Near Data Processing Performance Improvement Prediction via Metric-Based Workload Classification

Dimitrios Papalekas
Electrical and Computer Engineering
University of Thessaly
Volos, Greece
papaleka@e-ce.uth.gr

Athanasios Tziouvaras
Electrical and Computer Engineering
University of Thessaly
Volos, Greece
attziouv@e-ce.uth.gr

George Floros
Electrical and Computer Engineering
University of Thessaly
Volos, Greece
gefloros@e-ce.uth.gr

Georgios Dimitriou
Informatics and Telecommunications
University of Thessaly
Lamia, Greece
dimitriu@cs.uth.gr

Michael Dossis
Computer Science
University of W. Macedonia
Kastoria, Greece
mdossis@uowm.gr

Georgios Stamoulis
Electrical and Computer Engineering
University of Thessaly
Volos, Greece
georges@e-ce.uth.gr

***Abstract*—Contrary to the improvement of CPU capabilities, traditional DRAM evolution faced significant challenges that render it the main performance bottleneck in contemporary systems. Data-Intensive applications such as Machine Learning and Graph Processing algorithms depend on time and energy consuming transactions between the memory bus and the CPU caches. The emergence of 3D-Stacked memories that provide a very high bandwidth led to the exploration of the Process-In-Memory (PIM) paradigm where logic is added to the memory die and data are being processed where they reside. To fully exploit this model, there is a need to methodically determine the portions of code that are better fitted for Near-Data-Processing (NDP). To this extend, in this work, after presenting the key trends of the research field and examine proposed criteria, we simplify the process of a priori decision of a block's suitability by proposing a two-step metric-based application categorization able to predict the applications behavior when offloaded for NDP.**

***Index Terms*—Near-Data-Processing, Processing-In-Memory, 3D-Stacked memories**

I. Introduction

For decades now, the performance of CPUs has been improving at a very fast rate, minimizing the energy and time cost to perform demanding arithmetic operations. To execute these computations, all the data involved must be located in the core's cache memory. Data movement from the main memory to the processing cores can be up to four orders of magnitudes slower, hence energy consuming also. Nowadays the datasets of modern applications are immensely growing, increasing the need for data movement from the RAM to CPU and thus the number of cache misses [1].

Traditional DRAM modules failed to scale efficiently in terms of performance, energy and capacity altogether. As a result, the focus on computer systems research is to the Near-Data-Processing (NDP) paradigm [2], [3], that suggest the processing of data where they reside. Although NDP has been proposed for more than 50 years recent breakthroughs such as 3D-stacked memory [4] led to the realization of the concept. This paradigm aims to facilitate applications that are characterized by irregular access patterns, little memory locality and larger working sets [5].

NDP is divided to Processing-Using-Memory (PUM), that exploits simple bitwise operations in existing memory cells, and Processing-Near-Memory that integrates a logic layer directly underneath a 3D-stacked memory. The latter is further divided to instruction-offloading and function-offloading granularity. Our approach focuses on function-offloading as it the less demanding from the programmer's standpoint and requires no hardware adjustments, facilitating the widespread adoption of the paradigm.

To defend our proposition, in Section II we present a comprehensive analysis of the most prolific trends in the NDP research field, emphasizing on the DAMOV simulator environment [6]. Later on, in Section III, we proceed to simplify the existing profiling methods and keep the process solely in the simulator's environment, by proposing a two-step categorization method that is based on the metrics of arithmetic intensity (AI) and Misses-Per-Kilo-Instructions (MPKI) and is able to efficiently predict the NDP suitability of an algorithm. In Section IV we present the results of our simulations, and we finally give our conclusions in Section V.

II. NDP Trends

A. Overview

Since NDP was first introduced about five decades ago there have been many different methodologies proposed, each of them aiming to address a separate bottleneck of the processor-centric design. The majority of them attempt to eliminate unnecessary data movement between memory and processing components, in order to override the time consuming off-chip link. Based on whether they operate by impeding logic to existing DRAM modules or they implement a 3D memory-logic module, they are divided into Process-Using-Memory (PUM) and Process-In-Memory (PIM). The latter are further divided into instruction offloading and function offloading based on granularity, i.e. the level of insight of the candidate section of code.

