA) architectures. Here, multiple accelerators coexist within a single node and can cooperate for a single application composed of multiple kernels, or they can be partitioned among different applications. For efficient resource allocation, resource and memory management solutions should be closely interrelated to take into account data dependencies between tasks. The proposed approach regards to the hierarchical resource management strategy that includes the following levels of resource management [1]: **The Global Resource Manager** (GRM) runs on a general-purpose node (GN), and it is in charge of workload balancing and thermal control of the entire system; **The Local Resource Manager** (LRM) runs on the heterogeneous nodes (HN) and on the slave GN, and it is in charge of the allocation of intra-node resources, allowing multiple applications to share resources. Our approach manages resource allocation across different applications within a single *heterogeneous node* as a part of LRM in a way that maximises resource usage while preserving the predictable execution time of critical applications.

The contribution can be summarised as follows: **concept and implementation:** a core concept of Prediction-based Partitions Evaluation Algorithm for resource allocation and several implementations of this concept were derived; **assessment:** a comprehensive assessment was provided, including measurements for a proof of concept implementation, showing the overall feasibility of the approach, as well as its scalability.

The rest of this paper is organised as follows. In Section 2 we briefly introduce the target heterogeneous architecture. In Section 3 we state our problem and describe our proposed solution, while in Section 4 we provide an experimental evaluation. Finally, in Section 5, we draw some conclusions and highlight future research.

2. Background: Runtime management in MANGO Project

The MANGO project aims at addressing the power, performance, and predictability space by dynamically using heterogeneous processing elements in a QoS sensitive computing scenario. The key feature of the MANGO resource management system is its tight integration with the programming model, which lightens the burden on the application developers, as they do not need to handle the mapping of kernels and buffers on suitable HN units [6]. A host-side low-level runtime API allows developers to indicate to the runtime which components (kernels, memory buffers, and synchronisation events) need to be shared within the heterogeneous node. These components are then connected into a

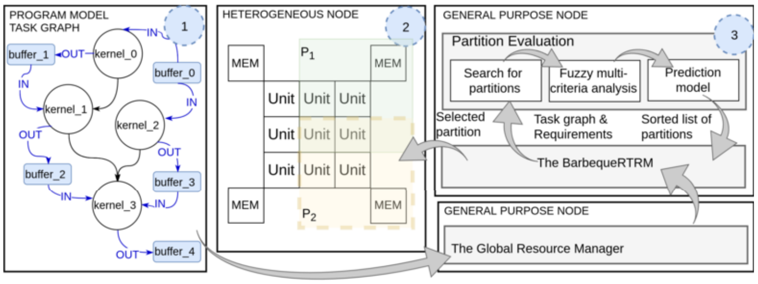

Figure 1. Overview of the target system: (1) task graph, (2) partition list, (3) partition evaluation system.

task-graph to provide the resource manager with the information needed to generate the best feasible resource allocation for the requested QoS. Figure 1 (1) shows an example of a task graph composed of several kernels and buffers. One or more processing units can perform each kernel. A buffer may be used as an output buffer for one kernel and as an input buffer for a different one. Therefore, the resource manager allocates resources based on knowledge of both the system hardware status and the application requirements and priorities.

2.1. Memory awareness

In our target architecture, all the memory modules in a given HN share a single physical address space that can be accessed by all the computational units (ARM-based nodes, GPU-like accelerators, and hardware accelerators) [1]. Despite the shared physical address space, we differentiate the following types of memory buffers: **shared memory buffer** is a memory buffer that may be simultaneously accessed by multiple kernels, such as `buffer_0` on Figure 1 (1); **private memory buffer** is a memory buffer that may be accessed by only one kernel, such as `buffer_4` on Figure 1 (1).

To achieve the optimal unit-memory connection characteristics, we proposed to use a fuzzy multi-criteria analysis with pairwise comparison [7]. The choice between memory units is based on a prediction of the future resource state to maximise the ability to allocate future high priority requests.

