NMPO

NMPO: Near-Memory Computing Profiling and Offloading

Stefano Corda[1,2]/Madhurya Kumaraswamy[1], Ahsan Javed Awan[3], Roel Jordans[1], Akash Kumar[2], Henk Corporaal[1]
[1]Eindhoven University of Technology [2]Technische Universität Dresden [3]Ericsson Research
{s.corda, r.jordans, h.corporaal}@tue.nl, m.kumaraswamy@student.tue.nl
stefano.corda@mailbox.tu-dresden.de akash.kumar@tu-dresden.de ahsan.javed.awan@ericsson.com

***Abstract*—Real-world applications are now processing big-data sets, often bottlenecked by the data movement between the compute units and the main memory. Near-memory computing (NMC), a modern data-centric computational paradigm, can alleviate these bottlenecks, thereby improving the performance of applications. The lack of NMC system availability makes simulators the primary evaluation tool for performance estimation. However, simulators are usually time-consuming, and methods that can reduce this overhead would accelerate the early-stage design process of NMC systems. This work proposes Near-Memory computing Profiling and Offloading (NMPO), a high-level framework capable of predicting NMC offloading suitability employing an ensemble machine learning model. NMPO predicts NMC suitability with an accuracy of 85.6% and, compared to prior works, can reduce the prediction time by using hardware-dependent applications features by up to 3 order of magnitude.**

I. Introduction

Modern big-data applications comprise machine learning, radio-astronomical imaging, and bioinformatics algorithms [1]. These workloads usually impose high compute and data requirements, which may cause bottlenecks. Most of the applications that process large datasets frequently stall in the cache hierarchy due to the data movement between the main memory, and the processor [2]. A proposed solution to this problem is *near-memory computing* (NMC) [3]–[5], which is an opposite computing paradigm to the classical compute-centric being data-centric and performs the computation near the memory, avoiding the data-movement mentioned above. NMC is possible by the recent advancement in memory technologies. Indeed, technologies such as 3D-stacked memory [6] have higher bandwidth, a large number of channels, reduced power consumption, and the possibility to place accelerators on the logic layer of the memory itself [7], [8]. Prior works show how NMC can efficiently be employed to improve the performance of applications such as graph processing [9], numerical simulations [10], machine learning [11], image processing [12], and radio-astronomical imaging [13].

Nevertheless, due to the poor availability of NMC systems, mainly consisting of prototypes, it is challenging to profile and evaluate the suitability of NMC architectures. Predominantly, system designers use simulation techniques to evaluate workloads [14]. These simulators need to be configured for each new DRAM technology and are time-consuming: they can take up to days or even weeks for real-world application with ever-growing big-data datasets. A systematic methodology for identifying any application's NMC suitability helps the programmer in faster early design stage exploration. Therefore, this work proposes a high-level Near-Memory computing Profiling and Offloading (NMPO) framework for evaluating the NMC suitability of applications by employing an ensemble machine learning algorithm. NMPO's goal is to provide a quick estimation of the NMC suitability by training a Random Forest (RF) model with micro-architecture dependent profiling characteristics. NMPO trains and predicts using hardware-dependent characteristics that are usually faster by approximately 2 to 3 orders of magnitude [15] compared to the platform-independent features employed in related work [16]. This huge overhead difference is due to the large memory requirements that hardware-independent analysis needs for certain analysis, while hardware-dependent characterization relies on fast hardware performance counters. Despite this benefit, NMPO still needs to run the time-consuming NMC simulations for training the ML model, and it also needs information about the NMC performance.

Summarizing the paper's contributions:

- NMPO, a fast high-level profiling and offloading framework for NMC systems, to analyze an application in the early design phases and evaluate if it is suitable for NMC offloading. We employ hardware-dependent profiling techniques and ensemble machine learning models with feature selection and hyper-tuning to build the framework.
- NMPO predicts NMC suitability with an accuracy of 85.6%, and it reduces the prediction time by 2 to 3 order of magnitude compared to the state-of-the-art NMC performance prediction model [16].

The paper is structured as follows: *Section II* presents the essential concepts on application characterization, NMC simulation and machine learning models. In *Section III* we explain the adopted methodology. Then, *Section IV* shows our framework evaluation results in terms of accuracy and speed. Related works are discussed in *Section V* and *Section VI* concludes the paper.

II. Background

This section reports the necessary background about performance monitoring counters (*II-A*), NMC simulation (*II-B*) and

ensemble machine learning models (*II-C*).

A. Application characterization

Key application features that are used later for taking offloading decisions can be collected in different ways. The quicker and easier way of evaluating an application on a traditional CPU is using hardware performance monitoring units (PMUs). Modern CPUs have specific programmable components programmed to gather information from different locations of the chip. Currently, a wide range of tools and libraries can be employed for this task, such as PAPI [17], LIKWID [18], and perf [19]. Perf is a ready-to-use utility available in most current Linux distributions. This utility collects an enormous amount of information from the analyzed application, such as cache misses, Clock cycles per Instructions (CPI), and floating-point operations. We summarize the main features that are collected in *Table I*.