The following analysis is a coherent assessment of some of the most significant propositions, nevertheless concentrates on the offloading candidate's selection methodologies which

they suggest, examining the issue more from the programmers' standpoint, since it is one of the most constraining concerns in the direction of PIM adoption.

B. Process-Using-Memory

Although 3D-stacked memory architectures revolutionized the PIM research field, there have been proposed approaches that attempt to take advantage of existing DRAM architecture to induce simple computation capabilities by implementing minimal changes to the memory chips. Amongst data intensive applications there is a fraction that comprises almost exclusively of bulk data movement operations, thus, blocks of instructions that require no computation such as batch initialization of a memory block or bulk data copy.

RowClone [7] implements a mechanism that issues a row-open request to multiple rows of a DRAM subarray and activates the source and the destination row back-to-back. A Pipelined Serial Mode transfers a large number of bytes from the source to the destination, reducing the number of requests needed. Ambit [8] is an extension of RowClone that tries to include bitwise operations in its range of application. Using a Triple-Row-Activation, the cells of the first two rows can participate as operand in bitwise majority functions and return the result to the third row. Tested on bitwise operations both RowClone and Ambit achieved a 11.6× speedup, but despite their results their applicability is limited to the aforementioned operations and their success relies on carefully carving the requests to fit well with the memory row size, a problem that lies within the programmer's or the compiler's considerations.

C. Process-In-Memory

1) Instruction-Level-Offloading: Many algorithms that are known to stress the memory subsystem utilize simple Read-Write-Modify (RWM) operations such as integer addition or equality checks. The support of RWM operations from the Hybrid Memory Cube (HMC) 3D-stacked memory model via Atomic Instructions made this category of applications the prevailing subject of instruction-offloading PIM research. Graph processing algorithms are infamous to invoke a significant amount of cache misses because of the random memory access patterns they produce, and are hence classic examples where this technique has been applied.

Two models of instruction-offloading PIM are the GraphPIM and CAIRO. GraphPIM [9] split the algorithm into three parts. The first and the third parts refer to the vertex's metadata so spatial locality and data presence in cache are presumable. The second one though is related to accessing other nodes and produces randomness in the access patterns. Most of the time it consists of simple RMW commands such as comparisons and additions. GraphPIM implements an instruction-level offloading mechanism that uses the HMC-Atomic commands to perform these RMW operations directly on the memory and bypass the need for data transfers. This methodology can entrain a 2.5× speedup over the baseline PIM model but the need from the programmer to analyze the algorithm and identify suitable instructions limits its applicability. CAIRO [10] functions like an extension of GraphPIM that proposed a compile-time mechanism to automatically identify PIM candidate instructions by performing five tests per instruction. These tests guarantee that the relying instruction can be mapped to an HMC Atomic command and none of the referenced data are located in a CPU register, thus, it is not accessible from the PIM.

We chose to analyze these two cases in order to showcase that instruction-level offloading is based on sound mathematical foundations but requires either the programmer to analyze the algorithm or strict offloading conditions to be met, affecting only a small group of instructions.

2) Function-Level-Offloading: In order to facilitate NDP offloading from the programmer's standpoint, many efforts concentrate on the creation of tools that will be able to automatically distinguish PIM candidate code blocks. A way to achieve the identification of such blocks is to run a profiling tool beforehand and configure the PIM execution based on the experiments result. For example, Intel's V-Tune can effectively detect if the application under examination is compute or memory bound. By inspecting those numbers and combining them with proposed research metrics, one can carry out more effective experiments. An alternative way is via the creation of compiler-based techniques that aim to unify the process of both identifying and executing PIM code blocks completely dismissing the programmer from any extra effort. Albeit, extending the compiler's functionality to include these capabilities requires several modifications to be made in the existing hardware which naturally stands as an obstacle to widespread PIM adoption.

