PEPERONI: Pre-Estimating the Performance of Near-Memory Integration

Oliver Lenke
Technical University of Munich
o.lenke@tum.de

Richard Petri
Technical University of Munich

Thomas Wild
Technical University of Munich

Andreas Herkersdorf
Technical University of Munich

ABSTRACT

Near-memory integration strives to tackle the challenge of low data locality and power consumption originating from cross- chip data transfers, meanwhile referred to as locality wall. In order to keep costly engineering efforts bounded when transforming an existing non-near-memory architecture into a near-memory instance, reliable performance estimation during early design stages is needed. We propose PEPERONI, an agile performance estimation model to predict the runtime of representative benchmarks under near-memory acceleration on an MPSoC prototype. By relying solely on measurements of an existing baseline architecture, the method provides reliable estimations on the performance of near-memory processing units before their expensive implementation. The model is based on a quantitative description of memory boundedness and is independent of algorithmic knowledge, what facilitates its applicability to various applications.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**;
• **Hardware** → *Emerging tools and methodologies.*

KEYWORDS

near-memory computing, memory boundedness, performance-estimation, design space exploration, tile-based MPSoC

1 INTRODUCTION

Over the last decades, the increasing demand of compute performance could be trailed by simple power, frequency and threshold scaling methods in combination with Dennard scaling and numerous microarchitectural improvements. Since these methods have meanwhile reached a saturation effect [14], multi- and many-core solutions have been established, often implemented in tile-based architectures, in order to further support the continuous need of increased compute performance [20].

Modern many-core architectures come, however, with a potential lack of data-to-task locality and an increased need of data movement. Even the continuous improvement of cache architectures, which were the main hardware measure to speed up memory accesses, cannot follow this trend any longer [13].

The orthogonal approach of near- or even in-memory computing can be a meaningful solution to bridge the increasing gap between processing node and memory. Offloading dedicated tasks to specific compute units that are located near or even inside the memory modules often implies low-latency memory accesses and potential power savings due to much lower traffic on the interconnect network [4].

Due to several technological improvements, such as the invention of through-silicon-vias (TSV) [11] on the one hand, and the increasing memory wall on the other hand, the research on near-memory processing got a lot of momentum within the last decade.

However, the development effort for near-memory computing solutions adds additional design challenges, as the design of near memory components has to be well aligned to the intended workload as well as to the overall system architecture.

In order to invest costly development effort wisely, an easy but robust design space exploration (DSE) within early development stages is crucial. Our work proposes a method to estimate the performance gain of near-memory processing units up-front of the actual implementation. The approach analyzes and quantifies the memory boundedness of the system under a given workload in absence of any near-memory computing first. Based on that, we estimate the speedup potential in case a given subtask would be offloaded to a near-memory processing unit. Near-memory cores and hardware accelerators offer different performance characteristics and are therefore considered individually. Finally the performance estimates are evaluated in an FPGA based system running all 12 benchmarks of the IMSuite [5]. In detail, we

(1) propose the PEPERONI model, a novel method to quantify memory-boundedness on a tile-based System-on-Chip
(2) compare our analysis to the Roofline model [21]
(3) use our model to estimate performance gains due to near-memory integration
(4) distinguish between software-programmable near-memory cores and dedicated hardware accelerators
(5) verify our estimations with FPGA-based measurements.