TABLE I: *Perf event list of the Host Machine.*

Event name	Units	Event name	Units
power/energy-pkg/	Joules	L1-dcache-loads	countof
power/energy-psys/	Joules	L1-dcache-stores	countof
power/energy-ram/	Joules	L1-icache-load-misses	countof
uncore_imc/data_reads/	MiB	LLC-load-misses	countof
uncore_imc/data_writes/	MiB	context switch	countof
fp_arith_inst_retired	Gflops	App execution time	seconds
branch-instruction/branches	countof	LLC-loads	countof
branch-misses	countof	LLC-store-misses	countof
cache-misses	countof	LLC-stores	countof
cpu-cycles OR cycles	countof	branch-load-misses	countof
instructions	countof	branch-loads	countof
L1-dcache-load-misses	countof	Instructions/cycle	IPC

B. NMC simulation

Since NMC systems adoption is still not widespread, simulators are necessarily employed to determine their performance. Extended versions of Ramulator [11], [12], [14], [16] are utilized because of its easy extendibility, speed and accuracy. Ramulator is a cycle-accurate and portable memory simulator that simulates a wide range of modern DRAM technologies such as HBM (High Bandwidth Memory), HMC (Hyper Memory Cube), and WideIO. *Fig. 1* shows a high-level representation of Ramulator. It consists of a memory controller that takes the simulation's input. This input can be a set of memory traces generated by a CPU simulator such as Zsim [20], which is called standalone mode, or it can be generated by an execution-driven engine such as Gem5 [21], which is named integrated mode. Ramulator's core consists of a tree of DRAM state-machines (left side of *Fig. 1*), where each node is a class instance such as HMC that derives its properties from its parents' nodes. Each DRAM class has a hierarchy of banks, channels, ranks, etc., representing different nodes having a specific label as property. Ramulator-PIM, an extended version of Ramulator, can simulate computing units such as Out of Order (Ooo) cores on the logic layer of 3D-stacked memory.

For the evaluation of power consumption metrics Ramulator is integrated with DRAM power models such as DRAMPower [22]. In *Table II* we summarize the main performance metrics that can be extracted using Ramulator-PIM.

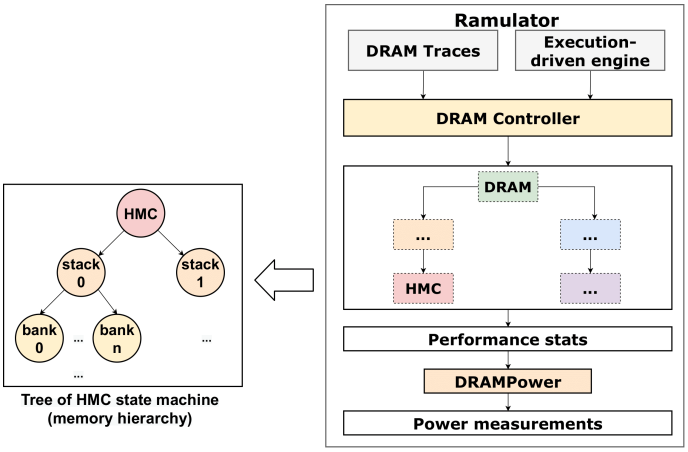

Fig. 1: *High-level overview of Ramulator.*

TABLE II: *NMC system performance metrics.*

Statistic	Units	Category
ramulator.cpu_cycles	cycle	Ramulator-PIM
ramulator.ipc	Instruction/cycle	Ramulator-PIM
ramulator.cpu_instructions	countof	Ramulator-PIM
ramulator.total_time	ns	Ramulator-PIM
Average Power	mW	DRAM Power
Total Trace Energy	pJ	DRAM Power

C. Ensemble machine learning

Complex decisions such as application offloading to suitable accelerators may require sophisticated tools such as machine learning (ML) prediction models. These models are usually trained on a section of the available features dataset and tested on the remaining part. Then, they are employed to predict, make decisions or classify an unknown dataset. While simple models such as a Decision Tree can be effective, in the case of many features, ensemble ML models are more accurate [23]. Ensemble ML models consist of several simple models trained on a different and random subset of the training dataset. The final decision, classification, or prediction is then made by evaluating all the simple models' results by selecting the most common outcome.

Random Forest (RF) [24] is an ensemble ML model that consists of a set of decision trees. RF uses either a categorical response variable, referred to in [25] as "classification", or a continuous response referred to as "regression". Similarly, the predictor variables can be either categorical or continuous. The decision trees are partitioned based on binary recursion. The predictor space uses a sequence of binary splits to partition. The root node contains the whole list of predictors. The splitting criterion gives a measure of "goodness of fit" (regression) or "purity" (classification) for a node, with large values representing poor fit (regression) or an impure node (classification).

RF model performance is boosted by tuning the hyper-parameters, which are characteristics of the model that can impact model accuracy and computational efficiency. These values are set before fitting the model and optimized through trial and error methods like grid search and random search. Multiple models are fitted with several hyper-parameter value sets, their performances are compared, and the best performing one is chosen. The popular hyper-parameters tuned for Random Forest models are; the number of decision trees (N_estimators), the number of features to resample (Max_features), the depth of each tree in the forest (Max_depth), the minimum number of samples required to split each node (Min_samples_split) and the minimum number of samples required for each leaf (Min_samples_leaf).

III. Methodology

The NMPO framework and the experimental setup are described respectively in *Subsection III-A* and in *Subsection III-B*.

