Future Generation Computer Systems 107 (2020) 257–273

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Application-driven Cache-Aware Roofline Model

Diogo Marques [a,*], Aleksandar Ilic [a], Zakhar A. Matveev [b], Leonel Sousa [a]

[a] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[b] Intel Corporation, United States of America

ARTICLE INFO

Article history:
Received 9 April 2019
Received in revised form 2 December 2019
Accepted 25 January 2020
Available online 3 February 2020

Keywords:
Cache-Aware Roofline Model
Exascale applications
Performance modeling
Application characterization

ABSTRACT

In the coming exascale era, the complexity of modern applications and hardware resources imposes significant challenges for boosting the efficiency via execution fine-tuning. To abstract this complexity in an intuitive way, recent application analysis tools rely on insightful modeling, e.g., Intel® Advisor with Cache-aware Roofline Model. However, these approaches mainly consider the maximum architecture capabilities, which may limit their usability when characterizing real-world applications. To address this issue, a novel Cache-Aware Roofline Model for more accurate performance modeling of multi-cores is proposed, which realistically resembles application requirements. The proposed fine-grain modeling relies on micro-benchmarking to decouple the attainable performance of the micro-architecture for different utilization scenarios and for a diverse set of functional units and memory levels. Memory sub-system traffic simulation, dynamic and static analyses are also used to derive the requirements of the applications. Experimental results for a real multi-core system with an Intel server processor and for a set of 13 kernels from exascale proxy applications, show that the proposed models provide more accurate application characterization, optimization hints and bottleneck detection in comparison to the state-of-the-art models.

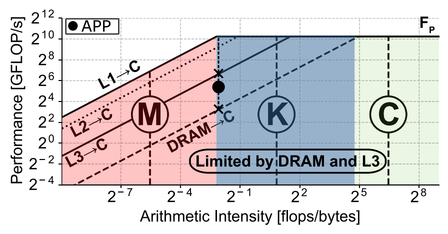
1. Introduction

The growing performance demands of modern applications lead to the development of supercomputers with thousands of cores, capable of executing trillions of floating-point operations per second (Tflops/s) and accessing fast and larger memories. While current systems can achieve a maximum performance of 143 500 Tflops/s [1], the first exascale system, *i.e.*, a machine able to perform more than quintillion operations per second, is expected to be fully operational at the end of 2021 [2], marking the beginning of the exascale era. Due to their huge computational capabilities, exascale systems are expected to provide several breakthroughs in the most diverse scientific areas. To attain such high performance, these systems have been the subject of continuous micro-architecture and technological improvements. These enhancements intrinsically augment the hardware complexity, *e.g.*, a higher number of cores with advanced functionalities, a more complex memory hierarchy, interconnects and integration of diverse emergent technologies [3–5]. For this reason, relating application characteristics with the capabilities of underlying hardware resources has become far from a trivial task.

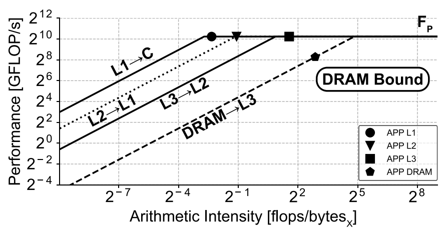
In this scenario, correlating application behavior with the upper-bound capabilities of different components in a highly parallel architecture is essential to identify the bottlenecks preventing applications from attaining maximum system performance. With this aim, simulation tools and architecture-specific runtime methods based on hardware counters were proposed [6,7], which rely on an extensive evaluation to provide an in-depth characterization of architecture/application interaction and capabilities. However, these methods may be neither user-friendly nor practical, thus only experienced users can extract useful information from them when detecting application bottlenecks. In contrast, simple and insightful models, such as Cache-Aware Roofline Model (CARM) [8], provide the means to quickly assess and relate the architecture characteristics with the application ability to exploit them, allowing software developers to fully focus on application tuning and enabling applications to reach the maximum performance. CARM relates application characteristics and system upper-bounds for performance, power consumption and energy-efficiency [8,9]. It includes the entire memory hierarchy (*i.e.* caches and Dynamic Random Access Memory (DRAM)) in a single plot, providing a simple and insightful evaluation of how different hardware resources influence application execution. For these reasons, CARM is officially part of Intel® Advisor (included in the Intel's main application development framework) [10]. Current roofline modeling approaches usually only consider the absolute maximums of the micro-architectures. The implementation of CARM in Intel® Advisor 2019 (Update 3), referred herein

* Corresponding author.
E-mail addresses: diogo.marques@inesc-id.pt (D. Marques), aleksandar.ilic@inesc-id.pt (A. Ilic), zakhar.a.matveev@intel.com (Z.A. Matveev), leonel.sousa@inesc-id.pt (L. Sousa).

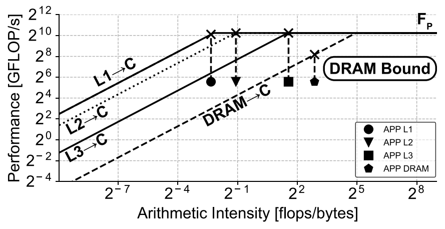
https://doi.org/10.1016/j.future.2020.01.044
0167-739X/© 2020 Elsevier B.V. All rights reserved.



(a) Cache-Aware Roofline Model.



(b) Hierarchical ORM.



(c) Integrated Roofline Model.

Fig. 1. State-of-the-art roofline modeling approaches.

as Intel® Advisor CARM 19.3, follows this same idea. However, this limits the capability of CARM to accurately characterize real-world applications that contain different amounts of load and store operations and may use diverse instruction set architecture (ISA) extensions. This is a gap that the work proposed in this paper intends to close.

CARM is already used to aid architecture design [11], and for optimization and characterization of applications from different domains, such as physics [12,13], quantum chemistry [14], medical image processing [15], seismic modeling [16] and scientific computation [17]. However, those works do not consider application heterogeneity, which may result in a characterization with reduced accuracy. For example, a real-world application may contain several execution phases (kernels) that rely on different instructions and hardware components, e.g., functional units or memory levels. As it is shown herein, characterizing these kernels within the model that only considers the absolute maximum performance of the micro-architecture, such as the Intel® Advisor CARM 19.3, may provide misleading characterization and optimization hints. In contrast, this work proposes an application-driven CARM methodology, herein referred as adCARM, which explicitly considers the relevant application specifics and demands to provide more precise insights and accurate characterization for real-world applications. The proposed methodology closes the gap between the state-of-the-art (SoA) CARM and the characteristics of applications that use diverse capabilities of the micro-architectures. Furthermore, since the proposed adCARM inherits the same interpretation methodology from SoA CARM, it maintains the simplicity and insightfulness for characterizing applications.

For this purpose, the work proposes a set of CARM-based models that encapsulate micro-architecture upper-bounds for the application-specific requirements and it includes the following set of contributions:

– extensive evaluation of a processor, the Intel® Xeon® 6140 Gold (code-named Skylake), to experimentally assess peak floating-point (FP) performance and sustainable bandwidth of different memory levels for different instruction set extensions and load/store ratios;
– a set of CARM-based metrics and models to improve bottleneck detection and derived optimization insights;
– characterization of a set of exascale proxy applications to demonstrate the insightfulness and usability of the proposed models.

To achieve these objectives, a fine-grain micro-benchmarking evaluation is performed to obtain an accurate characterization of achievable micro-architecture upper-bounds. The experimental results reveal that the adCARM provides a more precise characterization of modern applications when compared to the existing roofline modeling methodologies. Moreover, the proposed models are capable of accurately pinpointing the main application execution bottlenecks in different parts of the micro-architecture. This provides the means to select the best optimization techniques to improve application performance on modern computational systems.

2. Background: Cache-aware Roofline Model

CARM characterizes performance upper-bounds for a given architecture, with respect to the arithmetic intensity (AI), *i.e.*, the amount of performed computations over the total amount of requested data (*bytes*) [8]. By considering that memory operations and computations can be simultaneously executed in modern out-of-order processors, the overall execution is limited either by the time to perform computations or by the time to serve memory requests. Hence, CARM contains three distinct regions: memory bound region (slanted roof), compute bound region (horizontal roof) and a “mixed” region, where applications can be both memory and/or compute bound. For each memory level, the slanted roof intersects the horizontal roof at a single point, *i.e.*, the ridge point [8,9].

As it can be observed in Fig. 1a, CARM memory region includes all memory levels (L1, L2, L3 and DRAM) in a single plot, each limited by its corresponding slanted roof. The maximum attainable performance in this region is limited by the L1 cache bandwidth, while the remaining levels offer a lower attainable performance, due to the sustainable bandwidth reduction when data is fetched further away from the core. The right part of the model, delimited by the maximum FP performance (F_p), represents the compute bound region.

When characterizing the application behavior, CARM decouples the bottlenecks limiting the application execution, allowing to select suitable optimization techniques to be applied. For example, if an application is at the left side of the ridge point (kernel

Table 1
Tested system specifics — Intel® Xeon® 6140 Gold processor (Single-Socket).

#Cores		18
Nominal frequency [GHz]		2.3
Theoretical F_p, AVX512 DP FMA [GFLOP/s]		1324.8
L1	Size [bytes] ORM bandwidth [GB/s]	32k 7948.8
L2	Size [bytes] ORM bandwidth [GB/s]	1 MB 2649.6
L3	Size [bytes] ORM bandwidth [GB/s]	1.375M/Core 662.4
DRAM	Size [bytes] #Channels (8 bytes/channel) Frequency [MHz] ORM bandwidth [GB/s]	32G 2 2666 42.66

Table 2
Intel model specific registers used for micro-architecture benchmarking.

FP AVX512	DP SP	FP_ARITH_INST_RETIRED_512B_PACKED_DOUBLE FP_ARITH_INST_RETIRED_512B_PACKED_SINGLE
FP AVX	DP SP	FP_ARITH_INST_RETIRED_256B_PACKED_DOUBLE FP_ARITH_INST_RETIRED_256B_PACKED_SINGLE
FP SSE	DP SP	FP_ARITH_INST_RETIRED_128B_PACKED_DOUBLE FP_ARITH_INST_RETIRED_128B_PACKED_SINGLE
FP scalar	DP SP	FP_ARITH_INST_RETIRED_SCALAR_DOUBLE FP_ARITH_INST_RETIRED_SCALAR_SINGLE
Loads	MEM_INST_RETIRED_ALL_LOADS	
Stores	MEM_INST_RETIRED_ALL_STORES	

"M" in Fig. 1a), its execution is limited by memory accesses and can be improved by applying memory-related optimizations. On the other hand, an application positioned at the right side of the ridge point (kernel "C" in Fig. 1a) is limited by arithmetic operations and its execution can be improved by code vectorization or parallelization. Finally, an application placed in the "mixed" region (kernel "K" in Fig. 1a) may be limited by computations and/or memory transfers, depending on their instruction mix and on the memory levels exercised by the application.

Moreover, by plotting a vertical line at the AI of the application, as shown in Fig. 1a, it is possible to uncover the main sources of performance degradation. The intersections of this vertical line and the CARM roofs represent the potential execution bottlenecks that might limit application performance. The intersections right above and below the application point are identified as the main sources of performance degradation and should be the main target of optimization. In the example of Fig. 1a, the main bottlenecks of the application are accesses to L3 and DRAM memories (see black dot in Fig. 1a).

Besides CARM, the roofline modeling landscape includes two additional approaches: Original Roofline Model (ORM) [18] and its hierarchical variant [19], and Integrated Roofline Model (IRM) [20]. While the compute region is evaluated equally in all three models, their modeling of the upper-bounds in the memory subsystem differs. Hierarchical ORM (Fig. 1b) considers the bandwidth between memory levels, and its AI corresponds to the amount of performed computations over the amount of data requested by memory level "x" ($bytes_x$) [18]. Due to this property, a single application (kernel) is represented by "x" points in Hierarchical ORM, one for each memory level. Similar to CARM, the memory region of the Hierarchical ORM contains several roofs, each one representing a memory level. The main execution bottleneck corresponds to the minimum of the intersections between the AIs of the "x" points with their correspondent roofs (*e.g.*, DRAM bandwidth in Fig. 1b). Finally, IRM (Fig. 1c) aims at merging both CARM and ORM in a single method. This model uses the modeling approach of CARM in the memory subsystem *i.e.*, it considers the sustainable bandwidth seen from the core for each memory level, while adopting ORM methodology for application characterization and bottleneck detection (*e.g.*, in Fig. 1c, DRAM is the main execution bottleneck).