Although all proposed techniques converge to the fact that transaction bandwidth should be the main criterion of choice, none of them has been successful to systematically recognize the best fits for Near Data Processing among bandwidth-bound applications. So far application categorization is based on experimental analysis and practical observations of execution results, so many research groups attempt to provide a large domain of tested benchmarks and conclude based on their output.

Transparent Offloading and Mapping (TOM) [11] develops compiler-based techniques to offload bandwidth-intensive computations in GPUs by adding two extra mechanisms. The first one is a compiler extension that can identify loops which have the potential to be memory-bound based on the number of load/store word instructions they produce. Additionally, based on the observation that loops generate paternal memory accesses, often with spatial locality, a second mechanism copies the memory pages accessed inside the 3D-stacked memory to minimize the memory access time. The actual offloading is performed by a series of added hardware components that utilize the aforementioned run-time information.

To assist the widespread adoption of PIM techniques in modern memory systems, a more general approach of workload identification emerged. Ideally, the methodology should not require any programmer's knowledge of the underlying offloaded algorithm. DAMOV consists a comprehensive analysis in this direction. The DAMOV team developed the first open-source function-offloading granularity simulator. Furthermore, they used external profiling tools and well-established metrics, such as the roofline model to extract information about an application's dependency on the memory system. The main focus of their work is to methodically understand the reason behind any memory related slowdowns an algorithm may experience. For example, after traversing through a number of

processing cores ranging from 4 to 256, they investigate the total cache misses. If this number of misses decreases as the cores count increases, that means that the application makes a good utilization of the caches. They continue accordingly until they end up generating six function categories based on the source of the data transfer bottleneck.

Despite being essential to identify the exact reason of slowdown, an extensive methodology like this is not necessary if the primary goal is to define if the application can exploit NDP to improve its performance. To this extent, the methodology should be as simple as it gets, using only the minimum number of metrics and the classification steps actually needed.

III. Methodology and Metrics

All the above-mentioned works use either external profiling tools, isolated metrics, or concentrate on a specific category of applications. Thus, a need has emerged to constructively categorize applications, so that a programmer can decide on whether the algorithm could benefit from PIM, just by identifying in which class it belongs.

Prior works have come up with indicative metrics that can be observed as a guide for efficient workload profiling. Such metrics are:

1) The Arithmetic Intensity (AI). This metric indicates whether the application under examination is considered compute-bound or memory-bound. For an application to be considered compute-bound, the total time consumed performing computations must outweigh the time spent for data transfers by a significant amount. AI is defined as the quotient of instructions a CPU performs divided by the total bytes accessed in the main memory.
2) The Last Level Cache Misses Per Kilo Instruction (MPKI). This metric is used as an indicator of an application's dependence from the memory system. MPKI can be derived as the quotient of the number of the cache misses occurred during the execution divided by the average instructions among all processing cores. A high MPKI value can either mean that the application produces accesses to memory addresses that are far from one another, making it harder for the system to collect all the data needed inside the cache hierarchy, or that the application operates on a significant amount of data, the size of which exceeds the caches capacity by a lot.

The goal of our methodology is first of all to accurately extract the above-mentioned metrics for every algorithm under examination, and then to provide a combination of these metrics that is able to produce exact classes of functions with similar behavior when offloaded to PIM.

The methodology we developed can be broken down to three major steps. The first step is the profiling of each application via the extraction of its key metrics. This is done by feeding the source code of the function accompanied with the corresponding workload to the DAMOV simulator and filtering the resulting log files through our python facilitation scripts. The second step is the independent analysis of the AI and MPKI metrics in order to conclude on whether they can serve as sufficient indicators. As this step suggests, we propose a two-step workload classification method which relies on dividing the application firstly by their MPKI metric and then by their AI.