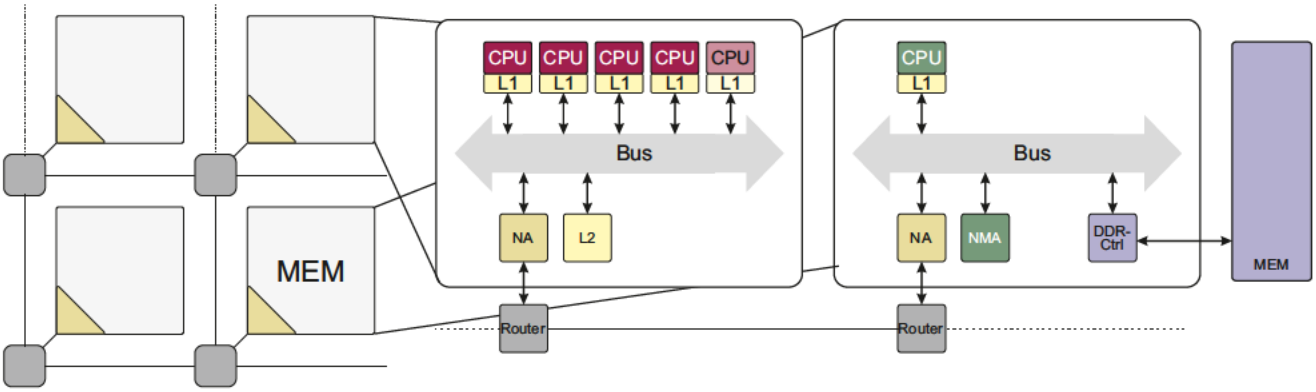

Figure 1: Schematic overview of the architecture: Left: 2x2 architecture with 3 compute-tiles and one memory-tile. Center: Internal structure of a compute-tile, including five Leon3-cores (one reserved for the operating system), a shared L2 cache and the network-adapter. Right: Internal structure of a memory-tile with off-chip DRAM module and optional near-memory cores or hardware accelerator, respectively

The rest of the paper is structured as follows. We summarize related approaches in Section 2 and give an overview on our system architecture in Section 3. In Section 4, we explore the system's behavior and it's boundedness, before we use these insights to calculate a speedup prediction in Section 5. In Section 6, we evaluate our estimation approach before we conclude in Section 7.

2 RELATED WORK

As the design and integration of near-memory computing is often a complex engineering task, several methods to estimate the potential gain before the actual implementation have already been explored. Besides several simulation-based techniques, also a few analytical models have been presented. Pimentel et al. [15] give a good overview on different design objectives. Specifically, conflicting targets as power vs. performance vs. cost are of high interest.

Beyond design characteristics, also different workload characteristics need to be analyzed. Reagen et al. [16] describe such an approach where existing C-Code of typical applications is used in high level synthesis of an accelerator. The results described there are reliable and come with low additional modeling effort. Different pareto-optimal design options were generated which helps to pre-select a potential solution before investing in actual design work. However, this approach neglects the influence of the system architecture, like the memory bandwith and latency, overheads for communication or offloading or coherency mechanisms between cores and accelerators. To determine reliable numbers for performance and power, it is furthermore desired to consider the overall architecture as a whole. Atlaf et al. [1] propose a performance model for hardware accelerators. They also consider the workload of the processor and the latency of data traffic. However, it focused only on standard accelerators while real near memory computing was not specifically considered. Also modeling the overall system utilization was neglected in this approach. Singh et al. [19], propose a novel approach based on machine learning techniques. It allows to simulate only a few algorithms and extrapolates further scenarios with reliable accuracy. This approach called NAPEL is able to predict performance and power numbers for near memory accelerated algorithms up to 220 times faster than conventional simulation based prediction methods.

Another analytical speedup prediction method was proposed by Rheindt et al. [17]. They derived a performance estimation from the proportional execution time of the desired workload. However, their approach still disregards architectural performance parameters such as the available memory bandwidth.

In contrast to previous works, our approach is an analytical work which incorporates the system characteristics and the current workload simultaneously and differs between near-memory hardware accelerators and programmable cores as well.

3 ARCHITECTURE

Our speedup estimation method is exemplarily demonstrated on an FPGA prototype architecture. We consider a tile-based architecture with 4 tiles, where a global off-chip DRAM is attached to one of the 4 tiles. A Network-on-Chip with four VCs connects each tile with its neighbouring tiles. Architectures like this can be seen as successors of traditional multi-core systems [20].

The bottom-right tile is connected to 1GB global memory. It is referred to as *memory-tile*, whereas all other tiles are identified as *compute-tiles*. Figure 1 shows an abstract view of the system as well as the internal structure of both a compute and a memory tile.

All tiles are equipped with five LEON3 cores, where one of these is reserved for the operating system in each tile. The cores feature private, write-through L1 caches with a size of 16kB for instructions and data each. Snooping-based cache coherence is ensured by an intra-tile AHB bus. It connects all CPUs with their shared L2 Cache and the network adapter. In the memory-tile, the CPUs are not used by default. Instead, all computations are distributed between the 3 compute-tiles only. In case the near-memory core is considered, one single CPU in the memory tile is enabled to act as near memory core. Near-memory hardware accelerators are considered to be implemented in the memory-tile and connected to the intra-tile AHB bus. We run our design in two configurations, either with one or with four active cores per compute-tile.