3. Micro-architecture benchmarking

The sustainable performance and memory bandwidth of x86 CPU architectures have greatly increased across different generations. This is mainly achieved with introduced micro-architectural enhancements, such as improving the execution capacity and including additional dispatch ports in the out-of-order engine [21]. This trend is followed by the most recent Intel core architecture, *i.e.*, Sunny Cove, that supports one additional port for store operations when compared to previous Intel CPU micro-architectures, containing 2 ports for loads and 2 ports dedicated to stores [22]. In particular for single-socket Intel® Xeon® Gold 6140 (Skylake Server architecture), the support for double-precision (DP) FP fused multiply–add (FMA) instructions from 512-Advanced Vector Extension (AVX512) at the nominal operating frequency, makes its theoretical performance equal to 1324.8 GFLOP/s@2.3 GHz (18 cores), *i.e.*, twice the performance offered by AVX instructions, which was the highest ISA supported in previous micro-architectures. This performance increase also occurs in the memory subsystem, whose theoretical L1 bandwidth for AVX512 in Intel® Xeon® Gold 6140 is equal to 7948.8 GB/s (18 cores@ 2.3 GHz), while for AVX instructions it is equal to 3974.4 GB/s.

Current multi-cores support a variety of instruction types and ISA extensions, enabling the existence of a vast range of potential factors that may impact the performance of different applications. Due to the micro-architecture complexity, a theoretical evaluation of its capabilities does not allow to pinpoint the main execution bottlenecks. As such, it is essential to firstly characterize and experimentally assess the sustainable upper-bound capabilities of the micro-architecture for different execution and utilization scenarios, most notably the upper-bounds of the memory subsystem and FP units for diverse instruction types and ISA extensions.

This paper focuses on fully assessing the potential of a computing system, as a case study with an eighteen-core Intel® Xeon® 6140 Gold processor. As shown in Table 1, the tested system contains three cache levels (L1, L2 and L3) and a DRAM: L1 (32 kB) and L2 (256 kB) caches are private to each core, while the accesses to L3 cache (1.375 MB/core) and DRAM (32 GB) are shared. At nominal frequency (2.3 GHz), the ORM bandwidth between memory levels is 7948.8 GB/s, 2649.6 GB/s, 662.4 GB/s and 42.66 GB/s, respectively for L1, L2, L3 and DRAM. In contrast to ORM, CARM considers the sustainable bandwidth for different memory levels, which can only be obtained through in-depth and precise micro-architecture benchmarking.

Therefore, a set of assembly micro-benchmarks was designed in order to assess the maximum performance of the system under several execution scenarios, *e.g.*, scalar memory accesses, Streaming Single Instruction, Multiple Data Extension (SSE) instructions, etc. These benchmarks have the structure of the Algorithms 1 and 2, respectively for ADD DP AVX512 instructions and SSE LD/ST ratio instructions. These algorithms only represent a subset of the benchmarks utilized in this evaluation, whose structure is easily adaptable to evaluate different utilization scenarios.[1]

The micro-benchmarks in Algorithms 1 and 2 contain two main loops. The outer loop ensures that the performed tests

[1] The micro-benchmarks are available upon request to the corresponding author.

Algorithm 1: FP Benchmark for DP AVX512 ADD Instructions.

```
for i = 0; i < time; i++{
    for j = 0; j< repeat; j++{
        vaddpd %zmm0, %zmm0, %zmm0
        vaddpd %zmm1, %zmm1, %zmm1
        vaddpd %zmm2, %zmm2, %zmm2
        vaddpd %zmm3, %zmm3, %zmm3
        (...)
    }
    vaddpd %zmm0, %zmm0, %zmm0
    vaddpd %zmm1, %zmm1, %zmm1
    vaddpd %zmm2, %zmm2, %zmm2
    vaddpd %zmm3, %zmm3, %zmm3
    (...)
}
```

Algorithm 2: LD/ST Ratio Memory Benchmark for SSE Instructions.

```
for i = 0; i < time; i++{
    for j = 0; j< repeat; j++{
        vmovaps 0x0(%rax), %zmm0
        vmovaps %zmm1, 0x16(%rax)
        vmovaps 0x32(%rax), %zmm2
        vmovaps %zmm3, 0x48(%rax)
        (...) //rax is updated at the end
    }
    vmovaps 0x0(%rax), %zmm0
    vmovaps %zmm1, 0x16(%rax)
    vmovaps 0x32(%rax), %zmm2
    vmovaps %zmm3, 0x48(%rax)
    (...)
}
```

achieve a predefined time duration, in order to increase the accuracy and stability of results. The inner loop contains 64 instructions, in order to exploit the capabilities of the Loop Stream Detector (LSD), presented in some Intel micro-architectures. Its utilization may increase benchmarking accuracy, by avoiding the utilization of the memory hierarchy for instruction fetching. Although the LSD is disabled in Intel Xeon Gold 6140 [21], this factor must be taken into account when evaluating other micro-architectures. Furthermore, in case that the LSD is not used, the loop fits completely in the L1 instruction cache, preventing evictions from this memory level. This avoids the utilization of the unified (instructions and data) L2 cache, which could affect the L2 bandwidth and degrade benchmark accuracy regardless of the functional unit being tested. Furthermore, in case that the number of tested instructions is not a multiple of the size of the inner loop, *i.e.*, 64 instructions, the remaining ones are placed outside this loop. To minimize data dependencies, the test code uses all available registers and, in order to increase accuracy, the results presented in this paper are obtained as the median across 1024 test code runs. To evaluate the amount of executed memory/arithmetic instructions, it was necessary to access the set of hardware counters presented in Table 2. These performance counters are used to validate the accuracy of the micro-benchmarks when performing the micro-architecture evaluation. The results from the experimental evaluation could also be obtained by only measuring the execution time of the benchmark and by counting the number of operations executed, which are defined a priori by the user as an input parameter. The micro-benchmarks are run with 18 threads, each bound to a single core, at nominal frequency, with the CentOS 7.5 operating system and compiled with Intel Compiler 19.03. Hyper-threading and turbo boost were turned off during experimental evaluation.

The performance of FP units for different ISA extensions is presented in Fig. 2. As expected, maximum performance (*i.e.*, F_P) is attained when executing AVX512 FMA instructions. For the same data precision, ADD/MUL AVX512 only delivers half of FMA flops (16 for DP and 32 for SP (single-precision)), hence their performance is half of F_P. Besides, DP ADD/MUL SSE and scalar FP instructions only attain one eighth and one sixteenth of F_p, respectively, since they handle smaller data width than AVX512 instructions. SP scalar FMA and ADD/MUL attain the same performance of their DP scalar counterpart since both execute the same amount of FLOPS per instruction (1 FLOP). While for AVX512, AVX and SSE extensions, FMA SP and ADD/MUL SP performances are double of their DP equivalents, the same scenario does not occur for divisions (DIV). For AVX512 DIV and AVX DIV, SP performance is 3.1× higher than DP performance, while for SSE, SP performance is 2.6× higher than DP. Although scalar DIV also executes the same amount of FLOPS for SP and DP (1 FLOP per instruction), their performance is slightly different, demonstrating that divider unit performance has a high dependency on the data precision.

The memory subsystem results (Fig. 3) show that the maximum bandwidth is achieved when requests are served by the L1 cache, when executing AVX512 instructions. There is a reduction in the sustainable bandwidth when data is fetched from the memory positioned further away from the core (*i.e.*, L2, L3 or DRAM). Similarly to FP units, SSE and scalar DP instructions achieve lower bandwidth than AVX and AVX512 DP instructions, since their vector length can only handle 16 and 8 bytes at a time, respectively.

Memory bandwidth is also affected by the utilization rate of memory dispatch ports, *i.e.*, the amount of load (LD) and store (ST) instructions performed, and by the accessed memory level. As shown in Fig. 3, the highest bandwidth is obtained when all accesses are served by L1 cache and for 2 loads and 1 store (2LD/ST) ratio. The theoretical L1 bandwidth is experimentally achieved for LD and LD/ST (5299.2/s), and for ST (2649.6/s). However, for 2LD/ST ratio, only 69.2% of the theoretical bandwidth is achieved, *i.e.*, 5562.42 GB/s. This result corresponds to the maximum sustainable L1 bandwidth stated by Intel in [21] (≈133 bytes per cycle). On the other hand, the ST ratio corresponds to the minimum sustainable bandwidth in all memory levels, independently of the ISA extension and load/store ratio. L2 and DRAM bandwidths are also greatly impacted by load/store ratio and their maximum corresponds to the case when only LD operations are performed, achieving for AVX512 instructions around 2630.12 GB/s and 41.41 GB/s, respectively.

The diverse features supported by modern micro-architectures are exercised differently by the real-world applications. As it is possible to observe in Figs. 2 and 3, this can lead to diverse FP performance and memory bandwidth upper-bounds, depending on the execution scenario, *e.g.*, ISA extension used, the amount of loads and stores, etc. Thus, it is crucial to take into account application requirements when performing their characterization, in order to attain an accurate identification of application bottlenecks. To attain this objective, it is necessary to perform an

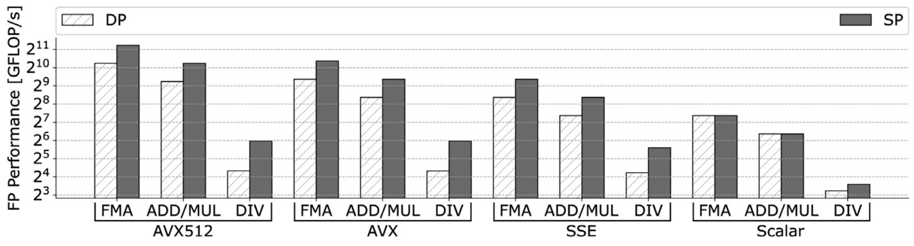


Fig. 2. Floating-point performance for different instruction set extensions and data size: DP — Double Precision; SP — Single Precision.

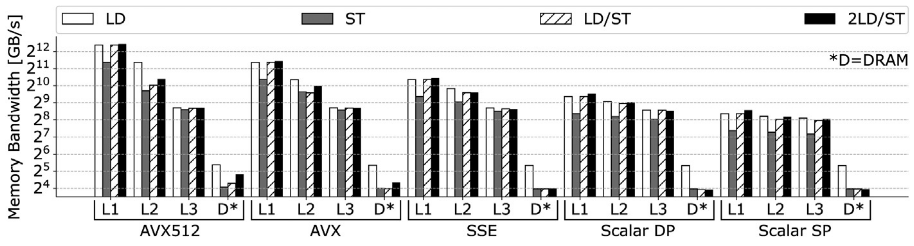


Fig. 3. Memory bandwidth for different instruction set extensions and data size: LD — Load; ST — Store.

evaluation of the micro-architecture capabilities according to the application specifics, to correlate the application behavior with the underlying hardware. While some of this information could be obtained with open-source benchmarks, such as SHOC [23], these benchmarks are not flexible enough to evaluate the performance upper-bounds of the modern architectures for different factors, which is essential to build the proposed adCARM.

4. Proposed models and metrics

As shown through in-depth micro-architecture benchmarking, FP performance and memory bandwidth depend on multiples factors, *e.g.*, different FP instructions, accesses to different memory levels, etc. Since applications may exercise a diverse set of hardware components during their execution, it is essential that insightful models, such as CARM, can take into account the features of each application, providing a more accurate characterization and more precisely pinpoint the potential execution bottlenecks, thus allowing for improved optimization hints.

To include this information in CARM, it is necessary to scale the upper-bounds of the compute and memory regions according to application specifics. Due to the support of a wide number of instruction types, this scaling depends on several factors, such as, ISA extension used and the type of executed operations (*e.g.*, FMA, ADD, etc.). For example, an application that only performs scalar instructions must be characterized in a model whose roofs correspond to the maximum performance of the micro-architecture under these conditions. In this section, the upper-bounds of the model are derived in order to encapsulate the impact of these factors in a single model, resulting in the adCARM. The notations used in the model derivation are presented in Table 3.

4.1. Scaling the performance upper-bounds

As referred previously, modern processors such as the Intel® Xeon® 6140 Gold were target of several enhancements in order to keep up with the growing performance demands. One example of these improvements is the support of multiple instruction types, which perform diverse operations over different data sizes. This greatly affects how each application exploits the hardware capabilities, since it influences both FP performance and memory bandwidth.