Although classification techniques have also been suggested in the past, we find our two-step approach to provide a sufficient trade-off between accuracy and simplicity. Its simplicity can be a key factor for a future transcription of this work as a methodology that can be executed by a compiler, further alleviating any programmer's involvement. In our evaluation we make a considerate selection of illustrative algorithms for every classification category in order to allow for the generalization of the results

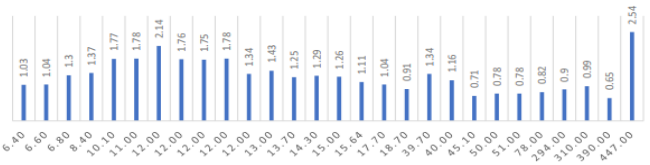

Fig. 1. Speedup over AI.

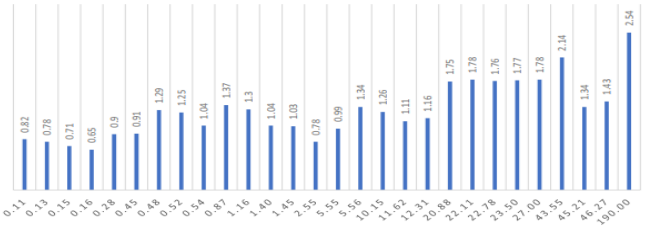

Fig. 2. Speedup over MPKI.

IV. Experimental Results

All experiments presented below have been performed on a suite of 28 representative applications, which are listed later on in Figure 4.

A. Arithmetic Intensity Analysis

In order to conclude on whether the metric of Arithmetic Intensity is a suitable one, we extract its value for the 28 applications (Figure 1). We observe that all functions subjected to a slowdown are concentrated in the area between 45.10 and 390. There are two exceptions in opposite directions. The Bezier Kernel calculation performs better on the host CPU, but is not included in the aforementioned area. The Padding holds the highest speedup calculated at $\times 2.54$, also holds the highest AI value (447). That being said, an AI value of 40 segregates the functions into two groups. The first group that consists of all applications with AI less than 40 provide an average speedup of 1.41. The second group that holds applications with AI greater than 40 provides an average slowdown of 0.81. In conclusion, the AI value of 40 manages to classify the apps with a 92% efficiency that could be improved by including AI as part of our two-step categorization method.

B. MPKI analysis

After simulating the same 28 applications we extract the MPKI values (Figure 2). Related studies have attempted to verify an MPKI threshold value of 10. The output verifies that all applications with MPKI greater than 10 could benefit from PIM. The average speedup these applications deliver is $\times 1.76$. Nevertheless, the metric is incapable of showing the existence

of functions that can benefit from the NDP even though they have a smaller MPKI result. We observe a random speedup pattern for the applications that resulted in an MPKI value less than 10. Using the knowledge of the underlying algorithm, we notice that many of these applications perform only trivial algebraic calculations (e.g., triple matrix multiplication). These observations lead us to assume that MPKI can be coupled with AI to provide a more precise characterization.

C. Applications Classification

Taking the above metric analysis into account, we observe that a sequential deployment of both could produce an accurate technique to determine the NDP suitability of a workload. Exploiting the fact that MPKI is accurate for applications that reside above the threshold value, we firstly divide the applications into two groups. Group A consists of all the functions with MPKI greater than 10 as mentioned before. We then apply the AI metric filtering to group B which contains the applications that the MPKI failed to accurately address. AI succeeds to address the potential performance improvements dividing the application into group B and group C. In group B we classify the functions that produce MPKI $<$ 10 but AI $<$ 40. Their average speedup value is $\times$1.20, less than the average value of group A, as expected. In group C we classify the applications with MPKI $<$ 10 and also AI $>$ 40, with average speed up of $\times$0.804. These compute-bound applicationss are better suited for host execution (Figure 3). Finally, we present our two-step classification method overview (Figure 4).