For evaluation, we run all 12 parallel benchmarks of the IMSuite [5], consisting of distributed parallel kernels, on our platform. We thereby execute and verify each benchmark sequentially.

Our prototype system uses an X10 runtime system following the PGAS programming paradigm what implies that each compute tile has it's private partition in the global memory. The runtime system schedules lightweight threads, so called *i-lets*, on remote tiles [8]. The use of object-oriented programming languages like X10 implies the necessity of transferring pointered data structures during inter-tile communication routines. To execute a so-called `at_statement`, all relevant data is combined to a graph and stored to the memory partition on the sender's side. Next, the receiving CPU is signalled to read the graph once and copy it to its own memory partition. After the copy operation has finished, the receiving CPU starts to process the received function. However, the graphcopy itself requires a sophisticated algorithm in order to adjust all pointers within the graph appropriately during the transfer. It is performed by a processor using a state-of-the-art software approach [9].

Offloading this graphcopy operation to near-memory processing unit is considered in this work. This can be either a dedicated accelerator or a near-memory core with an equivalent microarchitecture to all other cores in the system. We thus term the graphcopy operation as the *task-of-interest (TOI)* in the following sections.

The hardware accelerator traverses an arbitrary graph and creates a copy of it in the receiver's memory partition. Thereby, all pointers between objects are adjusted appropriately so that no references back to the original graph exist. It uses an additional *clone-map*, i.e. a previously allocated chunk of memory, to store the mapping of each object to it's copy. It's resource utilization is similar to one Leon3 Core. Details of the hardware accelerator can can be found in [18]. We do not claim the design and implementation of the hardware accelerator as a contribution in this work.

The prototype architecture is synthesized onto a Xilinx Virtex-7 FPGA which runs at 50 MHz. The memory controller and the DDR itself run at 100 MHz.

4 QUANTIFYING MEMORY BOUNDEDNESS

In order to accurately quantify the performance gain due to near-memory integration, a careful architecture analysis of the existing system in advance can provide meaningful insights. The question, whether and how much the system is memory bound, will highly affect the potential performance improvements, i.e. a highly memory bound scenario will profit more from near-memory integration than a compute bound one. Similar to [10] and [6], we determine memory boundedness by analyzing the run-time behavior of the application. This section describes the PEPERONI model, which leads to a quantitative assessment of memory boundedness and compares it with the well-known Roofline model.

Roofline Model. The classic Roofline model [21] visualizes the operational intensity (in Ops/Byte) on the horizontal axis and the compute performance (in Ops/s) on the vertical axis. The Roofline model displays the nominal peak values in two lines, also referred to as *roofs*. The horizontal line shows the nominal compute performance, while the diagonal roof results from the maximum memory bandwidth. The operational intensity of a benchmark determines a value on the horizontal axis. A system is called memory bound

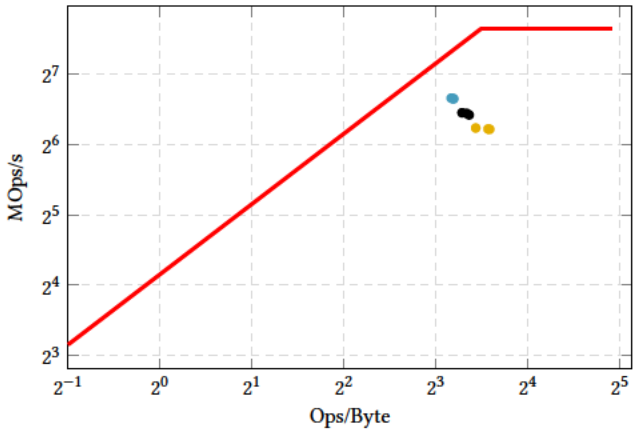

Figure 2: Tile-individual Roofline Model of the DR Benchmark. Red: Performance boundaries of a compute-tile. Black: Average Operating points on all 3 compute tiles. Blue: Average Operating points during the TOI only. Orange: Average Operating points of the TOI's complement.

when the vertical extension of that point crosses the diagonal line, otherwise it's called compute bound.