We adopt the following criteria for private buffer allocation: c_0^{pr} - distance between processing and memory units (in hops); c_1^{pr} - bandwidth; c_2^{pr} - jitter. Criteria $c_i^{sh}, i = 0..5$ are employed for shared buffer allocation and consist of the mean and standard deviation of the criteria c^{pr}. Fuzzy sets $\widetilde{C}$ are defined on the universal sets of P with the membership functions $m^l(p_i)$ that show the degree of membership of an element $p_i \in P$ to a fuzzy set $\widetilde{c_l}$ for each criterion in C ($l_{pr} = 0..2$ and $l_{sh} = 0..5$) on the basis of pairwise comparisons of elements of P with the relative importance coefficients w_l^{pr} and w_l^{sh}, $\sum w = 1$ for applying the concentration or dilation to the fuzzy sets.

3. Partition Evaluation Algorithm

3.1. Problem Statement

We are given a HN topology, $H = \{U_f, U_b, M_f, M_b\}$, where U_f and U_b indicates the set of free and busy units, respectively, M_f indicates the memory units with all the space available for allocation, M_b indicates the memory units with fully or partially allocated space and a set of task-graphs $Tg = \{tg_0, \ldots, tg_n\}$, $tg =< B, K >$ where B is a set of requested memory buffers and K is a set of kernels. Considering a particular tg, for each buffer $b \in B$ of size $S(b)$ there is a kernel or set of kernels $K(b)$ which uses b (e.g., kernel read buffer $k \xrightarrow{r} b$ and/or write to buffer $k \xrightarrow{w} b$) and a partition or set of partitions $P = \{< M, U >_0, ..., < M, U >_m\}$ appropriate to allocate b on H, where M is a memory unit of size $S(M)$, $S(M) \geq S(B)$, $U \in U_f$ and $\forall k_i \in K \ \exists u_j \in U$ that able to execute k_i. For each kernel $k \in K$ there is a set of preferred target processing architectures $Arch_{prefs}(k) =< Arch^0, ..., Arch^l >$ that is noted by developer. Each application has a specific priority level $appl = \{appl_h, appl_l\}$, where the high priority application $appl_h$ needed to be allocated with the requested QoS on the current HN, and the low priority application $appl_l$ could be rescheduled on the another HN. Figure 1 (2) shows a graphical representation of an example of partitions, which could be produced by the resource allocation algorithm.

All combinations of the preferred processing architectures among with all the possible mappings of the kernel to the specific unit and memory allocations can be found by brute-force exploration. However, this approach can be time consuming other the leading to find a redundant set of mapping solutions. The overall number of mappings can be estimated by the following formula:

$$N_{map} = \prod_{i=1}^{N_K} \frac{((\sum_{a=1}^{arch(i)} N_U^a) - k_i)!}{((\sum_{a=1}^{arch(i)} N_U^a) - (1 + k_i))!} \times (N_M)^{N_B} \tag{1}$$

where $k_i = \sum_{kernel=1}^{i} [\exists kernel = j | j \in 1..i \wedge arch(i) = arch(j)]$ e.g. k_i is the number of kernels for which at least one preferred architecture is the same, N_{map} is the number of mappings, N_K and N_B are respectively the number of kernels and buffers in tg, N_U^a is the number of units with the specified architecture a, N_M is the number of memory units.

Given the size of the solution space, the time needed to find suitable solutions can often be too long for considering the execution at runtime. The heuristic goal is to limit the number of resource partitions to consider, based on the exploitation of historical data about the previous application executions. Without additional constraints, buffers from different task-graphs could be allocated on the same memory unit. This allocation can cause unpredictable interference of concurrent applications on shared memory and routing bandwidth. The easiest way to ensure that the bus access requests are served immediately is to separate resources for concurrent applications. This approach does not solve the interference problem between concurrent tasks of a single application but mitigates the stochastic influence of independent applications.

We aim to find the best $P_s =< M_s, U_s >, P_s \in P$ for each sequentially arriving $tg_i \in Tg$ by the criteria $C = \{C^{pr}, C^{sh}\}$ with the following conditions: **size:** $\sum S(B) \leq \sum S(M_s)$; **isolation:** $M_s \in M_f$ and $\forall m_i, u_j$ such that $m_i \in M_s, u_j \in U_s \ \exists < u_j, m_i >$ and $\forall m_i, u_j$ such that $m_i \in M_s, u_j \in U_b \ !\exists < u_j, m_i >$, where the tuple $< u_j, m_i >$ defines the permitted networking connection between u_j and m_i; **multi-criteria analysis:** $P_s = max_i \left\{ \frac{min_l [m^l(p_i)]^w}{p_i} \right\}$; **prediction model:** the usage of M_f for $appl_l$ is avoided in the case of predicting the use of M_f by $appl_h$.