In particular, each memory instruction of type "i" (*e.g.* AVX512, AVX, SSE, Scalar, ...) requests a different amount of data, which affects the maximum attainable bandwidth. By assuming that each instruction type "i" is served at its maximum rate, the maximum attainable bandwidth of the memory level $y \in \{L1, L2, \ldots, DRAM\}$ for a load/store ratio "r" (B_r^y) can be defined as:

$$B_r^y = \frac{\beta}{T_{MEM}} = \frac{\beta}{\sum_i T_{\beta_i}} = \frac{\sum_i \beta_i}{\sum_i \frac{\beta_i}{B_{i,r}^y}} = \frac{\sum_i R_i^\beta \times \#\beta_i}{\sum_i \frac{R_i^\beta \times \#\beta_i}{B_{i,r}^y}}, \quad (1)$$

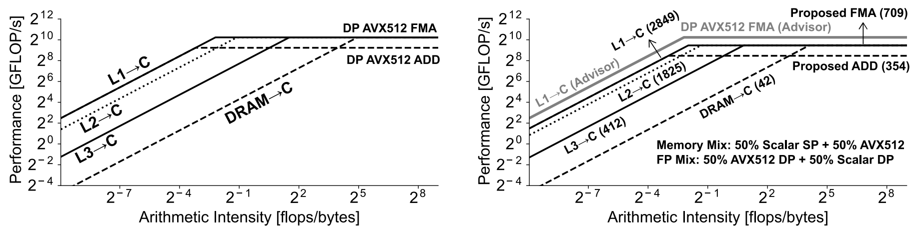
where $T_{MEM} = \sum_i T_{\beta_i}$ is the time to serve all memory requests, β is the total amount of data requested by the application (in bytes), β_i is the total amount of bytes requested by the instruction type "i". $R_i^\beta = \frac{INST_i^\beta}{INST_{MEM}}$ is the ratio of memory instructions "i" over the total amount of memory instructions of the application, $\#\beta_i$ is the amount of bytes operated at the level of a single instruction of type "i" (*e.g.*, $\#\beta_i = 64$ bytes for AVX512 or 16 for SSE) and $B_{i,r}^y$ is the maximum attainable bandwidth of memory level "y" for a given load/store ratio "r" and instruction type "i". The subset of values of $B_{i,r}^y$ is given in Fig. 3. For example, an application dominated by loads, with memory breakdown of 50% AVX512 ($i = 0$) and 50% SSE ($i = 1$), has maximum attainable bandwidth in L1 cache equal to

$$B_r^y = \frac{0.5 \times 64 + 0.5 \times 16}{\frac{0.5\times64}{5288.75} + \frac{0.5\times16}{1319.28}} = 3301.8 \text{ GB/s}, \quad (2)$$

i.e., 0.6 times lower than the absolute maximum sustainable bandwidth of L1 cache when only AVX512 is considered (5562.4 GB/s), which is utilized by the SoA approaches for architecture modeling. Hence, characterizing an application without taking into account its memory mix may provide misleading conclusions regarding the application potential to exploit the maximum capabilities of the system, and may even result in less accurate hints about its execution bottlenecks.

Similar concept can be applied to performance of FP units. By assuming that each FP instruction type "i" is executed at its maximum performance, the maximum attainable performance (P_a) is given by:

$$P_a = \frac{\phi}{T_{FP}} = \frac{\phi}{\sum_i T_{\phi_i}} = \frac{\sum_i \phi_i}{\sum_i \frac{\phi_i}{P_i}} = \frac{\sum_i R_i^\phi \times \#\phi_i}{\sum_i \frac{R_i^\phi \times \#\phi_i}{P_i}}, \quad (3)$$



(a) Intel® Advisor CARM 19.3. (b) adCARM.

Fig. 4. Comparison between Intel® Advisor CARM 19.3 and adCARM.

Table 3
Description of parameters used in Sections 4.1 and 4.2.

Parameter	Description
β_i	Total amount of bytes served by instructions of type 'i'
$\beta = \sum_i \beta_i$	Total amount of bytes of the application
T_{β_i}	Time to serve memory requests of instruction type 'i'
$T_{MEM} = \sum_i T_{\beta_i}$	Time to serve all memory requests
$\#\beta_i$	Number of bytes served by each instruction type 'i'
$INST_i^{\beta}$	Total amount of memory instructions of type 'i'
$INST_{MEM}$	Total amount of memory instructions
$R_i^{\beta} = \frac{INST_i^{\beta}}{INST_{MEM}}$	Ratio of memory instructions of type 'i'
$B_{i,r}^{y}$	Maximum bandwidth of memory level 'y', load/store ratio 'r' and instruction type 'i'
B_r^{y}	Scaled bandwidth for the memory level 'y' and load/store ratio 'r'
T_{ϕ_i}	Time to execute all FP instructions of type 'i'
$T_{FP} = \sum_i T_{\phi_i}$	Time to execute all FP instructions
ϕ_i	Total amount of flops performed by instruction type 'i'
$\phi = \sum_i \phi_i$	Total amount of flops performed
P_i	Maximum performance of FP instructions of type 'i'
$\#\phi_i$	Number of flops executed by each instruction of type 'i'
$INST_i^{\phi}$	Total amount of FP instructions of type 'i'
$INST_{FP}$	Total amount of FP instructions
$R_i^{\phi} = \frac{INST_i^{\phi}}{INST_{FP}}$	Ratio of FP instructions of type 'i'
P_a	Scaled FP performance
ϕ_M	Total amount of masked flops performed
T_M^{ϕ}	Execution time of masked FP instructions
η	Masking utilization
ϕ_U	Total amount of unmasked flops
T_U^{ϕ}	Execution time of unmasked FP instructions
P_U^{ϕ}	Performance of unmasked FP instructions
P_M^{ϕ}	Maximum performance of masked FP instructions
$F_{a,r}^{y}$	Application-driven CARM

where T_{FP} is the total time to perform FP instructions, ϕ is the total amount of FP operations (flops) executed by the application, while ϕ_i is the total amount of flops delivered by all instructions of type "i" contained in the application. $R_i^{\phi} = \frac{INST_i^{\phi}}{INST_{FP}}$ is the ratio of FP instructions "i" over the total amount of FP instructions of the application, $\#\phi_i$ is the amount of flops delivered by a single instruction of type "i" (*e.g.*, 16 flops for DP AVX512 FMA or 4 flops for SP SSE ADD/MUL) and P_i is the maximum attainable performance of the architecture for the instruction of type "i". The subset of values of P_i is presented in Fig. 2.

In order to improve the accuracy of the calculated upper-bounds, the maximum FP performance should also consider the utilization of masked FP instructions, which is supported by recent Intel micro-architectures. These instructions perform computations only over a subset of the data contained in a register and are able to prevent performance degradation in certain scenarios (*e.g.* these instructions can be used instead of "if" statements). While the execution time of masked instructions (T_M^{ϕ}) is equal to the execution of unmasked instructions (T_U^{ϕ}), the number of executed flops vary significantly, thus affecting the maximum attainable performance. In this scenario, the maximum FP performance when executing masked instructions (P_M^{ϕ}) is defined as:

$$P_M^{\phi} = \frac{\phi_M}{T_M^{\phi}} = \frac{\eta\phi_U}{T_U^{\phi}} = \eta \times P_U^{\phi}\,, \tag{4}$$

where ϕ_M is the number of masked flops executed, η is the masking utilization, *i.e.*, the percentage of unmasked flops that are executed and P_U^{ϕ} is the maximum performance of unmasked operations. Eq. (4) can be applied to Eq. (3), in order to increase the insightfulness of the adCARM.

It is also worth to mention that in this work, Eq. (3) is separately applied to instructions that perform the same type of operations (for example, FMA and ADD) by taking into account different data precision and instruction set extension mixes. Otherwise, the compute bound region of the herein proposed adCARM would only contain one roof, which would reduce model insightfulness. Besides, the dispatch ports for ADD/MUL and FMA are shared with high-latency instructions, *e.g.*, divider and square-roots. Hence, in this scenario, assuming maximum performance for all instruction types "i" supported by the micro-architecture, may lead to a scaling of the roofs that do not reflect the system upper-bounds for that execution scenario.

4.2. adCARM: Application-Driven CARM

From Eqs. (1) and (3), it is possible to derive an analytical model for the proposed adCARM, which states that the maximum attainable floating-point performance ($F_{a,r}^{y}$) of the architecture is calculated as:

$$\begin{aligned} F_{a,r}^{y} &= \min\{AI \times B_r^{y}, P_a\} \\ &= \min\{AI \times \frac{\sum_i R_i^{\beta} \times \#\beta_i}{\sum_i \frac{R_i^{\beta} \times \#\beta_i}{B_{i,r}^{y}}}, \frac{\sum_i R_i^{\phi} \times \#\phi_i}{\sum_i \frac{R_i^{\phi} \times \#\phi_i}{P_i}}\}\,. \end{aligned} \tag{5}$$

The Figs. 4a and 4b present Intel® Advisor CARM 19.3 and the adCARM for a certain mix of instructions, respectively. By comparing both models, it is possible to observe that by taking into account the characteristics of diverse instruction types, adCARM tunes the optimization space to the right dimension, showing that certain applications might not be able to achieve the maximum attainable performance of the processor (as considered in the SoA approaches). Hence, the proposed method is able to encapsulate diverse characteristics of modern applications, providing tighter performance upper-bounds, which leads to more accurate and insightful hints regarding application execution bottlenecks.

While the approach presented in this work can be applied to the compute roofs of current roofline approaches, the scaling of the memory roofs is a property of CARM approach and cannot be applied to ORM or Hierarchical ORM. In detail, ORM and Hierarchical ORM consider the maximum bandwidth between consecutive memory levels, which is not affected by the data size involved in the memory accesses. Hence, independently of the type of memory accesses performed by the application, ORM and Hierarchical ORM use always the same bandwidth roofs, while the proposed adCARM is able to fully adapt to application characteristics.

To build and use the adCARM two steps have to be fulfilled: **(1)** analysis of the instruction mix of the application in order to extract its specifics, and **(2)** an experimental evaluation according to application characteristics obtained in the previous step, by using micro-benchmarks that explore multiple processor features. These steps can be automatized in Intel® Advisor, by extending its instruction mix analysis and benchmarking capabilities, in order to extract the information necessary to build the proposed adCARM, allowing the users to fully focus on application characterization and optimization.

Moreover, the adCARM is not limited to a single micro-architecture. The experimental evaluation can be applied to other micro-architectures by porting the micro-benchmarks described in Section 3. Regarding the step 1, the instruction mix of the application can be obtained by using tools like Intel Software Development Emulator (SDE), which supports a wide range of Intel micro-architectures, or by using open-source tools such as DynamoRIO [24]. Tools based on performance counters could also be used to obtain some of the data related to application execution, *e.g.*, amount of loads, stores and FP instructions and cache misses. However, there are no counters available in Intel micro-architectures that allow decoupling the size of each memory access (64 bytes, 32 bytes, etc.), neither classifying the type of FP instructions (FMA, ADD, DIV).

Although adCARM roofs are scaled according to application specifics, it inherits the same simple and insightful characterization methodology from the corresponding SoA CARM. Thus, the methodology described in Section 2 can be used to take advantage of the proposed adCARM, *i.e.*, **(1)** verification of the model region where the application is placed (memory, memory and compute or compute regions) in order to narrow the optimization strategy and **(2)** plotting a vertical line at the AI of the application and observe the intersections of this line with the model roofs in order to identify the potential execution bottlenecks.

4.3. Proposed metrics

While adCARM inherits the simple and insightful characterization methodology from SoA CARM, it also contains some of CARM limitations. In particular, although CARM nature allows the model to show the existence of memory locality and cache reuse, it may not be able to pinpoint the exact memory level that should be the main focus of optimization. This can be considered as the main weakness of CARM. For example, for a given memory bound application, if 75% of memory requests are served by L1 cache and 25% are served by DRAM, the application performance might be placed between L2 and L3 roofs. This effect suggests the existence of data locality in private caches but it does not allow to derive easily that DRAM traffic must be the main focus of optimization. Aiming at tackling this issue, two additional CARM-based metrics are proposed to complement CARM analysis: memory share and memory impact.

Memory share of memory level "y" (MS^y) corresponds to the amount of data that is served by each memory level over the total amount of bytes of the application (β) and it is given by:

$$MS^y = \frac{\beta^y}{\beta}, \tag{6}$$

where β^y is the amount of data requested by the core and served by memory level "y", after the data transverses all previous memory levels. This corresponds to the memory traffic approach taken by CARM. This metric allows verifying the existence of cache reuse and the success of memory-related optimization. For example, after applying the cache blocking technique, it is expected to visualize an increase of cache memory shares, while DRAM share decreases, whereas in the herein adCARM, application performance gets closer to cache roofs.