In contrast to the original proposal, many authors extended the Roofline model by an operating point based on both the real operational intensity in Ops/Byte and the real compute performance in Ops/s ([2], [3], [7], [12]).

In Figure 2, we applied this extension of the Roofline model to our prototype architecture. We illustrate the characteristics of each tile individually while running the system with four application cores per tile. The figure shows exemplarily one of the 12 IMSuite benchmarks performing the Dijkstra Routing algorithm (DR).

Moreover, we performed a subtask measurement covering solely the TOI in order to compare the performance of the TOI with the overall performance on all thee compute-tiles individually. We plot 3 different sets of operating points, one for the total benchmarks (black), one for the TOI only (blue) and one for the remaining fraction of the benchmark, i.e. the complement of the TOI (orange). The plot illustrates the operating points of all three compute-tiles in one diagram. All tiles are designed equally and thus refer to the same roofs, only the operating points differ slightly between the three compute-tiles.

The model serves well for a qualitative comparison of the TOI and the complete application. Whereas the TOI is clearly memory bound, its complement is compute bound on two of three tiles. This boundedness analysis confirmed that improving memory accessibility for the TOI, e.g. by near memory acceleration, will foreseeably allow the TOI complement and, thus, the entire application to scale further in compute performance than without TOI acceleration.

PEPERONI Model. Even with the introduction of TOI the answer from the Roofline model between compute or memory bound is - justified with its simplicity - still a binary decision. However, although evidently many systems can be considered as highly memory- or compute bound, also scenarios limited roughly equally by compute performance and memory bandwith are conceivable. A system performing compute- and memory-limited phases alternately or even

in parallel would be a good example. To weight this more precisely, we propose a quantification method which determines compute- and memory boundedness in a range between 0 % and 100 %.

In order to analyze this, the system can be specified by its maximum peak values, similar to the Roofline model. Again, we apply the model to each tile individually.

$$\text{Maximum compute performance} : CP_{max,tile} \text{ in } [Ops/s] \quad (1)$$

$$\text{Maximum memory bandwidth} : BW_{max,tile} \text{ in } [Byte/s] \quad (2)$$

In addition to the theoretical performance maxima, we measure the real behavior of the system averaged over the complete execution time. We use the same measurements as for the Roofline model leading to the following parameters:

$$\text{Average compute performance} : CP_{avg,tile} \text{ in } [Ops/s] \quad (3)$$

$$\text{Average memory bandwidth} : BW_{avg,tile} \text{ in } [Byte/s] \quad (4)$$

Note that $CP_{max,tile}$ and $BW_{max,tile}$ denote the maximum peak values, whereas $CP_{avg,tile}$ and $BW_{avg,tile}$ express the real operating point which is strictly below the maximum and has been determined during run-time. We consider both $BW_{avg,tile}$ and $BW_{max,tile}$ after the last-level cache within each compute-tile. However, in other scenarios an observation on the bandwidth before the cache might be meaningful as well, especially if no near-memory integration is considered, but e.g. cache optimizations.

The common understanding of memory boundedness says that *the execution time is primarily limited by the available memory bandwidth*. This implies often that the bandwidth is heavily utilized in the given scenario. Similarly, a compute bound system would be mostly limited by a heavily exploited compute performance.

In order to quantify this, we calculate *compute and memory boundedness* as follows:

$$CB_{tile} = \frac{CP_{avg,tile}}{CP_{max,tile}}; \quad MB_{tile} = \frac{BW_{avg,tile}}{BW_{max,tile}} \quad (5)$$

The fraction of used compute performance and memory bandwith leads to a good estimation how close the system is operated with respect to the corresponding limits. In addition to this, we define *relative compute and memory boundedness*, a metric quantifying whether memory or compute limitations are more restricting.

$$CB_{rel,tile} = \frac{CB_{tile}}{CB_{tile} + MB_{tile}}; MB_{rel,tile} = \frac{MB_{tile}}{CB_{tile} + MB_{tile}} \quad (6)$$

The metrics $CB_{rel,tile}$ and $MB_{rel,tile}$ add together to 100 % in any scenario. If one would compare these values to the Roofline model, each operating point with $MB_{rel,tile} > 50$ % would be classified as Memory bound by the Roofline model, i.e. the vertical extrapolation would hit the diagonal roof first. Thus, our model does not contradict the Roofline model.