A. NMPO

NMPO (see *Fig. 2*) consists mainly of two separate parts: the first one for characterizing the application and training the machine learning model and the second one where the offloading decision is taken by employing the ML model's prediction result. More precisely, in the first phase, the applications are characterized on the host system (❶) employing PMUs and collecting information as reported in *Table I*. Then, the applications are simulated on the NMC system (❷), using Ramulator and DRAMPower to obtain the performance measurements (see *Table II*). The performance metrics gathered from these steps are applied to evaluate the NMC offloading suitability. Thus, we label the data for the machine learning model by our criteria of offloading based on Energy-Delay-Product speedup, which is computed as follows:

$$\text{EDP\_speedup} = \text{Host\_EDP}/\ \text{NMC\_EDP} \tag{1}$$

Accordingly, for the collected training data, we label the offloading decision as "yes" if EDP_speedup > 2, "maybe" if $1 <$ EDP_speedup > 2 and "no" if EDP_speedup < 1. Finally, the machine learning model is trained (❸) using the previous analysis metrics. We employ k-fold validation to evaluate the ML model. For each of the K folds, the model is trained on the remaining (K - 1) folds, which are considered training data and tested on the remaining data or the left-out fold, which serves as the testing data. The performance of the machine learning model is evaluated as the average performance over K-iterations of cross-validation. The hyper-parameters, which are the ML algorithm variables, are tuned to optimize the prediction model's accuracy. The application offloading of the unseen application is performed by first profiling the application Ⓐ, similarly to ❶, on the host system with PMUs (see *Table I*). Then, the trained ML model uses the extracted features to predict the offloading decision on an NMC system Ⓑ. More precisely, since the key feature is the Ramulator IPC (see *Fig. 7*), the ML model predicts this key feature for unseen application by employing RF regression model by using only the host system characterization and later predicts the NMC suitability by classifying the results. The performance of the machine learning model evaluates as the average performance over K-iterations of cross-validation. For example, let the RF ensemble model compute the regression error in predicting the kth part using RMSE and cross-validation score (CV) as:

$$RMSE_k = \sqrt{\frac{\sum_{i \in kth\ \text{part}} (\text{Predicted}_i - \text{Actual}_i)^2}{N}}$$

$$CV = \frac{1}{K} \sum_{k=1}^{K} RMSE_k$$

We shaped the NMC offloading decision as a classification problem, where the key error metric is accuracy that correspond to the ratio of correct prediction and the total number of prediction:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

We also applied the confusion matrix as an alternative tool to better visualize the same information. In the confusion matrix, each row of the matrix represents the instances in a predicted class, each column represents the instances in an actual class. The confusion matrix is named since it makes it easy to see if the system confuses one class for another.

B. Experimental setup

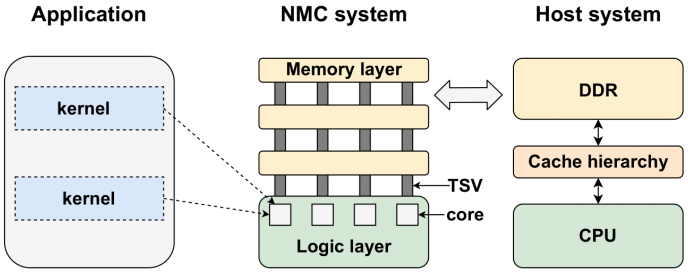

Fig. 3: *System overview.*

Our experimental setups consist mainly of a host processor (see *Fig. 3*), an Intel i9 9900K, and an NMC system with out-of-order (Ooo) cores placed on the logic layer of the HMC memory. The details of the host and NMC system are presented in *Table IV*.

TABLE IV: *Systems parameters.*

Host system	
Intel i9 9900K	8 cores, 2 threads per core, 1 socket, 4.7 GHz, 16 MB L3 cache, 64 GB DDR4 2666 MHz,
NMC system	
Ramulator	8 single issue Ooo cores 1.25 GHz 2-way, 2 cache-lines, 64 B per cache-line 32 vaults, 8 stacked-layers, 256 B row buffer, 4 GB HMC 16-bit full duplex high-speed SerDes I/O link 15 GBps