On the other hand, memory impact of memory level "y" (MI^y) corresponds to the fraction of time spent serving the requests of memory level "y" from a CARM perspective, *i.e*, between the core and the memory level "y". MI^y is calculated as:

$$MI^y = \frac{\frac{\beta^y}{B^y}}{\sum_m \frac{\beta_m}{B_m}}, \tag{7}$$

where B^y is the bandwidth of memory level "y" as considered in the adCARM (see Eq. (1)), β^y is the amount of data served by memory level "y" and $\sum_m \frac{\beta_m}{B_m}$ ($m \in \{L1, L2, \ldots, DRAM\}$) is the total time spent at serving all memory requests. In order to obtain a more accurate memory impact, the formula derived in Eq. (1) is used in Eq. (7), assuming that all memory levels execute the same relative amount of each type of instruction, *i.e.*, R_i^β is equal for all the levels. While MS^y pinpoints the existence of cache utilization, MI^y defines which memory level must be the main optimization focus. For example, since DRAM has much lower bandwidth than caches (as shown in Section 3), even if MS^y shows that majority of data is served by caches, it is still possible that DRAM is the main bottleneck.

It is worth to mention that the derivation of Eqs. (5)–(7) slightly resemble some of capacity equations proposed in [25, 26]. However, while the main focus of capacity equations is the co-design of hardware according to software requirements (or vice-versa), the adCARM aims at providing a more accurate characterization of applications, in order to select the best optimization techniques that allow the improvement of application performance. In fact, as it will be experimentally shown in Section 5, these proposed models and metrics provide a more accurate characterization of real-world applications when mixing multiple ISA extensions and different load/store ratios, when compared to the SoA roofline modeling approaches.

5. Application characterization

As shown in Section 3, micro-architecture upper-bounds depend on multiple factors, imposed by the application-specific requirements. Hence, to achieve an accurate characterization of applications, it is essential to consider application specifics when decoupling their bottlenecks and deriving an optimization strategy. With this aim, this work proposes the adCARM, which is able to adapt to the requirements of real-world applications, providing precise optimization hints to achieve an efficient execution in modern systems, to fully exploit their capabilities. In this Section, the impact of ISA extensions in roofline modeling approaches is analyzed through in-depth characterization of finite difference code (ISO-3DFD) [27,28], by evaluating the modeling insights provided for AVX512 and scalar versions of the application. Furthermore, a set of proxy applications from Exascale Computing Project Proxy Application Suite [29] is evaluated in Intel® Advisor CARM 19.3 and with the proposed adCARMs in order to assess the improvements derived from the utilization of the proposed model.

The applications were executed in the processor presented in Table 1 and in the same conditions applied for the micro-architecture benchmarking, *i.e.*, with 18 threads, each bound to a single core, at nominal frequency, with the CentOS 7.5 operating system and compiled with Intel Compiler 19.03. Hyper-threading and turbo boost were turned off during application execution. The performance and arithmetic intensity of each application are obtained through Intel® Advisor 19.3 analysis. The cache simulation tool contained in this framework provides an estimation of the data traffic between memory levels, which is used herein to derive the memory share and memory impact of the applications. However, this tool does not decouple the type of memory requests served by each memory level (*e.g.* 8B, 16B, etc.). This information could potentially be obtained through complex and time-consuming cycle-accurate simulators, which would contribute to increase the complexity of the proposed approach. To overcome these issues, it is considered that all memory levels perform the same relative amount of instruction types when deriving the memory impact and memory share metrics, which maintains methodology simplicity. Furthermore, Intel® Software Development Emulator (Intel® SDE) 3.0 [30] is used to obtain the FP and memory instruction mixes of each application, by following the methodology presented in [31]. The information obtained from SDE is then used to build the proposed models. The total amount of loads and stores requested by the core (that is used to calculate the load/store ratio of the application) is also obtained from Intel® SDE.

Algorithm 3: Pseudo-code of the main loop of ISO-3DFD

```
#define FA(ix, off) ((prev[ix+off] + prev[ix-off]))
#define HL 8

//Cache Blocking Loops
//Blocking Factors bx, by, bz
//Calculation of izEnd, iyEnd and ixEnd
for (iz=bz; iz < izEnd; iz++){
    for (iy=by; iy < iyEnd; iy++){
        next = &next_base[iz*dimz + iy*n1 + bx]
        prev = &prev_base[iz*dimz + iy*n1 + bx]
        vel = &vel_base[iz*dimz + iy*n1 + bx]
        //__assume_aligned to cache line size
        //Pragmas for prefetching distance
        for (ix=0; ix < ixEnd; ix++){
            val = prev[ix]*c0 + c1*(FA(ix,1) + FA(ix, v1) + FA(ix,f1))
                              + c2*(FA(ix,2) + FA(ix, v2) + FA(ix,f2))
                              + c3*(FA(ix,3) + FA(ix, v3) + FA(ix,f3))
                              + c4*(FA(ix,4) + FA(ix, v4) + FA(ix,f4))
                              #if(HL == 8)
                              + c5*(FA(ix,5) + FA(ix, v5) + FA(ix,f5))
                              + c6*(FA(ix,6) + FA(ix, v6) + FA(ix,f6))
                              + c7*(FA(ix,7) + FA(ix, v7) + FA(ix,f7))
                              + c8*(FA(ix,8) + FA(ix, v8) + FA(ix,f8))
                              #endif;

            next[ix] = 2.0*prev[ix] - next[ix] + val*vel[ix]
        }
    }
}
```

In order to validate the insights given by the proposed models, Top-Down analysis [7] is obtained through the Bottom-Up from the General Exploration in Intel® VTune™Amplifier 2019 (Update 3). Top-Down decouples the application execution bottlenecks in several categories, namely: retiring (RET), core-bound (CB), memory-bound (MB), bad speculation (BS) and frontend-bound (FE). RET and CB are mainly related to the utilization of dispatch ports in the micro-architecture, while MB represents the memory bottlenecks, mostly related to stalls inducted by memory accesses. BS and FE represent bottlenecks from branch misprediction and backend starvation.

5.1. Impact of instruction set extensions on existing roofline methodologies

ISO-3DFD [27,28] implements a finite difference method with isotropic, which can be used in several real-world applications (*e.g.*, seismic modeling, wave propagation). According to Intel® Advisor 19.3 analysis, this application only contains one main kernel (*loop at iso-3dfd_parallel.cc:107*). As it can be observed in Algorithm 3, this kernel performs an 8th-order and 3-dimensional stencil computation and the version used in this work already includes some optimizations related to memory accesses, such as, cache blocking and prefetching optimizations (through pragmas). Due to the nature of stencil computations, this loop is expected to be mostly memory bound, especially in what concerns the accesses to the elements in "z" direction that do not allow fully exploiting the data locality in privates caches, in contrast to the elements in "x" and "y" directions whose cache reuse is improved by cache blocking. To attain the effects of different ISA extensions on the characterization of state-of-the-art roofline modeling methods, two versions of ISO-3DFD are considered – Scalar and AVX512 versions – which are obtained by compiling the code with *no-vec* and *xCOMMON-AVX512* flags, respectively.

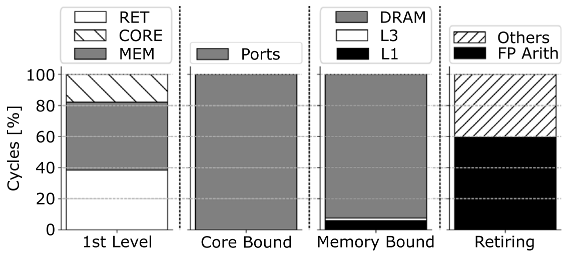
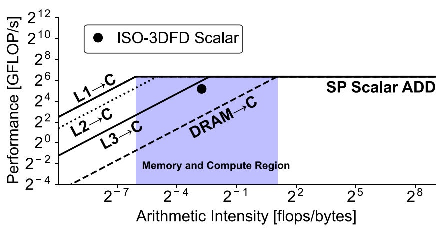
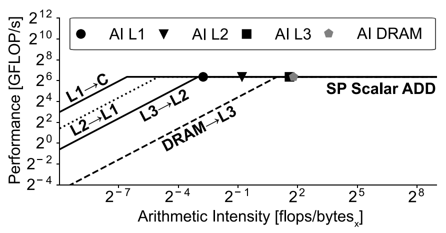


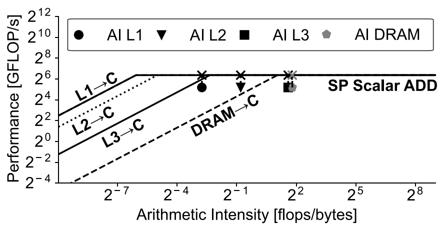
Fig. 5. Top-Down analysis for ISO-3DFD Scalar.

Top-Down analysis of Scalar ISO-3DFD, presented in Fig. 5, identified three main components in the 1st level: memory bound (43.6%), retiring (38.5%) and core bound (17.9%). Although the core bound contribution is significantly smaller than the one of memory bound, there is a considerable impact from retiring that may suggest that application performance can be limited by in-the-core or close-to-the-core components. However, 40% of the retiring belongs to "Others" category, indicating that non-FP instructions, such as memory requests, have a big impact in kernel execution. On the other hand, 92.3% of the memory bound stalls originate from DRAM, hinting that this application is likely to be limited by memory accesses, with very limited potential to reach any compute roof. However, by following the characterization methodologies presented in Section 2, it is possible to verify that the state-of-the-art approaches, presented in Figs. 6a (Intel® Advisor CARM 19.3), 6b (Hierarchical ORM) and 6c (IRM), do not corroborate the Top-Down analysis. While IRM and Hierarchical ORM characterize the kernel as pure compute bound, Intel® Advisor CARM 19.3 places the loop below the L3 roof in the compute and memory bound region, implying the potential of this kernel to achieve the SP Scalar ADD roof, while being mainly limited by L3 and DRAM.

By analyzing the instruction mix of Scalar ISO-3DFD, it is possible to verify that existing roofline methods do not adapt to application specifics. The FP instruction mix of this kernel contains only SP Scalar ADD instructions, while its memory mix shows the existence of data requests of 4B (65%) and 8B (35%). Besides, its load/store ratio is around 63, hence its memory footprint is dominated by load instructions. By scaling the roofs according to the application specifics, it can be observed that the memory roofs in the adCARM (Fig. 6d) move down when compared with the Intel® Advisor CARM 19.3 (Fig. 6a), due to the lower bandwidth attained by 4B and 8B requests when compared to AVX512 instructions. This effect increases the area of the memory region to the right, causing the kernel to be placed on top of L2 roof and to be classified as memory bound in the adCARM, as expected from Top-Down analysis. Accordingly, a possible optimization direction to improve application performance would be to improve the memory accesses, by using instructions that can handle bigger data sizes (*e.g.* AVX512).


(a) Intel® Advisor CARM 19.3.


(b) Hierarchical ORM.


(c) Integrated Roofline Model.

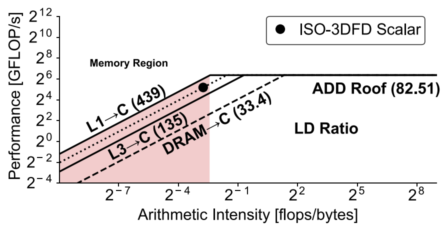

(d) adCARM.

Fig. 6. Characterization of ISO-3DFD Scalar version in state-of-the-art roofline approaches and adCARM.