This method can be seen as an additional approach to the Roofline model in order to quantify how much a system is bound by its compute performance and its memory bandwidth. This much more granular and quantitative description of compute and memory boundedness can now be used to estimate the speedup of near memory computing before going through an expensive design cycle. Chapter 5 demonstrates such a novel estimation for a near memory core as well as a dedicated hardware accelerator.

5 SPEEDUP ESTIMATION

Near-Memory Cores. The previously described analysis of the hardware architecture and its utilization can be used to quantify the attainable speedup when offloading the TOI to a near-memory processing unit. This estimation relies solely on measurements of the baseline architecture, i.e. before the near-memory integration has started.

We apply that Speedup estimation individually to each tile of our architecture. In this section, it is assumed to offload the TOI to a software programmable core, whereas in the following section a dedicated hardware accelerator is considered. Note that we consider only near-memory cores with the same microarchitecture as the host processor in this work. In our understanding, near-memory cores are, other than in-memory-cores, still on-chip and not integrated into the off-chip DRAM die. Thus, they run on the same speed as the other processors. The major advantage of such a near-memory core (NMC) is its typically much higher memory bandwith compared to an ordinary system core.

$$\text{Maximum memory bandwidth NMC} : BW_{nmcore} \text{ in } [Byte/s] \quad (7)$$

The increase in memory bandwith can be described formally with

$$BW_{nmcore} > BW_{max,tile} \quad (8)$$

In other words, we can assume the near-memory core to perform memory accesses faster. The Speedup factor of performing fixed-length memory accesses (i.e. one cacheline) can be appraised by

$$S_{mem} = \frac{BW_{nmcore}}{BW_{max,tile}} \quad (9)$$

Note that in our definition, the speedup is defined relative to a base value of 1, i.e. a speedup of 1 would imply no speedup. However, the whole TOI is certainly not accelerated by S_{mem}, especially if the system is only slightly memory bound.

In order to estimate the acceleration of the total application, we divide the total runtime into two disjunct parts, the TOI and its complement. This can be expressed formally by:

$$t_{app} = t_{app} \cdot f_{toi} + t_{app} \cdot (1 - f_{toi}) = t_{toi} + t_{comp} \quad (10)$$

Hereby we consider f_{toi} as the fraction of processing time that is spent to execute the TOI. We can further divide the TOI into a memory bound part and a compute bound one:

$$t_{toi} = t_{toi} \cdot MB_{rel,toi} + t_{toi} \cdot CB_{rel,toi} \quad (11)$$

Depending on the memory boundedness of the given task, it might benefit more or less from the increase in memory bandwidth. Assuming that the characteristics of the TOI are comparable on all compute tiles, we use average $CB_{rel,toi}$ and $MB_{rel,toi}$ over all tiles.

Offloading the TOI to the previously described near-memory core will certainly have a negligible influence on the execution-time of the complement t_{comp}. Also the compute bound fraction of the TOI will not be influenced severely, whereas the memory bound part might be accelerated by S_{mem}. Thus, our prediction of the application runtime under near-memory core acceleration calculates as follows:

$$t'_{app,nmc} = t_{comp} + t_{toi} \cdot CB_{rel,toi} + \frac{t_{toi} \cdot MB_{rel,toi}}{S_{mem}} \quad (12)$$