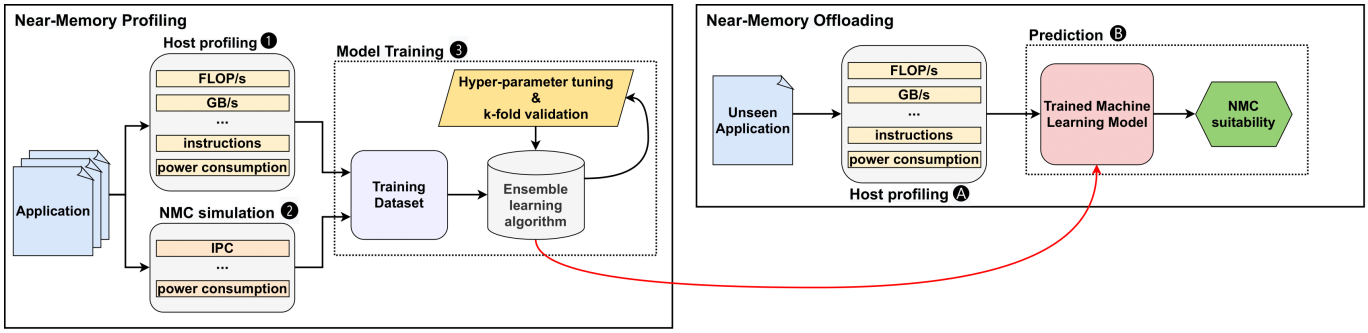

Fig. 2: *Near-Memory Computing Profiling and Offloading (NMPO) overview.*

TABLE III: *Applications and parameters.*

Application			Datasets levels								Time (min)	
Name	Task	Threads	1	2	3	4	5	6	7	**Test**	ML	RT
atax	Computes $A\hat{T}$ times Ax	8,16	4000	6000	8000	10000	12000	14000	16000	17000	3.25	180
chol	Decomposes a matrix to triangular matrices	8,16	1024	1500	2000	2200	2600	3000	3400	4000	6	720
doit	Multiresolution ADaptive NumErical Scientific	8,16	75	100	128	150	200	256	300	350	6.25	5760
gemv	Multiple matrix-vector multiplication	8,16	4000	6000	8000	10000	12000	14000	16000	18000	7.15	186
gesu	Summed matrix-vector multiplications	8,16	4000	6000	8000	10000	12000	14000	16000	18000	8.35	202
mvt	Matrix Vector Product	8,16	4000	6000	8000	10000	12000	14000	16000	18000	7.56	173
syrk	Symmetric rank k update	8,16	1024	1500	2000	2500	2750	3000	3500	4000	9.32	4568
syr2k	Symmetric rank 2k update	8,16	1024	1500	2000	2500	2750	3000	3500	4000	8.4	4898
trmm	Triangular matrix multiplication	8,16	1024	1500	2000	2500	2750	3000	3500	4000	7.35	5280
grid	Radio-astronomical visibilities gridder	8,16	128	256	512	2048	2560	3072	3584	4096	8.15	/
degrid	Radio-astronomical visibilities degriddeg	8,16	128	256	512	2048	2560	3072	3584	4096	8.32	/

The applications are profiled on the host system five times, extracting mean values with the perf package available with Ubuntu 18.04. The NMC system is simulated with Ramulator-PIM [26] once, since the results do not vary in different runs. Power and time parameters for HMC are derived from [22], [27] and fed to the NMC simulator. As a benchmark, we selected a set of application from Polybench since it consists of simple mathematical operations extensively used in modern applications, and are also commonly used in NMC related work [28]. We selected implementation of the benchmark using OpenMP [29], [30], in order to exploit parallelism on the CPU. Aside from synthetic benchmarks, we use the current state-of-the-art gridding, and degridding algorithm for radio-astronomical imaging Image Domain Gridding (IDG) release 0.7 [31], [32]. As shown in *Fig. 3*, we analyzed only the kernel of interest. While Polybench applications have just one kernel, IDG contains different kernel such as gridder and degridder. The benchmarks, their parameters, and the value associated with the different dataset sizes are listed in *Table III*. The datasets are carefully chosen to be large enough to generate DRAM accesses and evaluate whether the application is really suitable for NMC. We also reported the time spent by the machine learning (ML) for the training, hyper-tuning and prediction and the Ramulator simulation time for collecting training data (RT).

IV. Experimental Results and Evaluation

In this section, we discuss the results of application profiling and offloading. Further, empirical evidence in terms of validation and error metrics of the prediction models is presented. Finally, the prediction models are applied to the test cases for identifying the NMC suitability for a target application, thus aiding the users in early design stage explorations.

A. Application profiling

This stage provides the training data required to build and test our machine learning model *Section III-A*. Applications with chosen datasets levels in *Table III* are profiled to collect various statistics from perf, Ramulator-PIM and DRAMPower as discussed in *Section II-A*. The roofline model [33], [34] is a method for capturing the compute-memory ratio of computation and determines if the application is compute-bound or memory bound. The roofline model shows the application's achieved performance (GFLOP/s) and arithmetic intensity (FLOP/Byte) against the machine's maximum achievable performance.

In *Fig. 4* the roofline model of 16 threads test datasets is plotted as an example. Application such as gridder, degridder and doitgen are compute-bound; Symmetric rank update algorithms (syrk and syr2k) are in the DRAM-bound region, whereas the rest of the applications are L3-cache bounded. This tool helps to demonstrate the heterogeneity of the benchmark employed.

In *Fig. 5* Total energy (J) vs Execution time (s) is depicted for all test cases for 16 threads showing the above-mentioned applications heterogeneity. The proposed work uses the energy-delay product (EDP) of host and NMC, where energy is the total energy consumption of cores and delay is the amount of time for executing applications. Then, we compute the EDP Speedup (*Fig. 6* shows only the EDP