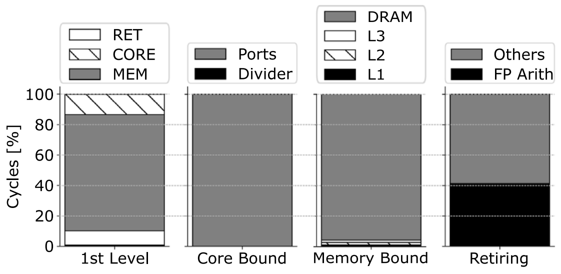


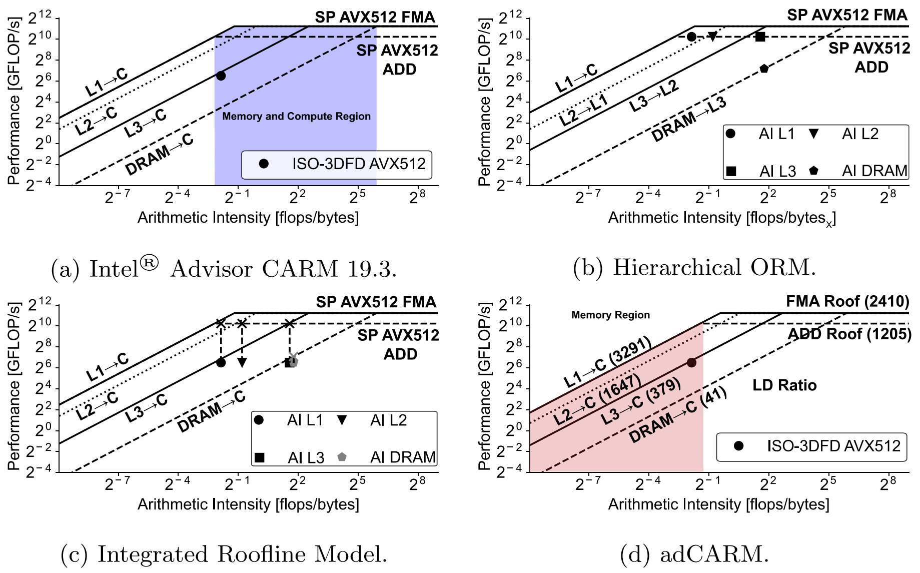
Fig. 7. Top-Down analysis for ISO-3DFD AVX512.

The hints provided by the adCARM are also confirmed by the proposed memory metrics. The memory share metric indicates that 97% of the data requested by Scalar ISO-3DFD is served by L1 cache, while only 1% is served by DRAM. A similar scenario occurs from memory impact perspective. L1 cache impact is around 89%, while DRAM only has an impact of 7%. It is worth emphasizing that a large impact for the L1 cache should be interpreted as a “positive impact” to the overall application performance, while the impact presented for all the other memory levels results in performance degradation, thus it should be interpreted as a “negative impact”. In fact, for a purely memory-bound kernel, the maximum performance can only be achieved with 100% of L1 impact (and traffic). As such, the L1 impact can be used as a relative measure of code optimization state (higher is better), while the impact value for other memory levels should be used as an additional measure for precise bottleneck detection (lower is better). In other words, the negative impact of accesses to other memory levels is the one that prevents the application from achieving the highest value for L1 accesses (*i.e.*, 100%). Based on these metrics it is possible to conclude that ISO3D-FD Scalar has its memory accesses very well optimized, corroborating with adCARM characterization, *i.e.*, limited by private caches. Due to DRAM accesses, the kernel is not able to attain the L1 roof and, for further performance improvements, DRAM should still be the main focus of optimization. These conclusions are quite different from the ones reached with the not-application-driven roofline models.

Regarding ISO-3DFD AVX512 kernel, its Top-Down analysis (Fig. 7) is dominated by memory bound (76.3%), with most the stalls coming from DRAM (95.8%). Retiring and core contributions are 9.2% and 13.4%, respectively. Due to the high memory component and low retiring and core, ISO-3DFD AVX512 is expected to be completely limited by memory. Similar characterization is obtained with the state-of-the-art rooflines, which are presented in Figs. 8a (Intel® Advisor CARM 19.3), 8b (ORM) and 8c (IRM). ORM and IRM hint that the main bottleneck is DRAM, which supports Top-Down analysis. However, these methods do not provide any insights related to the possible existence of cache utilization and memory locality, since their aim is to pinpoint the main execution bottleneck. Intel® Advisor CARM 19.3 places the loop close to L3 cache, indicating that some of the data traffic is not related to memory stalls and is handled by caches. However, Intel® Advisor CARM 19.3 also hints that the kernel has the potential to attain the SP AVX512 ADD roof, which is highly unlikely due to the strong memory bound nature of the code.

The FP mix of AVX512 ISO-3DFD is composed of AVX512 instructions, namely SP AVX512 ADD (87%) and SP AVX512 FMA (13%). Since it uses AVX512 instructions and does not mix different data precision, the compute region of the proposed adCARM does not suffer any changes when compared with the existing roofline approaches, as it can be observed in Fig. 8d. On the other hand, its memory instruction mix contains two distinct request types, 8B (40%) and 64B (60%), and it is dominated by load instructions (load/store ratio of 14). This affects mainly the bandwidth of L1 and L2 caches, reducing the maximum attainable performance that can be achieved in the memory region of the proposed model. This causes the effect previously observed for ISO-3DFD Scalar when scaling the roofs, *i.e.*, the memory roofs move down in the adCARM, increasing the memory region area to the right. This changes the region where the kernel is placed from the memory and compute region (Intel® Advisor CARM 19.3) to the memory region (adCARM), which fully corroborates the Top-Down analysis. Similarly to the characterization of ISO-3DFD Scalar, the adCARM hints that to improve application performance it is necessary to apply optimizations related to memory. One possible strategy to improve application performance corresponds to increasing the granularity of the memory accesses, by reducing the amount of 8B transfers.

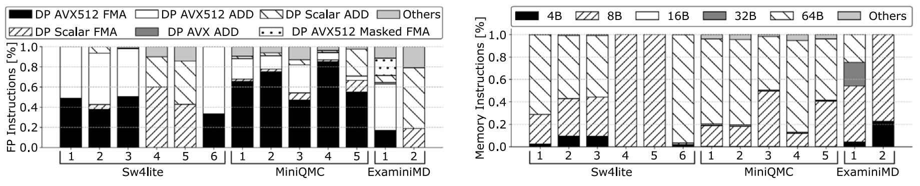
Finally, the memory share metric confirms the existence of cache utilization, with 42% of the traffic handled by L1, 47% by L2 and only 10% by DRAM, confirming the insights provided by adCARM, which places the kernel on top of L3 roof. From memory



(a) Intel® Advisor CARM 19.3. (b) Hierarchical ORM.

(c) Integrated Roofline Model. (d) adCARM.

Fig. 8. Characterization of ISO-3DFD AVX512 version in state-of-the-art roofline approaches and adCARM.



(a) FP Instruction Mix. (b) Memory Instruction Mix.

Fig. 9. The instruction Mix of exascale proxy applications.

Table 4
Application kernels characterized in Intel® Advisor CARM 19.3 and adCARM.

Application	Domain	Input set	#Kernel	Kernel name	SDE LD/ST ratio	LD/ST ratio used
Sw4lite	3D seismic modeling	LOH.1-h100.in	1	Loop at rhs4sg_rev.C:116	13.9	LD
			2	Loop at rhs4sg_rev.C:357	5.6	2LD/ST
			3	Loop at ew-cfromfort.C:991	21.9	LD
			4	Loop at ew-cfromfort.C:80	5.67	2LD/ST
			5	Loop at ew-cfromfort.C:49	3.34	2LD/ST
			6	Loop at ew-cfromfort.C:120	3.11	2LD/ST
MiniQMC	Monte Carlo	#Atoms = 256	1	Loop at ParticleBConds.h:184	2.43	2LD/ST
			2	Loop at MultiBspline.hpp:282	2.25	2LD/ST
			3	Loop at MultiBspline.hpp:89	2.43	2LD/ST
		#Electrons = 3072	4	Loop at TwoBodyJastrow.h:334	2.47	2LD/ST
			5	Loop at TwoBodyJastrow.h:316	2.68	2LD/ST
ExaMiniMD	Molecular dynamics	Region box block 0 128 0 128 0 128	1	Loop at force_lj_neigh_impl.h:178	8.08	LD
			2	Loop at force_lj_neigh_impl.h:270	11	LD

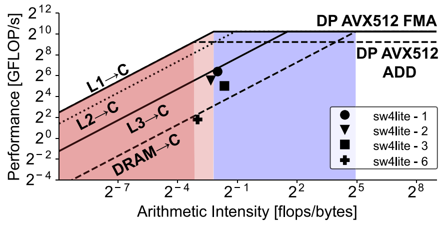
impact point of view, most of the performance is degraded due to DRAM (84%), while 14% of the impact is due to the private caches, pinpointing DRAM as the main bottleneck .

5.2. *Characterization of exascale proxy applications with Cache-aware Roofline Model*

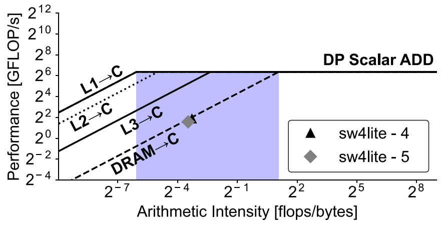
5.2.1. *Extracting the application-specific parameters for model construction*

Since exascale systems are yet to be released, it is not possible to evaluate the performance of applications when running on these machines. One solution is the utilization of proxy applications that represent, until a certain level, the characteristics of their full versions. For this reason, an accurate characterization of these applications is an important asset to extract useful information about their execution bottlenecks. Furthermore, the approach considered to derive the adCARM proposed in this work is general enough to be applied to any out-of-order processor, with several functional units and a memory hierarchy with several memory levels. This is the case of future exascale systems. Hence, adCARM is expected to be an important tool when optimizing application performance even in these future systems.

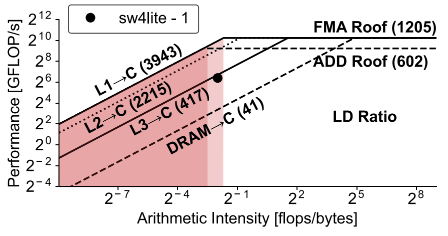
To demonstrate the usability and insightfulness of the adCARM, a set of proxy applications from Exascale Computing Project (ECP) Benchmark Suite [29] are characterized in Intel® Advisor CARM 19.3 and with the adCARM, namely: Sw4lite, MiniQMC and ExaminiMD. As it can be observed in Table 4, a total of 13 kernels are evaluated in this work. The majority of hotspots in these applications have a load/store ratio of 2 loads and 1 store, while only four loops are dominated by loads, *i.e.*, kernels 1 and 2 from Sw4lite and the two loops from ExaminiMD.



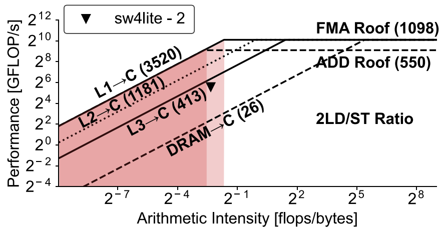
(a) Intel® Advisor CARM 19.3 - Kernels 1, 2, 3 and 6.



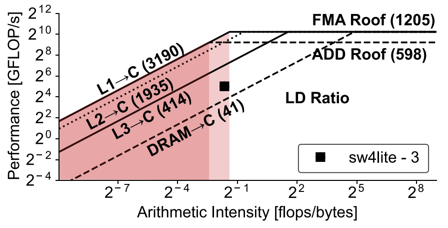
(b) Intel® Advisor CARM 19.3 - Kernels 4 and 5.



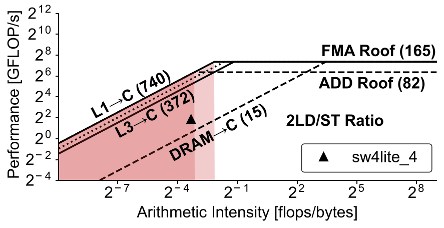
(c) adCARM - Kernel 1.



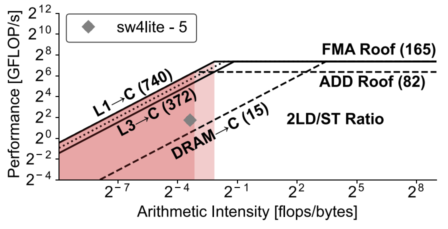
(d) adCARM - Kernel 2.



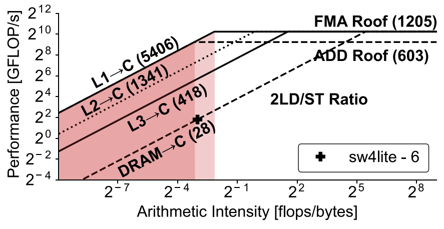
(e) adCARM - Kernel 3.



(f) adCARM - Kernel 4.



(g) adCARM - Kernel 5.



(h) adCARM - Kernel 6.

Fig. 10. Characterization of Sw4lite kernels in Intel® Advisor CARM 19.3 and adCARM.

The input sets of each application were selected in order to guarantee that all levels of the memory hierarchy were utilized, to provide a full analysis of the application behavior and their possible bottlenecks.