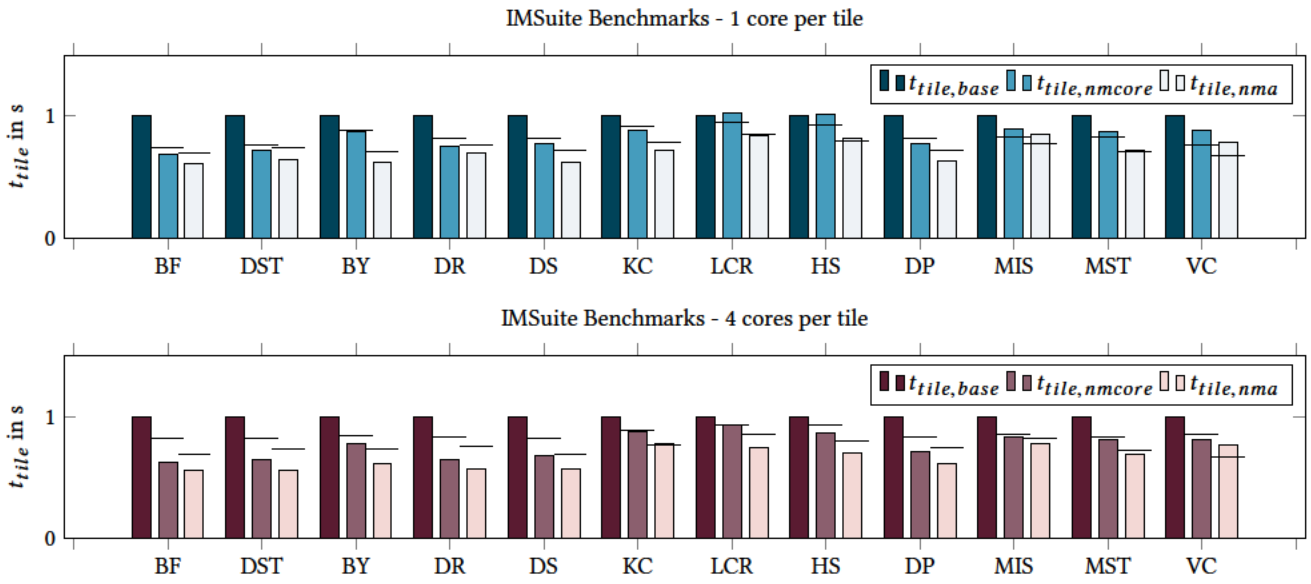

Figure 3: Execution time of the IMSuite benchmarks without near-memory acceleration ($t_{tile,base}$), and with near-memory core ($t_{tile,nmcore}$) or near-memory hardware accelerator ($t_{tile,nma}$) enabled. The horizontal lines illustrate the PEPERONI estimation based on $t_{tile,base}$. All values are normalized to $t_{tile,base}$.

Near-Memory Hardware Accelerators. In contrast to a near-memory programmable core, dedicated hardware accelerators offer additional improvements due to their task-specific hardware implementation, what implies a higher nominal compute-performance for specific algorithms than a CPU. This comes from a possibly extraordinary high level of parallelism within hardware circuits.

Also the memory access pattern of hardware accelerators can be different. While CPUs typically load or store full cachelines, a memory access of the hardware accelerator is not necessarily bound to that fixed length in the general case. The transfer time per memory access of arbitrary length can be calculated by

$$t_{transfer} = t_{arb} + k \cdot t_{word} \tag{13}$$

while k denotes the number of transmitted words within one access. However, determining the specific memory access pattern is rather challenging in early design stages of the accelerator. In case the designer has detailed knowledge on the access pattern yet, this information should be used evidently. Otherwise, it is proposed to appraise it by the CPU access pattern allowing for possible inaccuracies. In order to not loose generality, we use this simplification in our evaluation as well. With that, the total time of memory transfers can be appraised by summing up all single memory accesses:

$$t_{mem} = \sum_{i=1}^{n} t_{transfer,i} \tag{14}$$

Depending on the implementation of the hardware accelerator, the memory transfer time highly dominates its total operating time. We thus consider only the memory transfer time in order to estimate the runtime of the near-memory hardware accelerator. Our estimation for the total application runtime under near-memory hardware acceleration calculates then as follows:

$$t'_{app,nma} = t_{comp} + t_{mem} \tag{15}$$

This assumes that all arithmetic operations within the near-memory accelerator are masked by simultaneous memory accesses and thus not extend the execution time. While this would be impossible in conventional CPUs, it is a meaningful approximation for hardware accelerators, even if they perform non-trivial algorithms.

6 EVALUATION

In order to verify the accuracy of our PEPERONI speedup estimation method described in Section 5, all 12 benchmarks out of the IMSuite [5] have been executed on the system prototype in different variants. The execution without any near-memory acceleration serves as baseline. Compared to that, an existing near-memory core with the same parameters as the host processor, and a graphcopy-specific hardware accelerator, have been employed to perform the graphcopy operation. The actual behavior under near-memory acceleration is thereby compared to the estimated performance. During the baseline execution, all relevant performance metrics have been measured: The CPU time, the number of operations and the memory Byte. They have been determined once only within the TOI and once during the *Region of interest* of the benchmark. We always consider the mean value of five identical runs in order to get rid of statistical deviations. Based on this, all metrics discussed in Section 4 and Section 5 can be calculated.