In the case of heterogeneous systems, significant on-chip constraints, such as limited memory and route bandwidth, need to take into account. At this stage, we only consider

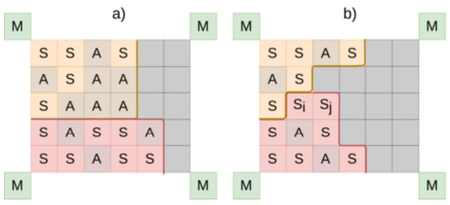

Figure 2. An example of partitions a) with rectangular isolated areas b) with irregular isolated areas (S - selected unit, A - additional unit, M - memory unit).

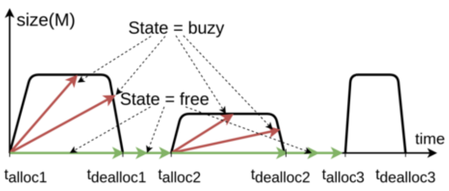

Figure 3. Input vectors and output classes for the prediction models vased on SVM algorithm.

the subproblem of resources allocation to available tiles without taking into account the consequences for the communication. Heuristics have to be used to find a solution with a reasonable quality within an acceptable time. Accordingly, we investigated the partition evaluation method based on memory usage prediction and memory characteristics comparison. A graphical overview of the proposed approach is presented in Figure 1 (3).

3.2. *Isolated partition*

The processing units and the memory units are connected through a 2D-mesh NoC. Each processing unit has four routing ports with an XY routing algorithm implemented to allow network connection between units. In order to limit bandwidth utilisation, processing units have to be allocated close to the memory unit. We proposed to select the processing units in *the nearest von Neumann neighbourhood* to the memory unit so as to ensure the smallest average distance from each processing unit to the memory unit. We consider two ways of forming the isolated area presented in Figure 2: **rectangular:** selected units are supplemented by adjacent ones to form a rectangular area; **irregular:** selected units are supplemented by adjacent ones to provide access to the memory units.

The available routing bandwidth on the boundary is set to 0 so as to avoid bandwidth resources usage from others partition. The rectangular area guarantees a networking connection from each processing unit to the memory unit in the specific partition. However, this isolation method occupies a large number of free processing units. On the other hand, the second approach makes it possible to use processing resources more economically. However, this type of partitioning requires moving from the simple XY routing algorithm to the adaptive XY routing algorithm, since memory unit becomes unreachable for some processing units (as an example units S_i and S_j in Figure 2b).

3.3. *Prediction model*

In general, a resource management system is based on an algorithm that takes runtime decisions on the basis of continuously updated information about the state of the resources. By predicting the future state of resources we can improve the quality of the management decisions [8]. We are suggesting the models that can, at runtime, predict the future use of resources so that management decisions aimed at increasing the service of high priority requests can be made.

Predictions are based on statistical information. These can be seen as statistical series, that is ordered collections of data $Y = \{Y_{i-n-1}, ..., Y_i\}$ beginning at moment t_{i-n-1}

and covering events up to the final moment t_i, where Y_j is a pair $Y_j = (t_j, v_j)$. The first element t_j defines the moment in time and the second element v_j defines the value of one variable of interest (in our case, it is an allocated size of the memory unit). We considered statistical series based on: **time:** time sampling is carried out so that the time between allocations is taken into account; **events:** resources allocation is considered as the occurrence of an event for which only the order of allocation and not the time matters.

Two linear prediction models that can be applied to runtime contexts were implemented: **the moving average method:** $\widetilde{Y_{i+1}} = m_{i-1} + \frac{1}{n}(Y_i - Y_{i-n})$, where m_{i-n} - moving average for n periods before the forecast; **the exponential weighted average method:** $\widetilde{Y_{i+1}} = \alpha Y_i + (1-\alpha)\widetilde{Y_i}$, where α - smoothing constant.

To take into account the adequacy of the prediction model Theil's coefficient of inequality $U = \sum_{i=1}^{n}(Y_i - F_i)^2 / (\sum_{i=1}^{n} F_i^2 + \sum_{i=1}^{n} Y_i^2)$ was chosen. Theil's coefficient takes the value equal to zero when the prediction model is accurate, and the value equal to one when the forecast is inadequate. The proposed evaluation algorithm does not consider prediction in cases coefficient is greater than 0.5.