From the instruction mix of each kernel, obtained from the Intel® SDE, it is possible to verify that these loops cover a wide range of ISA extensions and instruction types. The FP instruction mix presented in Fig. 9a shows that while some kernels are completely dominated by FP AVX512 instructions (kernels 1, 2 and 6 from Sw4lite and kernels 1,2,4 and 5 from MiniQMC), others can be dominated by Scalar instructions (kernels 4 and 5 from Sw4lite and kernel 2 from ExaminiMD) or may contain significant contributions of scalar instructions mixed with vector instructions (kernel 3 from Sw4Lite, kernels 3 and 5 from MiniQMC and kernel 1 from ExaminiMD). The existence of scalar instructions impacts negatively the performance of these kernels, implying that they are not able to attain the maximum FP performance of the architecture if their main bottlenecks are related to computational units. It is also worth to mention that kernels 4 and 5 from Sw4lite, 3 from MiniQMC and kernels 1 and 2 from Examinimd have some impact from "Others" FP instructions (*e.g.* divisions, square roots, comparisons), which may contribute to further degrade their performance.

A similar scenario can be observed in the memory mix presented in Fig. 9b. Except the kernel 6 from Sw4Lite, which is completely dominated by memory transfers of 64 bytes, all the remaining kernels have significant contributions from diverse types of memory instructions, mainly from scalar requests, *i.e.*, 8B and 4B. For this reason, these applications may have their bandwidth degraded due to the lower performance of these instructions. Some kernels also have a small portion of "Others", which corresponds to 1B and 2B memory transfers. However, their contribution does not surpass the 5% for any application, hence their impact on the memory bandwidth is not very significant.

Since these applications contain a diverse instruction mix, at both FP and memory levels, their maximum attainable performance does not correspond to the architecture absolute maximums. Due to this instruction diversity in real applications, it is essential to consider the upper-bounds of the micro-architecture that meet the application requirements in order to provide an accurate characterization. As it was observed in the characterization of ISO-3DFD Scalar, by considering the instruction mix of the applications, the bottlenecks pinpointed by CARM can change

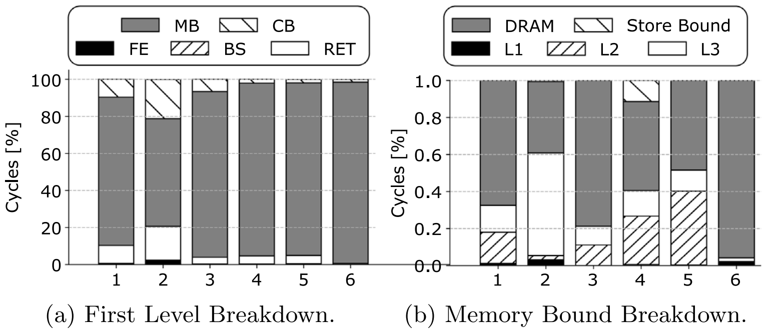


Fig. 11. Top-Down for Sw4Lite.

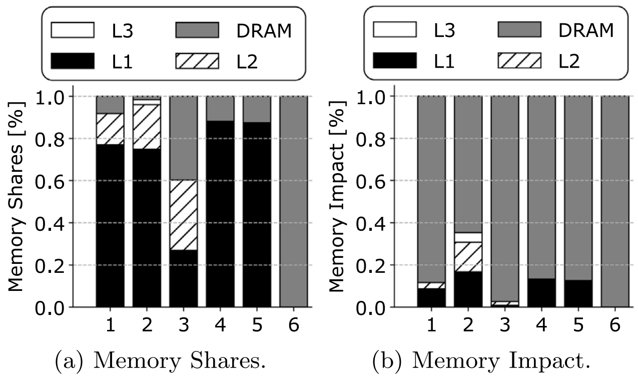


Fig. 12. Memory share and memory impact metrics for Sw4Lite.

significantly, thus changing the region of the model where the application is classified. This behavior is also expected for the ECP proxy applications evaluated in this work, since the scalar instructions presented in the memory and FP mixes of these workloads translate in a lower FP performance and memory bandwidth, lowering the roofs of the adCARM and impacting the application characterization.

5.2.2. In-depth characterization of exascale proxy applications

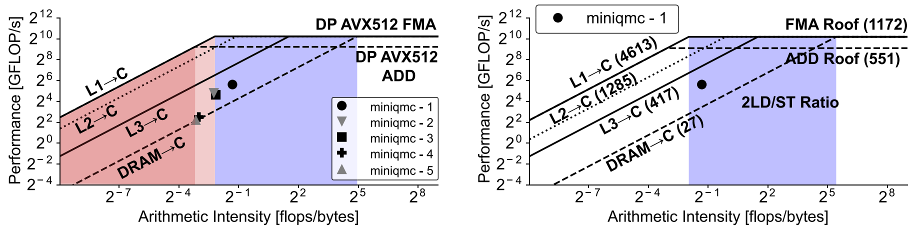
By using the instruction mix obtained from Intel® SDE (Figs. 9a and 9b), it is possible to build the adCARM for each kernel, by applying Eq. (5) presented in Section 4.2.

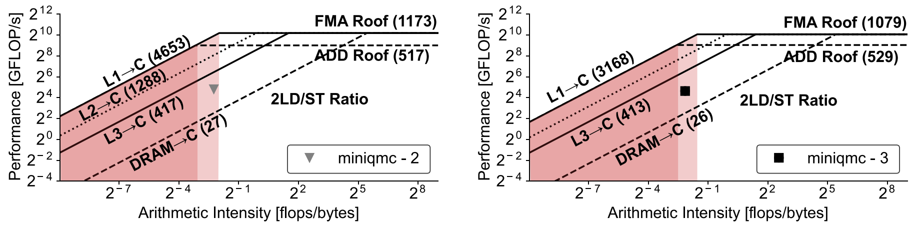
Sw4lite characterization in Intel® Advisor CARM 19.3 and proposed adCARMs is presented in Fig. 10. By comparing Intel® Advisor CARM 19.3 and adCARM characterization, Sw4lite kernels are, in general, close to the memory region in the adCARM, easily visualized by the approximation of the dots to the shaded areas presented in models. This confirms Top-Down insights, which characterizes all Sw4Lite kernels as highly memory bound, as it can be observed in Fig. 11a. Moreover, kernels 1, 2 and 3 are similarly characterized by the two models, although the adCARMs shorten up the optimization space by considering tighter upper-bounds for each kernel, as it can be observed by the lower attainable performance of L1 and L2 caches. These kernels are also placed between L3 and DRAM roofs on both models, hinting the existence of accesses to caches. On the other hand, it is possible to observe some changes between models in kernels 4, 5 and 6. Kernels 4 and 5 change their characterization region from "mixed" region in Intel® Advisor CARM 19.3 to memory bound region in the adCARM. These kernels also surpass the DRAM roof in the proposed model, suggesting some cache utilization, while in the state-of-the-art model they are placed on top of the DRAM roof. The characterization of kernels 1, 2, 3, 4 and 5 in the adCARM corroborates with the Top-Down results presented in Fig. 11b, where each kernel is significantly impacted by accesses to L2 and L3 caches. Kernel 6 is positioned on top of the DRAM roof in adCARM, while in Intel® Advisor CARM 19.3 it is below DRAM, hinting the existence of bottlenecks related to latency. The Top-Down analysis confirms adCARM insights, by defining this kernel as completely limited by DRAM bandwidth without hinting the existence of bottlenecks related to latency.

These observations are also confirmed by the perspective of the memory metrics proposed in this work. As it can be observed in Fig. 12a, most of the memory requests of kernels 1, 2, 3, 4 and 5 are handled by private caches with most of the requests served by L1 cache. Although kernel 3 has around 60% of its memory transfers served by private caches, the remaining accesses are related to DRAM, which contributes to reducing its performance. Moreover, kernel 6 requests are entirely handled by DRAM, which corroborates with the proposed CARM characterization, *i.e.*, it is positioned on top of the DRAM roof. Although the majority of the loops have some cache utilization, according to memory impact results, presented in Fig. 12b, DRAM is the main bottleneck of all Sw4lite kernels, which prevents the kernels from surpassing the L3 cache roof. To further improve application performance, it is necessary to reduce DRAM traffic, through memory-access related optimizations. For example, reducing the amount of scalar memory transfers (8B and 4B) executed by these loops (Fig. 9) could lead to performance improvements, since this type of instructions is associated with lower performance.

Similarly to Sw4lite, almost all of MiniQMC kernels move closer to the memory region of the adCARM when compared to the Intel® Advisor CARM 19.3, as it can be observed in Fig. 13. Once again, this is in accordance with the memory bound nature exposed by Top-Down analysis (Fig. 14a). Furthermore, it is possible to visualize in the memory breakdown of Top-Down analysis (Fig. 14b) that almost all MiniQMC kernels have their performance heavily impacted by DRAM accesses together with requests to private caches, which are responsible to place the kernels between L3 and DRAM in the adCARM. The kernel 1 is the only hotspot limited by the stores, *i.e.*, the store execution is the main source of stalls in the memory bound component.

Moreover, while kernels 1, 2 and 3 are positioned between L3 and DRAM roofs in both models, kernels 4 and 5 are placed above DRAM in the adCARM, whereas in Intel® Advisor CARM 19.3 these two kernels are positioned on top of DRAM roof. Hence, differently from Intel® Advisor CARM 19.3, adCARM hints the existence of cache utilization for all kernels, which corroborates with memory share metric, presented in Fig. 15a. As pinpointed by memory shares, most of the memory accesses are served by L1 cache in all kernels, which boost application performance, placing all the kernels above the DRAM roof in the application-driven model. However, as it is shown in Fig. 15b, DRAM still dominates the memory impact of the kernels, avoiding the kernels from outperforming L3 roof. It is worth to mention that the kernel


(a) Intel® Advisor CARM 19.3 - Kernels 1, 2, 3, 4 and 5.
(b) adCARM - Kernel 1.


(c) adCARM - Kernel 2.
(d) adCARM - Kernel 3.

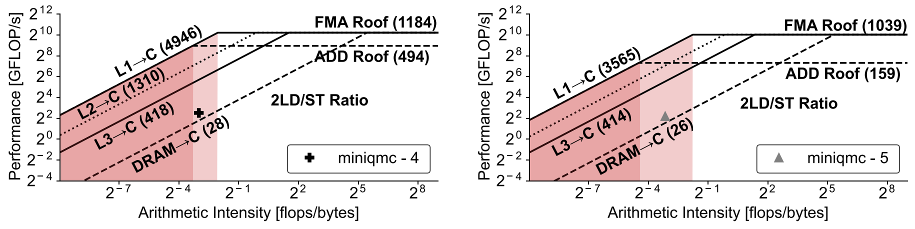

(e) adCARM - Kernel 4.
(f) adCARM - Kernel 5.

Fig. 13. Characterization of MiniQMC kernels in Intel® Advisor CARM 19.3 and adCARM.

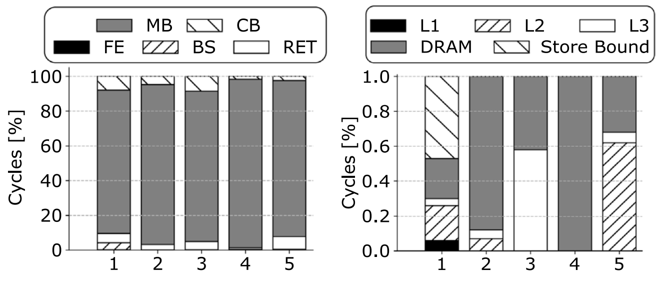

(a) First Level Breakdown. (b) Memory Bound Breakdown.

Fig. 14. Top-Down for MiniQMC.