Figure 3 compares the measured execution time of the total benchmarks with and without near-memory processing and compares them with the PEPERONI estimations. All values are normalized to the baseline variant. In the 1-core variant, we predict the total

execution time with a root-mean-square-error of 0.07 for the near-memory core and 0.11 for the hardware accelerator, respectively. Predicting near-memory hardware accelerators is more difficult, since two independent aspects - the hardware implementation and the near-memory integration - are estimated simultaneously.

The estimation of the 4-core variant comes with a slightly increased error, since several uncertainties - such as the parallel and sequential portion of the benchmark and whether the TOI resides in the sequential part or not - have to be faced. Despite of these uncertainties, PEPERONI leads to a root-mean-square-error of 0.17 and 0.19 for the near-memory core and the accelerator, respectively.

Although the accuracy of our estimations is reliable, a moderate prediction error is present at least in some cases. Especially for the BF, DST, DR and DS benchmarks, PEPERONI under-estimates the performance of the near-memory accelerator. This can be explained with an additional speedup due to secondary effects, i.e. the acceleration of the TOI enables the system to enter the parallel part of the application faster and to use the resources in a more balanced way. Considering that PEPERONI does not require any algorithmic knowledge, these results give a good quantitative overview on the potential speedup, if a certain task would be offloaded to near-memory computing units.

7 CONCLUSION

We presented PEPERONI, a model to estimate the speedup potential of near-memory integration. It is based on a run-time analysis of an existing baseline architecture, i.e. before integrating near-memory computing units. This model is used to approximate the memory boundedness of the system quantitatively. Based on that, we estimate the total application runtime under near-memory acceleration and distinguish thereby between software-programmable cores and hardware accelerators. The performance estimation was performed on 12 different benchmarks in two platform configurations with a root-mean square-error of 0.07 and 0.17 for a near-memory core and 0.11 and 0.19 for near-memory hardware accelerators, respectively.

All in all, we envision the PEPERONI method to ease the decision whether near-memory processing brings a satisfying performance improvement. Adapting the model to various other architectural modifications is considered as future work.

ACKNOWLEDGMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)-Project Number 146371743-TRR 89: Invasive Computing. We thank Sven Rheindt, Akshay Srivatsa and Andreas Fried for the valuable discussions and meaningful inputs as well as the reviewers for their helpful feedback.

REFERENCES

[1] M. Shoaib Bin Altaf and . A. Wood. 2017. LogCA: A High-Level Performance Model for Hardware Accelerators. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. ACM, 375–388. https://doi.org/10.1145/3079856.3080216

[2] V. Caparrós Cabezas and M. Püschel. 2014. Extending the roofline model: Bottleneck analysis with microarchitectural constraints. In *2014 IEEE International Symposium on Workload Characterization, IISWC 2014, Raleigh, NC, USA, October 26-28, 2014*. IEEE Computer Society, 222–231. https://doi.org/10.1109/IISWC.2014.6983061

[3] B. da Silva, A. Braeken, E. H. D'Hollander, and A. Touhafi. 2013. Performance Modeling for FPGAs: Extending the Roofline Model with High-Level Synthesis Tools. *Int. J. Reconfigurable Comput.* 2013 (2013), 428078:1–428078:10. https://doi.org/10.1155/2013/428078

[4] A. Farmahini Farahani, J. Ho Ahn, K. Morrow, and N. Sung Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*. IEEE Computer Society, 283–295. https://doi.org/10.1109/HPCA.2015.7056040

[5] S. Gupta and V. K. Nandivada. 2015. IMSuite: A benchmark suite for simulating distributed algorithms. *J. Parallel Distributed Comput.* 75 (2015), 1–19. https://doi.org/10.1016/j.jpdc.2014.10.010

[6] A. Hutcheson and V. Natoli. 2011. Memory Bound vs . Compute Bound : A Quantitative Study of Cache and Memory Bandwidth in High Performance Applications.