To assess the possibility of using machine learning in runtime contexts, we used dlib C++ library's [9] implementation of the **pegasos algorithm for online training of SVM**. The prediction model was redefined to a simple binary classification problem in the following way: **input:** model is described as a two-dimensional input vector by the size and time derived from the last allocation; **output:** prediction is represented by two possible states of the units (busy or free) as output classes; **kernel:** radial basis function kernel defines the allocation-deallocation trend. A graphical representation of the input vectors and output classes is shown in Figure 3. In addition to online training, the **trainer for a C-SVM using the SMO algorithm** for solving the same binary classification problems was used.

3.4. Partitions evaluation

The overview of partitions evaluation algorithm is presented in [10] with buffer analysis described in detail in Algorithm 1. The input of the Algorithm 1 receives the quantitative interaction characteristics between the memory unit m onto which the analysed buffer b is mapped and the processing units u_i onto which the kernels that read and/or write to the buffer are mapped, such that $prop_r = (u_i \xrightarrow{r} m).Properties, i = 0..n_r$ and $prop_w = (u_i \xrightarrow{w} m).Properties, i = 0..n_w$. At the first step, it looks through the memory-units characteristics of each analysed partition (line 1). Some of these characteristics change at runtime (e.g., available bandwidth) or are constant (e.g., distance in hops). If the current buffer is used by one kernel privately (line 2), then the value of each criterion $val(c_{pr})$ is saved as-is (lines 3-4). Otherwise, the value of each criterion $val(c_{sh})$ is accumulated by calculating the mean and standard deviation (lines 6-9). In the next part, the matrices of pairwise comparisons M^c are filled. The total number of matrices coincides with the number of criteria. Since the matrix of pairwise comparisons is diagonal, symmetric and transitive, at line 11 all diagonal elements are set to 1, at lines 15-16 and 17-18 elements are calculated as a ratio of the specific criterion values of the two compared partitions i and j depending on the optimisation goal, that is, $c \rightarrow max$, as for bandwidth, or $c \rightarrow min$, as for distance. The degree of membership $dm^c[i]$ is calculated for each partition i (line 20) and for each criterion c (line 21) as one over the sum of the elements in the corresponding column of the matrix M^c (lines 21-23). At line 24, the

ALGORITHM 1: Buffer analysis

```
Data: Memory-units read-write characteristics $< m, prop_r, prop_w >^0, ..., < m, prop_r, prop_w >^n$
Result: Partition scores $s = \{s_0, ..., s_n\}$ for the buffer $b$
for $i = 0$ to $n$ do
    if $prop_r^i.size() + prop_w^i.size() = 1 \vee (prop_{r,w}^i.size() = 1 \wedge prop_r^i = prop_w^i)$ then
        foreach $c_{pr} \in C$ do
            $val(c_{pr}) \Leftarrow (prop_r^i \vee prop_w^i).GetProperty(c_{pr})$ ;
    else
        foreach $c_{sh} \in C$ do
            foreach $prop_r^i$ and $prop_w^i$ do
                $val(c_{sh}) \Leftarrow prop_r^i.GetProperty(c_{sh})$;
                $val(c_{sh}) \Leftarrow prop_w^i.GetProperty(c_{sh})$;
for $i = 0$ to $n$ do
    $M^c[i][i] \Leftarrow 1$ /* matrix of pairwise comparisons */
    for $j = i + 1$ to $n$ do
        foreach $c \in C$ do
            if $c \rightarrow max$ then
                $M^c[i][j] \Leftarrow val(c)[i]/val(c)[j]$;
                $M^c[j][i] \Leftarrow val(c)[j]/val(c)[i]$;
            else
                $M^c[i][j] \Leftarrow val(c)[j]/val(c)[i]$;
                $M^c[j][i] \Leftarrow val(c)[i]/val(c)[j]$;
for $i = 0$ to $n$ do
    foreach $c \in C$ do
        $dm^c[i] \Leftarrow \sum_{j=0}^{n} M^c[i][j]$;
    $dm^c[i] \Leftarrow (1.0 \div dm^c[i])^w$ /* degree of membership */
    $s[i] \Leftarrow \min_c(dm[i])$ /* intersection */
for $i = 0$ to $n$ do
    if $GetPrediction(m_i) = Buzy$ then
        $s[i] \Leftarrow s[i] \times -1.0$;
```

score of evaluated partition is set equal to the minimal value of the corresponding degree of membership. Finally, at lines 25-27, the overall score for each partition is updated according to the predicted memory state.