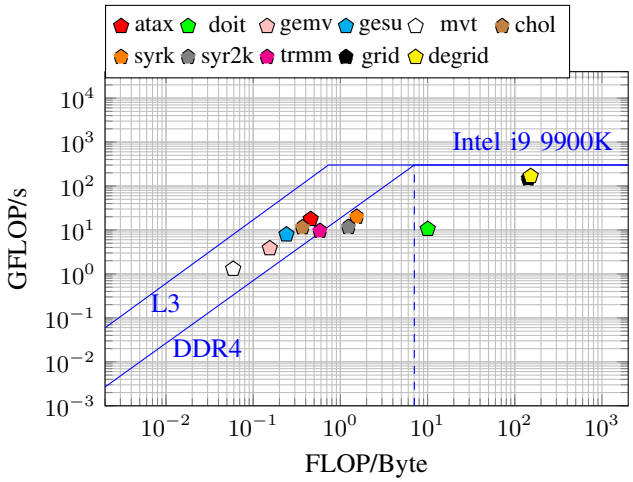

Fig. 4: *Roofline model of test cases using 16 threads.*

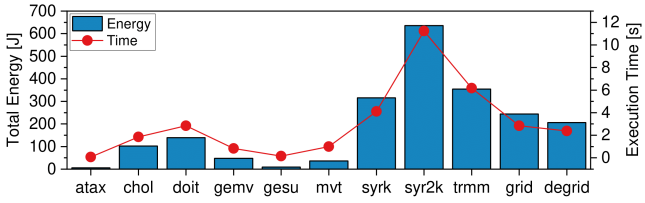

Fig. 5: *Execution time and total energy of test cases on Intel i9 using 16 threads.*

speedup for the test cases using 16 threads) for each training dataset.

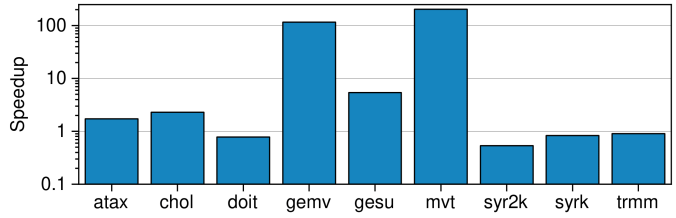

Fig. 6: *EDP Speedup of Polybench test cases using 16 threads.*

B. Application offloading

1) Feature selection: Feature selection methods are intended to reduce the number of input variables to ones that are the most beneficial to a model to predict the target variable. This technique is employed to improve estimators' accuracy scores or boost their performance on very high-dimensional data sets. In our analysis, we selected the essential features using Pearson correlation. It is represented by a number between -1 and 1 that indicates the extent to which two variables are linearly related. A value closer to 1 implies a stronger positive correlation, and a value closer to -1 indicates a negative correlation.

In *Fig. 7*, we show the correlation of the main features we used in this work. It may be easily visible that the correlation is

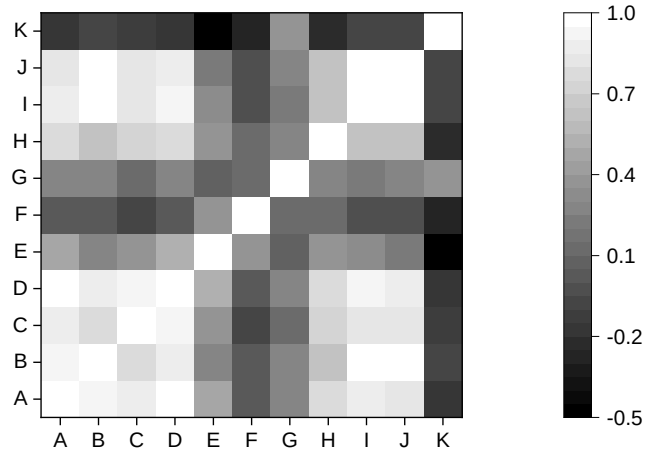

Fig. 7: Correlation plot of input features (see legend in *Table V*).

TABLE V: *Legend for* Fig. 7.

Feature	Symbol
Host Total energy (J)	A
Host EDP	B
Host Total DRAM access (GB)	C
Host FLOPs	D
Host GFLOP/s	E
Host FLOP/B	F
Ramulator IPC	G
Ramulator Total Time (ns)	H
Ramulator/DRAMPower Total trace energy (pJ)	I
Ramulator EDP	J
Speedup	K

equal to 1 for the same metric, while in the other cases is lower. Ramulator IPC is a key factor for making offloading decisions, and indeed it has the highest correlation with the EDP speedup. Since it is time-consuming to run Ramulator each time for a new unseen application or application with a different data set, we deploy an RF regression model to predict the Ramulator IPC and consequently predict the NMC suitability classification. This step is quicker than NMC simulation and enables early design exploration of unseen applications.

2) NMC suitability prediction: After the model is trained, validated and tuned, the final step is to test it on an unseen application. Similarly to [16], we trained the model using the data of all the application *excluding* the one the model will predict. In this manner, the prediction will be more complex, and the application can be considered *unseen*. Since Ramulator is time-consuming and, in particular, it takes several days to simulate for the radio-astronomical imaging algorithms, even with a small image such as 128x128 pixels, we used only Polybench applications for the training (excluding the predicted application if necessary). In particular, for this small dataset, more than 8640 minutes are necessary and large disk space is required, such as a few terabytes. Furthermore, we simulate the above-mentioned small dataset for Gridder and Degridder, which are well-known compute-bound application [32] and will not benefit from NMC in any case to prove the unsuitability of these kernel for NMC offloading. Indeed, their

EDP speedup is small (near 0).

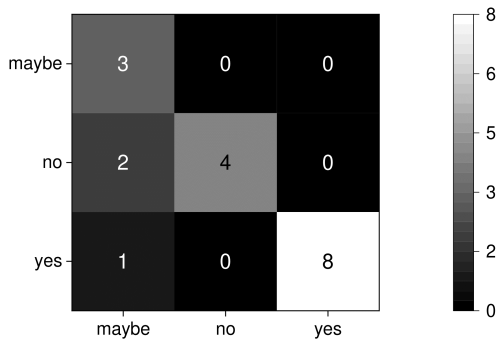

Fig. 8: Confusion Matrix.