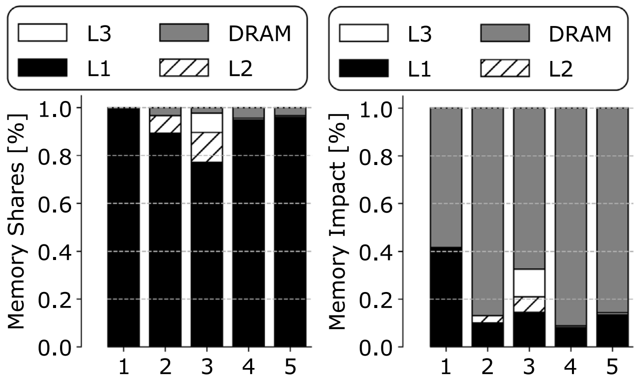
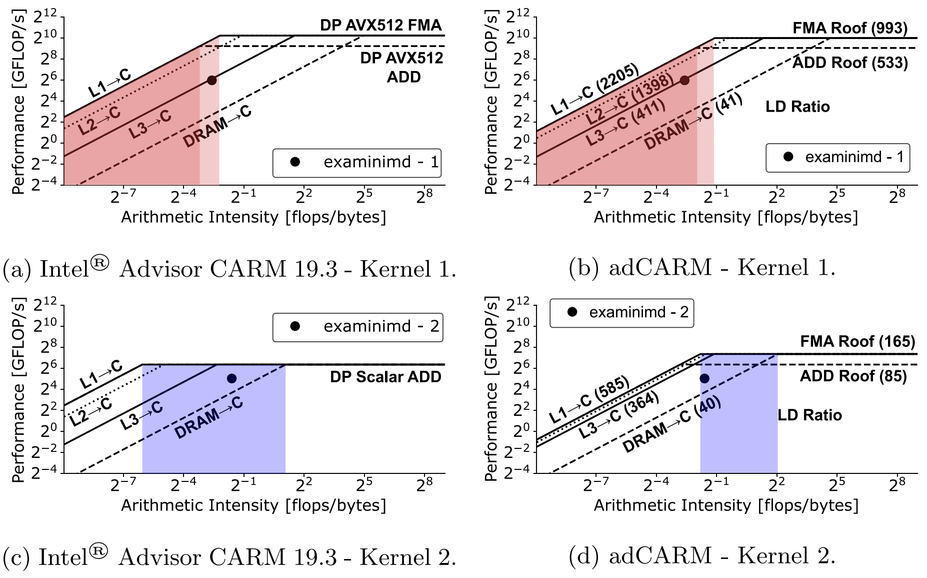

(a) Memory Shares. (b) Memory Impact.

Fig. 15. Memory share and memory impact metrics for MiniQMC.

with higher performance, *i.e.*, MiniQMC-1, is the kernel with the highest L1 impact (49%).

Finally, ExaminiMD characterization is presented in Fig. 16. As it can be observed, kernel 1 is characterized as purely memory bound in the adCARM. However, differently from Sw4lite and MiniQMC kernels, ExaminiMD loops have high contributions from retiring and core bound, showing the potential to be compute bound (Fig. 17a). ExaminiMD-1 is balanced between these three Top-Down components, suggesting bottlenecks in both memory hierarchy and core resources. On the other hand, its retiring breakdown indicates that these contributions come from non-FP instructions, *i.e.*, they can be related to memory requests and other types of instructions. This hints that the high retiring component could be the result of memory requests served by private caches. This is confirmed by the memory breakdown presented in Fig. 17b, which shows that most of the stalls come from L1 cache (68%), although DRAM still has an impact of 20%. Hence, DRAM may be one of the execution bottlenecks. This is also corroborated by memory impact metric (Fig. 18b) which suggests that performance-wise, DRAM has an impact of 50%, contributing to performance degradation and preventing the loop from attaining private caches roofs. Since its memory requests are mostly handled by L1 cache, it is positioned above DRAM, in particular, on top of L3 cache in both models.

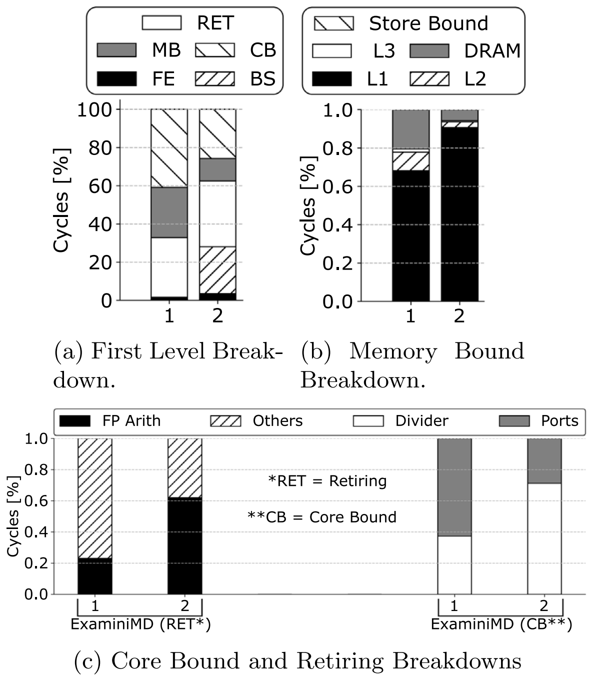
As presented in the Top-Down of ExaminiMD-2 (Fig. 17a), this hotspot is completely dominated by retiring and core bound, with some impact of bad speculation, which might cause performance degradation. The mild memory bound component of this kernel is completely dominated by L1, hinting that this loop is not limited by memory. Since most of its retiring is composed of FP arithmetic (Fig. 17c), this kernel has a strong compute bound nature. This can be visualized in the proposed model, where although the



(a) Intel® Advisor CARM 19.3 - Kernel 1. (b) adCARM - Kernel 1.

(c) Intel® Advisor CARM 19.3 - Kernel 2. (d) adCARM - Kernel 2.

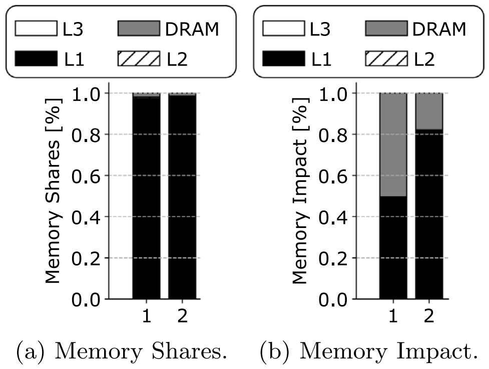
Fig. 16. Characterization of ExaMiniMD kernels in Intel® Advisor CARM 19.3 and adCARM.



(a) First Level Breakdown. (b) Memory Bound Breakdown.

(c) Core Bound and Retiring Breakdowns

Fig. 17. Top-Down for ExaminiMD.



(a) Memory Shares. (b) Memory Impact.

Fig. 18. Memory share and memory impact metrics for ExaminiMD.

kernel gets closer to the memory region, it maintains its compute bound nature. While Intel® Advisor CARM 19.3 hints that it is not possible to surpass the Scalar ADD roof, the adCARM shows that the loop has the potential to achieve the FMA Roof (it contains some DP Scalar FMA instructions), although it needs to transverse the private caches roofs. This is supported by memory share and memory impact metrics (Figs. 18a and 18b, respectively), that present high contribution from L1 cache. Nevertheless, DRAM has still a small impact on the performance, thus to transverse the private caches roofs it might be necessary to focus the optimization on DRAM traffic. The core bound breakdown (Fig. 17c) also pinpoints the divider arithmetic unit as one of the main sources of stalls that limits application performance and may prevent the loop from achieving the FMA performance.

From the analysis performed with adCARM and metrics, several conclusions can be derived regarding the bottlenecks that may limit application performance in modern systems. For example, for an application with well optimized memory accesses (*e.g.*, ISO-3DFD), even a few DRAM accesses may have a significant "negative impact" on performance. Since DRAM accesses lead to an increased performance degradation, emergent and novel memory technologies may be necessary for memory hierarchy to improve the performance of certain applications. This situation becomes worse when considering that applications may contain diverse types of instructions from different ISA extensions and data sizes. As it is shown in the proposed adCARMs, this can result in the reduction of the maximum attainable performance of the applications in all CARM regions. Hence, to fully exploit the potential of modern systems, algorithm design and prototyping need to be carefully performed in order to reduce the impact of instructions that may prevent the application from reaching the absolute performance maximums as offered by the architecture.

6. Related work

Currently, roofline modeling panorama is dominated by three distinct approaches, namely: Cache-Aware Roofline Model (CARM) [8], Original Roofline Model (ORM) [18,19] and Integrated Roofline Model (IRM) [20]. Although all three modeling approaches relate performance with arithmetic intensity (AI), ORM AI and IRM/CARM AI are distinct, since these methods observe memory traffic differently (as referred in Section 2). The work presented in [32] extends ORM memory region by decoupling the total amount of bytes in store and load operations, to provide insights about which memory operation is the main bottleneck. The extended model contains two DRAM roofs (one for stores and one for loads) and the applications are represented by two AIs — store AI and load AI. However, this method does not decouple the time relative to only loads and only store operations, which may limit its application characterization capabilities. On the other hand, the proposed adCARM considers application specifics and memory ports utilization by evaluating the load/store ratio of the application to define tighter performance upper-bounds that meet application requirements. Besides, differently from CARM, ORM memory region cannot encapsulate different instruction set extensions and data precision, since the bandwidth between memory levels is not affected by these parameters. Furthermore, regarding FP peak performance, the work presented in [32] only considers the peak performance for sequential and parallel codes, whereas the adCARM considers different flavors of FP instructions, independently of the number of cores.

A set of ORM extensions are also presented in [33]. Through simulation methods and a performance analysis based on directed acyclic graphs, the model proposed in [33] considers the throughput of several micro-architectural components, such as reservation station and reorder buffer. However, since this model is mainly constructed based on the simulations, it may have a limited adaptability in general application optimization frameworks (*e.g., Intel® Advisor*). Moreover, the high amount of considered micro-architecture bottlenecks may also impact the model insightfulness, which is one of the main strengths of the roofline models. On the other hand, the adCARM evaluates application specifics through source-code/assembly analysis, maintaining model simplicity while improving its usability.

Execution cache-memory performance model (ECM) [34] takes a similar approach to roofline modeling as adopted in CARM, by considering that (to some extent) computations and memory transfers are overlapped. It also considers the amount of loads and stores performed by each application, however this methodology is only applied to AVX instructions. On the other hand, the adCARM reflects different load/store ratios when performing instructions from the different instruction set extensions. Besides, differently from the adCARM, which main goal is to provide first order insights regarding application bottlenecks and optimization, ECM aims at predicting application performance and its usability is limited to memory bound applications, whereas adCARM characterizes memory and compute bound applications.

The work presented in [35] takes a different perspective for application analysis, by evaluating the performance portability across different architectures for several applications, parallel programming models and scientific libraries. Although performance portability is not the main focus of roofline approaches, these methods can be used to evaluate the different bottlenecks that limit application performance for diverse system configurations. In fact, the performance portability method in [35] is already adapted to ORM [36], demonstrating the usability of roofline approaches to evaluate application performance when changing the configuration of the underlying hardware. Since the adCARM proposed in this paper provides a more accurate application characterization, this method can be also used to obtain accurate hints regarding application execution in different systems. Besides, the adCARM can also provide insights about application bottlenecks for future memory configurations, e.g., when used together with Cache Simulation Tool in Intel® Advisor. In this scope, an ORM-based methodology to predict the performance for a future processor is proposed in [37], showing that the roofline methods can be used with this aim.

Moreover, the work in [38] presents a fine-grained profiling with low overhead, aiming at predicting the performance of MPI applications before their execution. Although the adCARM uses online measurements to obtain application performance, prediction models to estimate application performance could be utilized together with adCARM, providing more accurate information about the bottlenecks that limit application performance without running it on the system. This would allow software developers to select the best optimization techniques before deploying the workloads in the real systems.

CARM is also extended to different architectures in several state-of-the-art works. The works [39,40] extend CARM usability to non-uniform memory access (NUMA) computing systems, by considering the bandwidth of remote memories in several different conditions (*e.g.* congested and contented accesses) and also different memory types (*e.g.* MCDRAM). In [41], the CARM approach is applied to NVIDIA graphic units. Since the herein adCARM is based on assembly-code analysis to identify application requirements, it is general enough to be also applied to the described architectures.

Differently from the Intel® Advisor CARM 19.3, which only considers the absolute maximum of the micro-architectures, the adCARM considers the different application characteristics when constructing memory and compute roofs. This work can be used to improve Intel® Advisor CARM implementation, providing to hardware engineers and software developers an insightful and more powerful model for bottleneck detection. In fact, since the adCARM uses assembly analysis to obtain application instruction mix, which is already partially implemented in Intel® Advisor (code analytics tab), it is a feasible method to extend Intel® Advisor CARM implementation without losing simplicity.