In the case of prediction errors, the proposed approach acts in the following way. If the appearance of the high priority applications is underpredicted, it allocates a higher number of the low priority applications and a fewer number of the high priority applications. In the case of overprediction of the high priority applications, the number of low priority applications is fewer than it could be allocated.

4. Experimental Evaluation

As the deeply heterogeneous architecture targeted by our work is currently under development, we investigated the proposed partition evaluation algorithm on the singe application execution in [10]. Overall, the proposed approach succeeded in evaluating the best and worst resource mappings. The memory status prediction among with partitions isolation is evaluated through a simulation-based approach.

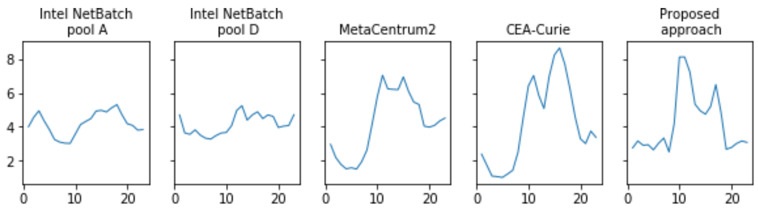

Figure 4. Arrival rates

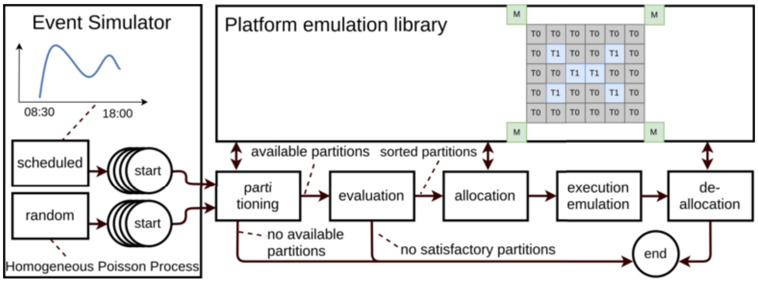

Figure 5. General scheme of the simulation

4.1. Experimental Setup

Event simulator mimics the job submission of users on a time-driven basis. The application pool was composed of workflows with a specific priority. According to our scenario where applications often perform the same tasks multiple times, but additional requests must also be handled, the arrival process was modelled including a mix of scheduled recurring applications targeting the same workflow with the same resources requests and non-recurring applications targeting various workflows. A polynomial function proposed in [11] defined the scheduled arrival process. As shown in Figure 4, accounting records [12] from the national grid of the Czech republic and Curie supercomputer operated by CEA have the arrival process similar to the mentioned above. To model events occurred completely at random at intermittent times, the Poisson process was used. The test tasks flow is consists of 30 days, where weekends are simulated by only non-recurring applications targeting various workflows occurred at random.

The target platform emulation library implements hardware-dependent API and allows performing resource allocation in a simulation mode. The three major resources, including units (processors/accelerators), memory buffers located in DDR memories, and bandwidth, are under control of the local resource manager of the emulator. All these resources are kept as internal configuration and offered for reservation based on their availability and platform restrictions. The selected configuration of the HN includes 30 processing units of two types of architecture placed in five rows and six columns grid and four memory units. The general scheme of the emulation is shown in Figure 5. Here, the experiments employed the same configuration and sequences of emulated applications.