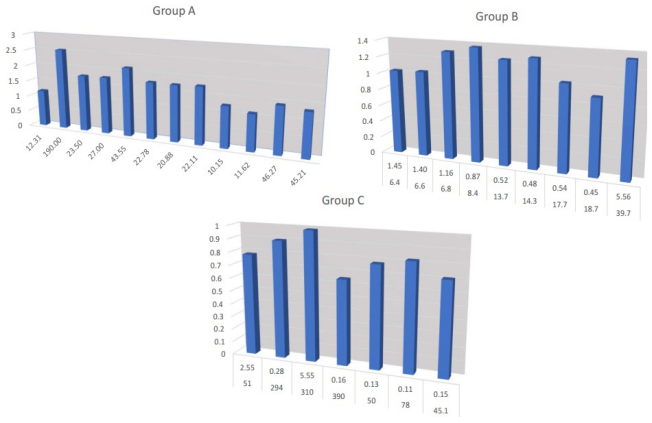

Fig. 3. Application Groups.

V. Conclusion

In this work we deducted a comprehensive review analysis of the most prevalent categories in the domain of NDP research. In order to support the unrestricted research about the adoption of PIM we narrowed down our focus to software simulation using the DAMOV simulator. For the facilitation of the programmer, we concentrated our efforts to function-offloading NDP approaches with no knowledge of the underlying algorithm. In this direction we presented a simplified, yet accurate two-step classification method that is based on two prevalent metrics, AI and MPKI, and successfully divides applications based on their NDP execution speedup.

We believe that the future of NDP exploration should highlight whether the generic approach of function-offloading will

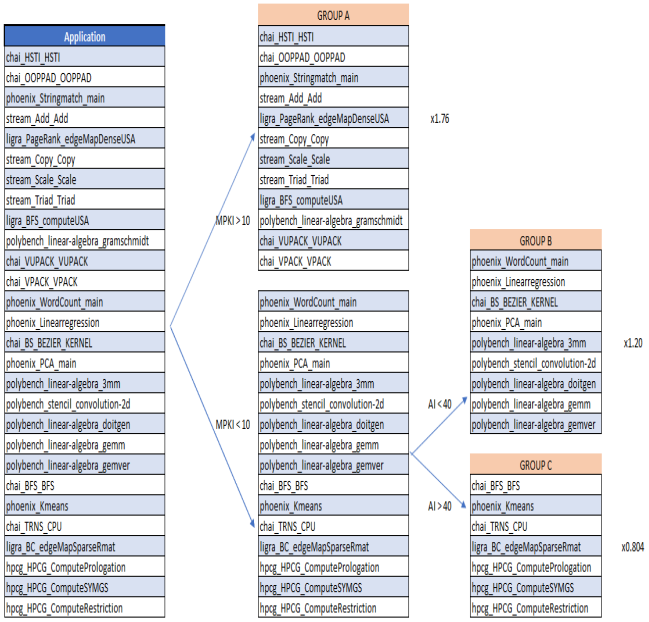

Fig. 4. Application Classification.

be beneficial enough to prevail against the faster but confined instruction-offloading. We also propose that the simplicity of this analysis could be a good starting point for the development of a real-time compiler-based offloading technique.

References

[1] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019.

[2] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, 2004, p. 162.

[3] D. Elliott, W. Snelgrove, and M. Stumm, "Computational ram: A memory-simd hybrid and its application to dsp," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, 1992, pp. 30.6.1–30.6.4.

[4] Memory cube consortium. "hybrid memory cube specification 2.1".

[5] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-memory: A workload-driven perspective," *IBM Journal of Research and Development*, vol. 63, no. 6, pp. 3:1–3:19, 2019.

[6] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, "Damov: A new methodology and benchmark suite for evaluating data movement bottlenecks," *IEEE Access*, vol. 9, pp. 134 457–134 502, 2021.

[7] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 185–197.

[8] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast bulk bitwise and and or in dram," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 127–131, 2015.

[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] 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.*, vol. 14, no. 4, 2017.

[11] 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, pp. 204–216.