The confusion matrix *Fig. 8* reports the distribution of true and false positive of the prediction done by NMPO. For instance, in the bottom row 9 tests are predicted correctly 8 times as "yes" and 1 time as false positive "maybe". The prediction results are slightly related to the roofline model, which is still a good tool for application characterization. Indeed, compute-bound applications do not benefit from NMC, L3 memory bounded application benefit from NMC, and the other applications can benefit from NMC based on the dataset size as applications tend to incur frequent cache misses in L3 and stall on data to be fetched from DRAM [35]

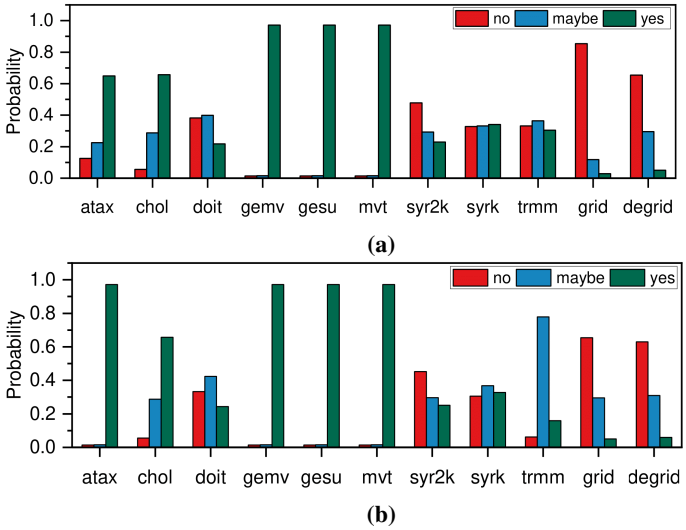

Fig. 9: Model probability of predictions: (a) 8 threads, and (b) 16 threads.

The machine learning model classification probability for the test cases is reported in *Fig. 9* for both 8 and 16 threads test cases. For instance, for the atax test case using 8 threads, the probability of predicting "yes" is about 60%, while for mvt it is 100%. This heavily depends on the training datasets employed.

The overall model accuracy is reported in *Fig. 10* per applications. While some applications have a 100% accuracy,

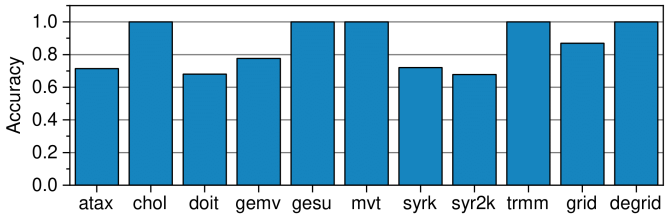

Fig. 10: Accuracy of offloading using NMPO.

some of them are below 80%. In average, the accuracy is 85.6%.

3) Improved estimation for training time: Similar to [16] the main bottleneck in these design space explorations is usually located in the training phase, where the NMC system must be simulated. This procedure usually can take days for a single application for real-world datasets. Furthermore, in [16] the prediction phases consist in characterizing the application employing PISA [15]. However, PISA is slower than PMUs and for specific applications needs more than 64 GB of DDR4, making this step really challenging. We reported in *Fig. 11* the execution time speedup of perf compared to PISA. We employed the datasets reported in *Table VI*, which are smaller compared to the ones in *Table III* but that has value since the PISA overhead increases with the dataset size. We can notice 2 to 3 order of magnitude improvement comparing perf to PISA, thus making the use of perf for the prediction phase more convenient.

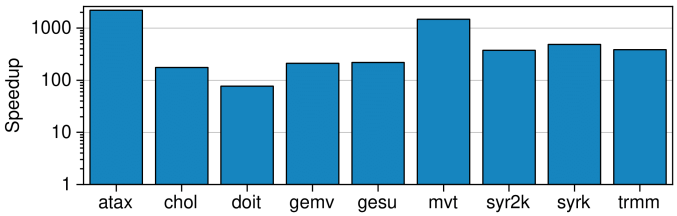

Fig. 11: *Perf vs PISA execution time comparison.*

TABLE VI: *Dataset employed for comparing perf and PISA.*

Application	Dataset	perf time [s]	PISA time [s]
atax	2000	0.23	503.85
chol	512	0.16	28.01
doit	64	0.27	20.78
gemv	2000	0.26	55.62
gesu	2000	0.32	69.27
mvt	2000	0.24	356.16
syrk	512	0.54	201.25
syr2k	512	0.86	416.44
trmm	512	0.36	140.6

V. Related Work

Near-memory computing past works focused mainly on selecting specific memory-bound applications and optimize them with custom architectures on the logic-layer of 3d-stacked memory [4]. A few of them focused on offloading

mechanisms or performance prediction to decide if the NMC system's scheduling is beneficial. We summarize the main related work on application offloading on NMC systems in *Table VII*.