Seven task-graphs presented in Figure 6 were created relying on the synthetic workflow [13]. The task-graphs have a various number of kernels (from 4 up to 13) and input/output buffers for each kernel. Runtime, types of kernels, memory requests and the

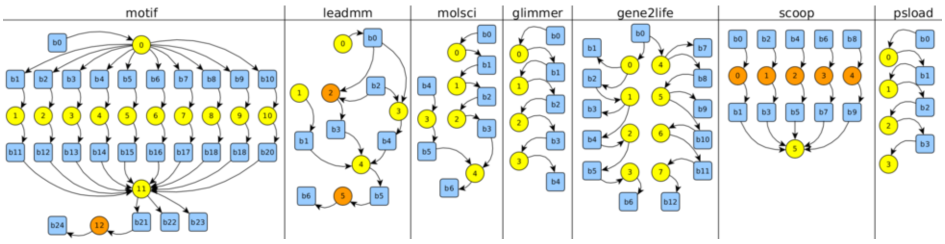

Figure 6. Task-graphs of the synthetic workflows

Table 1. Basic statistics for program simulation

	motif	leadmm	molsci	glimmer	gene2life	scoop	psload
runtime	9090	4990	1020	901	540	98	50
kernels T0/T1	12/1	4/2	5 /0	4/0	8/0	1/5	4/0
memory (MB)	1696	3535	55	2185	2.6	22	379
n. partitions	8.75E+30	1.25E+11	8.36E+10	2.61E+08	1.99E+18	1.81E+10	6.53E+07

maximum number of possible mappings on the experimental architecture according to the equation (1) are presented in Table 1.

In this paper, we consider the design-time mapping policies called recipe [10]. This file is used to specify both the per-task requirements and, optionally, a set of resource mapping solutions that the resource manager should consider at runtime. In order to investigate the time spent by evaluation and allocation algorithms, we have limited the number of mappings in the recipe to 50 and 700.

We analysed the proposed approach with the following prediction models and ways to form statistical series: **Base:** baseline approach without prediction model(PM); **MAonEvent:** approach with PM based on moving average method with event-based statistical series; **MAonTime:** approach with PM based on moving average method with time-based statistical series; **EXPonEvent:** approach with PM based on exponential weighted average method with event-based statistical series; **EXPonTime:** approach with PM based on exponential weighted average method with event-based statistical series; **SVM:** approach with PM based on pegasos algorithm for online training of SVM. **SVMtrain:** approach with PM based on C-SVM training on the 30 days arrival flow simulated in addition to the one used for experiments.

4.2. Experimental Results

First, we investigated the proposed approach with the number of mappings in the recipe limited to 50. As we expected, the resource allocation algorithm without isolation and without prediction models gives the high degree of successful allocations. As shown in Figure 7, the inclusion of the prediction model based on the moving average method in the algorithm without isolation increases by 17% the density of the high priority requests with a decrease by 12% of the overall number of the hosted application. As shown in the Figure 7 (row 3), it is caused by a high rate of rejected low priority application.Other prediction models have no significant impact on the amount of hosted applications (approx 5%). The isolation decreases the number of applications that can be allocated on the HN simultaneously. For instance, rectangular area isolation reduces the number of hosted ap-

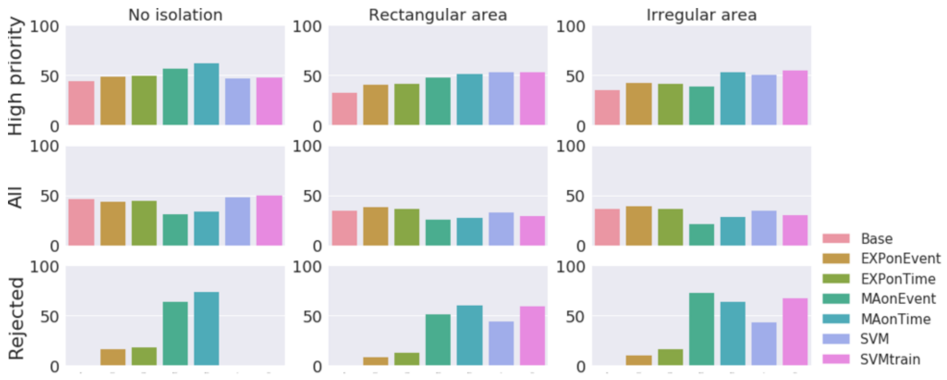

Figure 7. Percentage of the hosted high priority applications (row 1) all successful allocations (row 2) and rejected low priority applications by prediction (row 3) using resource allocation algorithm based on the rectangular and irregular area isolation and without isolation (max 50 mappings per application).

plications by 10%. Thus, for the resource allocation algorithms based on isolation, the prediction model increases the percentage of successful high priority applications. Let us consider hosted high priority applications which use resources that are allocated on the basis of rectangular area isolation using exponential weighted average prediction model. As shown in Figure 7, the percentage of these applications increases by 10% regarding the applications which use resources without prediction model and decreases by less than 5% regarding the applications which do not use isolation. In addition to this, the total amount of hosted applications increases slightly. The prediction model based on the moving average method in the algorithm with isolation shows a significant decrease in the overall number of the hosted application due to a large number of rejects due to the memory state prediction and therefore does not allocate applications with low priority.