[7] A. Ilic, F. Pratas, and L. Sousa. 2014. Cache-aware Roofline model: Upgrading the loft. *IEEE Comput. Archit. Lett.* 13, 1 (2014), 21–24. https://doi.org/10.1109/L-CA.2013.6

[8] M. Mohr, S. Buchwald, A. Zwinkau, C. Erhardt, B. Oechslein, J. Schedel, and D. Lohmann. 2015. Cutting out the middleman: OS-level support for x10 activities. In *Proceedings of the ACM SIGPLAN Workshop on X10, Portland, OR, USA, June 15 - 17, 2015*, Olivier Tardieu and José Nelson Amaral (Eds.). ACM, 13–18. https://doi.org/10.1145/2771774.2771775

[9] M. Mohr and C. Tradowsky. 2017. Pegasus: Efficient data transfers for PGAS languages on non-cache-coherent many-cores. (2017), 1781–1786. https://doi.org/10.23919/DATE.2017.7927281

[10] D. Molka, R. Schöne, D. Hackenberg, and W. E. Nagel. 2017. Detecting Memory-Boundedness with Hardware Performance Counters. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*, Walter Binder, Vittorio Cortellessa, Anne Koziolek, Evgenia Smirni, and Meikel Poess (Eds.). ACM, 27–38. https://doi.org/10.1145/3030207.3030223

[11] M. Motoyoshi. 2009. Through-Silicon Via (TSV). *Proc. IEEE* 97, 1 (2009), 43–48. https://doi.org/10.1109/JPROC.2008.2007462

[12] G. Ofenbeck, R. Steinmann, V. Caparrós Cabezas, D. G. Spampinato, and M. Püschel. 2014. Applying the roofline model. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, Monterey, CA, USA, March 23-25, 2014*. IEEE Computer Society, 76–85. https://doi.org/10.1109/ISPASS.2014.6844463

[13] P. Kogge. 2017. *Memory Intensive Computing, the 3rdWall, and the Need for Innovation in Architecture*. Univ. of Notre Dame. https://memsys.io/wp-content/uploads/2017/12/The_Wall.pdf

[14] J. Parkhurst, J. A. Darringer, and B. Grundmann. 2006. From single core to multi-core: preparing for a new exponential. In *2006 International Conference on Computer-Aided Design, ICCAD 2006, San Jose, CA, USA, November 5-9, 2006*, Soha Hassoun (Ed.). ACM, 67–72. https://doi.org/10.1145/1233501.1233516

[15] A. D. Pimentel. 2017. Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration. *IEEE Des. Test* 34, 1 (2017), 77–90. https://doi.org/10.1109/MDAT.2016.2626445

[16] B. Reagen, Y. S. Shao, G.-Y. Wei, and D. M. Brooks. 2013. Quantifying acceleration: Power/performance trade-offs of application kernels in hardware. In *International Symposium on Low Power Electronics and Design (ISLPED), Beijing, China, September 4-6, 2013*, Pai H. Chou, Ru Huang, Yuan Xie, and Tanay Karnik (Eds.). IEEE, 395–400. https://doi.org/10.1109/ISLPED.2013.6629329

[17] S. Rheindt, A. Fried, O. Lenke, L. Nolte, T. Sabirov, T. Twardzik, T. Wild, and A. Herkersdorf. 2020. X-CEL: A Method to Estimate Near-Memory Acceleration Potential in Tile-Based MPSoCs. In *Architecture of Computing Systems - ARCS 2020 - 33rd International Conference, Aachen, Germany, May 25-28, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12155)*, André Brinkmann, Wolfgang Karl, Stefan Lankes, Sven Tomforde, Thilo Pionteck, and Carsten Trinitis (Eds.). Springer, 109–123. https://doi.org/10.1007/978-3-030-52794-5_9

[18] S. Rheindt, A. Fried, O. Lenke, L. Nolte, T. Wild, and A. Herkersdorf. 2019. NEMESYS: near-memory graph copy enhanced system-software. In *Proceedings of the International Symposium on Memory Systems, MEMSYS 2019, Washington, DC, USA, September 30 - October 03, 2019*. ACM, 3–18. https://doi.org/10.1145/3357526.3357545

[19] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal. 2019. NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 27. https://doi.org/10.1145/3316781.3317867

[20] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. 2007. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro* 27, 5 (2007), 15–31. https://doi.org/10.1109/MM.2007.89

[21] S. Williams, A. Waterman, and D. A. Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76. https://doi.org/10.1145/1498765.1498785