TABLE VII: *NMC offloading related work.*

Name	Year	Offloading	Accelerator	Memory
Zhang et al. [36]	2014	estimation model	GPU	HMC
Ahn et al [37]	2015	compiler and run-time	GPU	HMC
Hsieh et al. [38]	2016	run-time	Ooo cores	HMC
Hadidi et al. [39]	2017	compiler	Fixed function units	HMC
Ahmed et al. [40]	2019	compiler	Fixed function units	HMC
Corda et al. [41]	2019	PCA	in-order cores	HMC
Singh et al. [16]	2019	ML model	in-order cores	HMC

Zhang et al. [36] employ a performance prediction model to decide how to schedule applications on their GPU-based NMC architecture. Ahn et al. [37] propose an offloading ad data mapping mechanism hidden to the programmer. This compiler-based mechanism can efficiently schedule workloads on their NMC-GPU system employing metrics such as memory bandwidth cost-benefit and memory mapping benefits. Hsieh et al. [38] propose an ISA extension to support NMC execution on an NMC system consisting of Ooo cores and HMC. The programmer must use this specific instruction to offload specific instruction to the NMC architecture. Hadidi et al. [39] extend GraphPIM [9] propose a compiler-based mechanism for instruction offloading on CPU/GPU-NMC systems. Ahmed et al. [40] propose a compiler-based mechanism able to detect code sections that reduce the off-chip data movement when accelerated on a CPU connected to HMC. Corda et al. [41] employ PISA-NMC [42], an extended version of PISA capable of extracting metrics related to memory and task parallelism, to evaluate the correlation of these metrics and the NMC offloading suitability using the Principal Component Analysis (PCA). Singh et al. [16] design a high-level framework for predicting unseen application performance on an NMC system. This framework consists of a tuned random-forest model trained with hardware-independent feature and performance on an NMC system with HMC and in-order cores. While the model is capable of predicting the energy-delay-product accurately, prediction is slow. Indeed, this prediction needs to gather the hardware-independent feature of the unseen application using PISA [15], which may take from 2 to 3 orders of magnitude compared to the application's execution time in the host system as we show in *Section IV-B*. We use the hardware-dependent application features collected with a small execution time overhead to predict the NMC offloading suitability to overcome this critical issue. Furthermore, while in [16] specific datasets are so small that they cannot be sampled by perf the PMUs (execution time lower than 0.001s), we selected large datasets that can generate DRAM traffic. This makes it possible to evaluate which applications are suitable for NMC offloading when accessing external DRAM.

Performance prediction of unseen applications on specific architectures is a widely researched topic. However, just some of them focus on NMC. Indeed, as shown in *Table VIII*, most of them focus on CPU and GPU as target offloading architecture. Concerning the machine learning model employed, in past work, linear regression, ANN and random forest have been employed with different tuning options. Similar to Singh et al. [16] and Mariani et al. [43] we employ the random forest algorithm because it can achieve higher prediction accuracy.

TABLE VIII: *Performance prediction related work.*

Name	Year	ML model	Architecture
Joseph et al. [44]	2006	Linear Regression	CPU
Calotoiu et al [45]	2013	Empirical model	CPU
Bailey et al. [46]	2014	Linear Regression	CPU/GPU
Wu et al. [47]	2015	ANN	GPU
Mariani et al. [43]	2017	Random Forest	Cloud HPC
Singh et al. [16]	2019	Random Forest	NMC

VI. Conclusion

We present NMPO, a high-level framework based on ensemble learning models and hardware-dependent profiling that facilitate quick and precise predictions to offload suitable applications to NMC kernels. This framework aids in the early design stage exploration of unseen applications on modern DRAMs like HMC. Unlike slow simulators, NMPO employs an ensemble learning technique called Random Forest with hyper tuning to speculate the offloading of an application. Furthermore, NMPO is much faster than the current state-of-the-art NMC simulator, and other machine learning-based frameworks with platform-independent profiling since hardware-dependent characterization used in NMPO has far less execution time overhead than hardware-independent ones. Thus, NMPO with 85.6% accuracy, quicker analysis and user-friendliness is the go-to ML-based framework for early design stage exploration.

Acknowledgments

This work is funded by the European Commission under Marie Sklodowska-Curie Innovative Training Networks European Industrial Doctorate (Project ID: 676240). We would like to thank Gabor Nemeth from Ericsson Research for his feedback on the draft of the paper.

References

[1] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "Performance characterization of in-memory data analytics on a modern cloud server," in *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, 2015, pp. 1–8.

[2] A. J. Awan, "Performance characterization and optimization of in-memory data analytics on a scale-up server," Ph.D. dissertation, KTH Royal Institute of Technology, 2017.

[3] G. Singh, L. Chelini, S. Corda, A. Javed Awan, S. Stuijk, R. Jordans, H. Corporaal, and A. Boonstra, "A review of near-memory computing architectures: Opportunities and challenges," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 608–617.

[4] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "Near-memory computing: Past, present, and future," *Microprocessors and Microsystems*, vol. 71, p. 102868, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933119300389

[5] O. Mutlu, "Processing data where it makes sense in modern computing systems: Enabling in-memory computation," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 5–6. [Online]. Available: https://doi.org/10.1145/3299874.3322805

[6] D. Lee, G. Pekhimenko, S. Khan, S. Ghose, and O. Mutlu, "Simultaneous multi layer access: A high bandwidth and low cost 3d-stacked memory interface," 2015.

[7] J. T. Pawlowski, "Hybrid memory cube (hmc)," *In HCS*, 2011.

[8] H. Jun, J. Cho, K. Lee, H. Son, K. Kim, H. Jin, and K. Kim, "Hbm (high bandwidth memory) dram technology and architecture," *2017 IEEE International Memory Workshop (IMW)*, pp. 1–4, 2017.