The algorithm based on the SVM online training gives the approx. 20% advantage in hosted high priority applications regarding the baseline for the algorithm with rectangular area isolation and approx 8% increase regarding the baseline for the algorithm without isolation. Also, in the case of isolation, the prediction model based on the moving average method gives a large number of rejects due to the memory state prediction and therefore does not allocate applications with low priority. The trained SVM algorithm both for rectangular and irregular area isolations provides approx. the same number of high priority applications as the algorithm with prediction based on the moving average method. At the same time, for irregular area isolation, the amount of high priority application allocation increases by 20% regarding the baseline without prediction model, and increases approx. by 10% regarding the algorithm without isolation and prediction.

Generally, the proposed approach, with the number of mappings in the recipe limited to 50, gives adequate time for evaluation and allocation for considering the execution of the policy at runtime. As shown on Figure 8, algorithms with the prediction model based on the SVM algorithms, with both online and pre-trained learning, give approx. four times higher evaluation time than the algorithms with linear statistical prediction models. Nevertheless, the allocation time for the SVM algorithm with online training is approx. ten times higher than those of the statistical methods.

A more significant number of mappings in the recipe increase both evaluation and allocation time. Figure 9 shows the dramatical increase in evaluation time for algorithms with machine learning prediction models, especially for resource allocation without re-

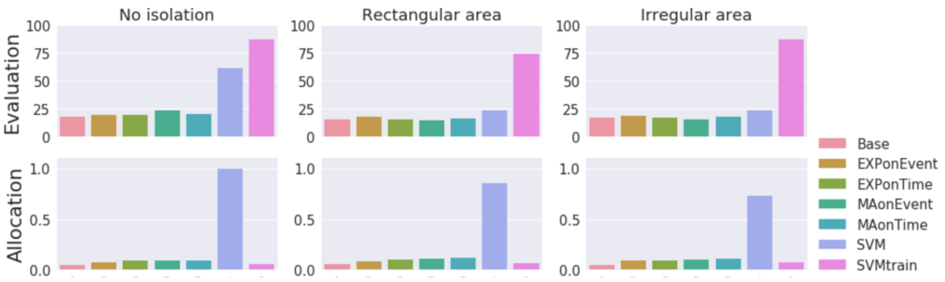

Figure 8. Evaluation (row 1) and allocation (row 2) time (in *ms*) using resource allocation algorithm based on the rectangular and irregular area isolation and without isolation (max 50 mappings per application).

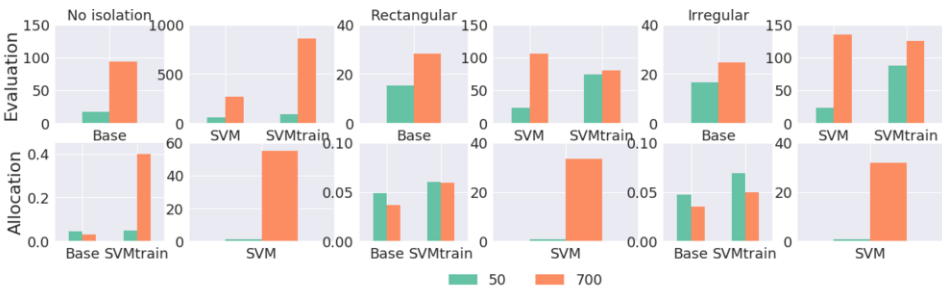

Figure 9. Evaluation (row 1) and allocation (row 2) time (in *ms*) using resource allocation algorithm based on the rectangular and irregular area isolation and without isolation.

source isolation. The reason is in the fewer number of possible mappings with additional conditions on the simultaneous use of memory tile. The allocation time increases rapidly for approach with the prediction model based on the SVM algorithm with online training. It is caused by the time spent on training SVM following each new allocation.

5. Conclusions & Future Developments

In this paper, we have introduced a predictive method for partition evaluation within deeply heterogeneous architectures with NUMA shared memory. The target platform deals with workloads with different priorities for resource allocation requests, classified as a high priority and best effort. Through the use of predictive algorithms, we were able to serve up to 53% of the high priority requests vs a baseline of 32% without prediction on the isolated area. The effort on evaluation and allocation time by statistical prediction model is inessential and is about 15 and 0.1 ms respectively. It is worth noticing that prediction model based on machine learning algorithm with online training gives higher evaluation and allocation time (20 and 0.8 ms respectively) and pre-trained algorithm gives higher evaluation time (up to 100 ms) while allocation time is in the same range as for baseline algorithm. Nevertheless, partition evaluation using the prediction model based on machine learning is still under consideration due to relatively short time per allocation and slightly better results also for allocation without isolation. The proposed approach does not assume the only correct prediction model. We aim using the prediction model and the mapping isolation, depending on the global state of the system. The Global

Resource Manager taking into account the current workload and thermal state of the entire system is in charge of using the specific isolation and prediction policy. In future works, we will provide additional analysis by considering the influence of the knowledge of the possible plan of the application execution on the resources prediction models.

6. Acknowledgments

This research was partially funded by the H2020 EU projects "MANGO" (grant no. 671668) and "RECIPE" (grant no. 801137 [14]).

References

[1] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, E. Cappe *et al.*, "Exploring manycore architectures for next-generation hpc systems through the mango approach," *Microprocessors and Microsystems*, 2018. [Online]. Available: https://doi.org/10.1016/j.micpro.2018.05.011

[2] B. Koller and M. Gienger, "Enhancing high performance computing with cloud concepts and technologies," in *Sustained Simulation Performance 2014*. Springer, 2015, pp. 47–56.

[3] D. Mishra and P. Kulkarni, "A survey of memory management techniques in virtualized systems," *Computer Science Review*, vol. 29, pp. 56–73, aug 2018.

[4] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross, "Omnisc'io: A grammar-based approach to spatial and temporal i/o patterns prediction," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2014, pp. 623–634.

[5] A. Pupykina and G. Agosta, "Survey of memory management techniques for hpc and cloud computing," *IEEE Access*, pp. 1–23, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2954169

[6] G. Agosta, W. Fornaciari, G. Massari, A. Pupykina, F. Reghenzani, and M. Zanella, "Managing heterogeneous resources in hpc systems," in *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, ser. PARMA-DITAM '18. New York, NY, USA: ACM, 2018, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/3183767.3183769

[7] A. Pupykina and G. Agosta, "Optimizing memory management in deeply heterogeneous hpc accelerators," in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, Aug 2017, pp. 291–300. [Online]. Available: https://doi.org/10.1109/ICPPW.2017.49

[8] C. Ababei and M. Ghorbani Moghaddam, "A survey of prediction and classification techniques in multicore processor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, pp. 1–1, 10 2018. [Online]. Available: https://doi.org/10.1109/TPDS.2018.2878699

[9] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755843

[10] G. Massari, A. Pupykina, G. Agosta, and W. Fornaciari, "Predictive resource management for next-generation high-performance computing heterogeneous platforms," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, D. N. Pnevmatikatos, M. Pelcat, and M. Jung, Eds. Cham: Springer International Publishing, 2019, pp. 470–483.

[11] M. Calzarossa and G. Serazzi, "A characterization of the variation in time of workload arrival patterns," *IEEE Transactions on Computers*, vol. C-34, no. 2, pp. 156–162, Feb 1985.

[12] D. Feitelson. (2015) Logs of real parallel workloads from production systems. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html

[13] L. Ramakrishnan and D. Gannon, "A survey of distributed workflow characteristics and resource requirements," Department of Computer Science, School of Informatics Indiana University, Tech. Rep., 2008. [Online]. Available: http://www.cs.indiana.edu/l/www/ftp/techreports/TR671.pdf

[14] W. Fornaciari, G. Agosta, D. Atienza, C. Brandolese, L. Cammoun *et al.*, "Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '18. New York, NY, USA: ACM, 2018, pp. 187–194. [Online]. Available: http://doi.acm.org/10.1145/3229631.3239368