[9] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "Graphpim: Enabling instruction-level pim offloading in graph computing frameworks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 457–468.

[10] J. v. Lunteren, R. Luijten, D. Diamantopoulos, F. Auernhammer, C. Hagleitner, L. Chelini, S. Corda, and G. Singh, "Coherently attached programmable near-memory acceleration platform and its application to stencil processing," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 668–673.

[11] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C. Wu, M. Hempstead, and X. Zhang, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 790–803.

[12] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, "ipim: Programmable in-memory image processing accelerator using near-bank architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 804–817.

[13] S. Corda, B. Veenboer, A. J. Awan, A. Kumar, R. Jordans, and H. Corporaal, "Near memory acceleration on high resolution radio astronomy imaging," in *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, 2020, pp. 1–6.

[14] B. Y. Cho, Y. Kwon, S. Lym, and M. Erez, "Near data acceleration with concurrent host access," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA '20. IEEE Press, 2020, p. 818–831. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00072

[15] A. Anghel, L. M. Vasilescu, R. Jongerius, G. Dittmann, and G. Mariani, "An instrumentation approach for hardware-agnostic software characterization," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2742854.2742859

[16] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "Napel: Near-memory computing application performance prediction via ensemble learning," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[17] D. Terpstra, H. Jagode, H. You, and J. J. Dongarra, "Collecting performance data with PAPI-C," in *Tools for High Performance Computing 2009 - Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing, September 2009, ZIH, Dresden*, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds. Springer, 2009, pp. 157–173. [Online]. Available: https://doi.org/10.1007/978-3-642-11261-4_11

[18] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops*, ser. ICPPW '10. USA: IEEE Computer Society, 2010, p. 207–216. [Online]. Available: https://doi.org/10.1109/ICPPW.2010.38

[19] B. Gregg, "Performance counters," *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018. [Online]. Available: http://www.brendangregg.com/perf.html

[20] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 475–486. [Online]. Available: https://doi.org/10.1145/2485922.2485963

[21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[22] Y. L. Karthik Chandrasekar, Christian Weis and K. Goossens, "Drampower: Open-source dram power & energy estimation tool." [Online]. Available: http://www.drampower.info

[23] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, ser. MCS '00. Berlin, Heidelberg: Springer-Verlag, 2000, p. 1–15.

[24] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, p. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[25] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[26] CMU-SAFARI, "Zsim+ramulator - a processing-in-memory simulation framework," https://github.com/CMU-SAFARI/ramulator-pim, 2020.

[27] C. Weis, A. Mutaal, O. Naji, M. Jung, A. Hansson, and N. Wehn, "Dramspec: A high-level dram timing, power and area exploration tool," *Int. J. Parallel Program.*, vol. 45, no. 6, p. 1566–1591, Dec. 2017. [Online]. Available: https://doi.org/10.1007/s10766-016-0473-y

[28] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling techniques for gpu architectures with processing-in-memory capabilities," in *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2016, pp. 31–44.

[29] Cavazos-lab, "Polybench/acc," https://github.com/cavazos-lab/PolyBench-ACC, 2016.

[30] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," in *2012 Innovative Parallel Computing (InPar)*, 2012, pp. 1–10.

[31] ASTRON, "Image domain gridding (idg)," https://gitlab.com/astron-idg/idg, 2020.

[32] B. Veenboer and J. Romein, "Radio-astronomical imaging on graphics processors," *Astronomy and Computing*, vol. 32, p. 100386, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213133720300408

[33] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[34] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, and M. Püschel, "Applying the roofline model," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 76–85.

[35] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "How data volume affects spark based data analytics on a scale-up server," in *BPOE*. Springer, 2015, pp. 81–92.

[36] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "Top-pim: Throughput-oriented programmable processing in memory," in *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 85–98. [Online]. Available: https://doi.org/10.1145/2600212.2600213

[37] K. Hsieh, E. Ebrahim, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[38] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.

[39] R. Hadidi, L. Nai, H. Kim, and H. Kim, "Cairo: A compiler-assisted technique for enabling instruction-level offloading of processing-in-memory," *ACM Trans. Archit. Code Optim.*, 2017.

[40] H. Ahmed, P. C. Santos, J. P. C. Lima, R. F. Moura, M. A. Z. Alves, A. C. S. Beck, and L. Carro, "A compiler for automatic selection of suitable processing-in-memory instructions," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 564–569.

[41] S. Corda, G. Singh, A. J. Awan, R. Jordans, and H. Corporaal, "Platform independent software analysis for near memory computing," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 606–609.

[42] ——, "Memory and parallelism analysis using a platform-independent approach," in *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 23–26. [Online]. Available: https://doi.org/10.1145/3323439.3323988

[43] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting cloud performance for hpc applications: A user-oriented approach," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 524–533.

[44] P. J. Joseph, Kapil Vaswani, and M. J. Thazhuthaveetil, "Construction and use of linear regression models for processor performance analysis," in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, 2006, pp. 99–108.

[45] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[46] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. De Supinski, "Adaptive configuration selection for power-constrained heterogeneous systems," in *2014 43rd International Conference on Parallel Processing*, 2014, pp. 371–380.

[47] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "Gpgpu performance and power estimation using machine learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 564–576.