7. Conclusions

The continuous micro-architectural improvements have lead to an increase in the complexity of the underlying hardware of modern systems. Future exascale systems are expected to involve thousands of cores and advanced technologies in different hardware subsystems, making it difficult for application designers to attain the maximum performance offered by these machines. For this reason, powerful and insightful methods, such as CARM, are crucial to provide a fast analysis regarding application bottlenecks and optimization hints. However, since current state-of-the-art approaches for roofline modeling only consider the absolute maximum performance of the micro-architecture, these methods do not provide enough accuracy for relating hardware capabilities and the ability of the applications to explore them.

To close this gap, this paper proposes an application-driven CARM for precise performance modeling, in order to improve model insightfulness and usability when characterizing real-world applications, as well as to ease the application optimization process. With this aim, a set of assembly micro-benchmarks was carefully constructed to uncover the upper-bounds of an Intel Xeon Gold 6140 processor for different load/store ratios, memory levels and FP instructions and ISA extensions. The benchmarking results have shown that these parameters have a huge effect on sustainable bandwidth of different levels of the memory hierarchy, as well as on the FP performance. Since modern applications use different types of instructions, it is essential that CARM is able to adapt to their specifics, in order to provide an accurate characterization.

The adCARM proposed in this paper, together with the metrics used to complement CARM analysis, has provided a more precise application characterization, when compared to the SoA

CARM, for a set of 13 kernels from the ECP Benchmark Suite (Sw4lite, MiniQMC and ExaminiMD). The insights derived from the proposed model clearly correlate with the execution bottlenecks obtained from an analysis tool contained in Intel® VTune™ Amplifier (Top-Down), thus showing that the adCARM offers a significant improvement over the state-of-the-art roofline models. Besides, through an in-depth characterization of a finite difference method application (ISO-3DFD), it was possible to verify that the characterization of real-world workloads benefits from considering application specifics for micro-architecture modeling, leading to more accurate conclusions in what concerns the execution bottlenecks that limit application performance, when compared with existing roofline methodologies. As future work, it is expected to include information related to memory latency, diverse memory access patterns and to consider the impact of high latency instructions in the compute roofs, such as divisions and square-roots, which can also impact the application performance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by Intel Corporation and Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT), under grants SFRH/BD/136053/2018, UIDB/50021/2020 and PTDC/CCI-COM/31901/2017 (LISBOA-01-0145-FEDER-031901).

References

[1] Top 500, The list, 2020, https://www.top500.org/lists/2019/11/ [Online; visited January-2020].
[2] ECP 2019, Application development report, 2019, https://exascaleproject.org/wp-content/uploads/2019/11/ECP_AD_update_2019_11_25_spreads.pdf [Online; visited November-2019].
[3] NERSC, Cori configuration, 2019, https://docs.nersc.gov/systems/cori/ [Online; visited November-2019].
[4] H. Schmit, R. Huang, Dissecting Xeon+FPGA: Why the integration of CPUs and FPGAs makes a power difference for the datacenter, in: Proceedings of the International Symposium on Low Power Electronics and Design, ACM, 2016, pp. 152–153.
[5] J. Doweck, W.-F. Kao, A.K.-y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, A. Yoaz, Inside 6th-generation Intel core: New microarchitecture code-named Skylake, IEEE Micro 37 (2) (2017) 52–62.
[6] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, et al., The gem5 Simulator, ACM SIGARCH Comput. Archit. News 39 (2) (2011) 1–7.
[7] A. Yasin, A Top-Down method for Performance Analysis and Counters Architecture, in: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, IEEE, 2014, pp. 35–44.
[8] A. Ilic, F. Pratas, L. Sousa, Cache-aware Roofline model: Upgrading the loft, IEEE Comput. Archit. Lett. 13 (1) (2013) 21–24.
[9] A. Ilic, F. Pratas, L. Sousa, Beyond the Roofline: Cache-Aware Power and Energy-Efficiency Modeling for Multi-cores, IEEE Trans. Comput. 66 (1) (2016) 52–58.
[10] A. Shinsel, Intel Advisor Roofline, 2017, https://software.intel.com/en-us/articles/intel-advisor-roofline [Online; posted 02-March-2017].
[11] J.-S. Park, H.-E. Kim, H.-Y. Kim, J. Lee, L.-S. Kim, A vision processor with a unified interest-point detection and matching hardware for accelerating a stereo-matching algorithm, IEEE Trans. Circuits Syst. Video Technol. 26 (12) (2015) 2328–2343.
[12] S. Titarenko, M. Hildyard, Hybrid multicore/vectorisation technique applied to the elastic wave equation on a staggered grid, Comput. Phys. Comm. 216 (2017) 53–62.
[13] T. Koskela, J. Deslippe, B. Friesen, K. Raman, Fusion PIC Code Performance Analysis on The Cori KNL System, in: Cray User Group Conference, 2017.
[14] A. Mathuriya, Y. Luo, R.C. Clay III, A. Benali, L. Shulenburger, J. Kim, Embracing a new era of highly efficient and productive quantum Monte Carlo simulations, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, pp. 1–12.
[15] E. Serrano, A. Ilic, L. Sousa, J. Garcia-Blas, J. Carretero, Cache-Aware Roofline Model and Medical Image Processing Optimizations in GPUs, in: Proceedings of the International Conference on High Performance Computing, Springer, 2018, pp. 509–526.
[16] V. Etienne, T. Tonellot, K. Akbudak, H. Ltaief, S. Kortas, T. Malas, P. Thierry, D. Keyes, Optimization of finite-difference kernels on multi-core architectures for seismic applications, in: Intel EXtreme Performance Users Group, 2018.
[17] D. Marques, H. Duarte, A. Ilic, L. Sousa, R. Belenov, P. Thierry, Z.A. Matveev, Performance analysis with Cache-Aware Roofline Model in Intel Advisor, in: Proceedings of the International Conference on High Performance Computing & Simulation, IEEE, 2017, pp. 898–907.
[18] S. Williams, A. Waterman, D. Patterson, Roofline: an insightful visual performance model for multicore architectures, Commun. ACM 52 (4) (2009) 65–76.
[19] D. Doerfler, J. Deslippe, S. Williams, L. Oliker, B. Cook, T. Kurth, M. Lobet, T. Malas, J.-L. Vay, H. Vincenti, Applying the roofline performance model to the Intel Xeon Phi Knights Landing processor, in: Proceedings of the International Conference on High Performance Computing, Springer, 2016, pp. 339–353.
[20] T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, et al., A novel multi-level integrated roofline model approach for performance characterization, in: Proceedings of the International Conference on High Performance Computing, Springer, 2018, pp. 226–245.
[21] Intel Corporation, Intel® 64 and IA-32 Architectures Software Developer Manual, 2013, [Online].
[22] I. Cutress, Intel's Architecture Day 2018: The Future of Core, Intel GPUs, 10nm, and Hybrid x86 - Sunny Cove Microarchitecture: A Peek At the Back End, 2019, https://www.anandtech.com/show/13699/intel-architecture-day-2018-core-future-hybrid-x86/2 [Online; visited March-2019].
[23] A. Danalis, G. Marin, C. McCurdy, J.S. Meredith, P.C. Roth, K. Spafford, V. Tipparaju, J.S. Vetter, The scalable heterogeneous computing (SHOC) benchmark suite, in: Proceedings of the Workshop on General-Purpose Computation on Graphics Processing Units, ACM, 2010, pp. 63–74.
[24] D. Bruening, Q. Zhao, S. Amarasinghe, Transparent dynamic instrumentation, ACM SIGPLAN Not. 47 (7) (2012) 133–144.
[25] D.J. Kuck, Computational capacity-based codesign of computer systems, in: High-Performance Scientific Computing, Springer, 2012, pp. 45–73.
[26] W. Jalby, D.C. Wong, D.J. Kuck, J.-T. Acquaviva, J.-C. Beyler, Measuring computer performance, in: High-Performance Scientific Computing, Springer, 2012, pp. 75–95.
[27] C. Andreolli, Eight Optimizations for 3-Dimensional Finite Difference (3DFD) Code with an Isotropic (ISO), 2017, https://software.intel.com/en-us/articles/eight-optimizations-for-3-dimensional-finite-difference-3dfd-code-with-an-isotropic-iso [Online; posted 02-March-2017].
[28] C. Andreolli, P. Thierry, L. Borges, G. Skinner, C. Yount, J. Reinders, J. Jeffers, Characterization and optimization methodology applied to stencil computations, in: High Performance Parallelism Pearls, Morgan Kaufmann Walthan, MA, 2015, pp. 377–396.
[29] ECP, Proxy Apps Suite, 2019, https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/ [Online; visited November-2019].
[30] A. Tal, Intel® Software Development Emulator, 2019, https://software.intel.com/en-us/articles/intel-software-development-emulator [Online; visited November-2019].
[31] R. Karthik, Calculating "FLOP" using Intel® Software Development Emulator (Intel® SDE), 2019, https://software.intel.com/en-us/articles/calculating-flop-using-intel-software-development-emulator-intel-sde [Online; visited November-2019].
[32] G. Ofenbeck, R. Steinmann, V. Caparros, D.G. Spampinato, M. Püschel, Applying the roofline model, in: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, IEEE, 2014, pp. 76–85.
[33] V.C. Cabezas, M. Püschel, Extending the roofline model: Bottleneck analysis with microarchitectural constraints, in: Proceedings of the IEEE International Symposium on Workload Characterization, IEEE, 2014, pp. 222–231.
[34] J. Hofmann, J. Eitzinger, D. Fey, Execution-cache-memory performance model: Introduction and validation, 2015, arXiv preprint arXiv:1509.03118.
[35] S.J. Pennycook, J.D. Sewall, V.W. Lee, Implications of a metric for performance portability, Future Gener. Comput. Syst. 92 (2019) 947–958.

[36] C. Yang, R. Gayatri, T. Kurth, P. Basu, Z. Ronaghi, A. Adetokunbo, B. Friesen, B. Cook, D. Doerfler, L. Oliker, et al., An empirical roofline methodology for quantitatively assessing performance portability, in: Proceedings of the IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC, IEEE, 2018, pp. 14–23.

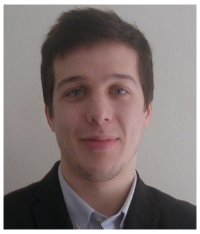
[37] J. Kwack, G. Arnold, C. Mendes, G.H. Bauer, Roofline analysis with Cray performance analysis tools (CrayPat) and roofline-based performance projections for a future architecture, Concurr. Comput.: Pract. Exper. 31 (16) (2019) e4963.

[38] G. Lu, W. Zhang, H. He, L.T. Yang, Performance modeling for MPI applications with low overhead fine-grained profiling, Future Gener. Comput. Syst. 90 (2019) 317–326.

[39] N. Denoyelle, B. Goglin, A. Ilic, E. Jeannot, L. Sousa, Modeling large compute nodes with heterogeneous memories with Cache-Aware Roofline Model, in: Proceedings of the International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Springer, 2017, pp. 91–113.

[40] N. Denoyelle, B. Goglin, A. Ilic, E. Jeannot, L. Sousa, Modeling Non-Uniform Memory Access on Large Compute Nodes with the Cache-Aware Roofline Model, IEEE Trans. Parallel Distrib. Syst. 30 (6) (2018) 1374–1389.

[41] A. Lopes, F. Pratas, L. Sousa, A. Ilic, Exploring GPU performance, power and energy-efficiency bounds with Cache-Aware Roofline Modeling, in: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, IEEE, 2017, pp. 259–268.



Diogo Marques is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at the Instituto Superior Técnico (IST), Universidade de Lisboa, Lisbon, Portugal. He is also a member of the SiPS research group at Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His current research interests include insightful modeling of multi-core processors and heterogeneous systems.

Aleksandar Ilic is received the Ph.D. degree in electrical and computer engineering from Instituto Superior Técnico (IST), Universidade de Lisboa, Lisbon, Portugal, in 2014. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, IST, and a Senior Researcher of the Signal Processing Systems Group, Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), Lisbon, Portugal. His research interests include high-performance and energy-efficient computing and modeling on parallel heterogeneous systems. He has contributed to more than 40 papers in international journals and conferences.

Zakhar Matveev received his Ph.D. degree in computer science (computational geometry and computer graphics) from Nizhny Novgorod State University of Architecture and Civil Engineering (Russia). His current focus is a hardware/software co-design, Roofline performance model, memory and vector parallelism optimization tools. The professional interests are in the areas of high performance computing co-design, parallel programming, computer graphics, code modernization and scalable software design. Zakhar is a product architect for Roofline performance methodology automation and is also an active contributor to Xeon Phi performance analysis user groups and conference workshops.

Leonel Sousa received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 1996. He is currently a Full Professor with UL. He is also a Senior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). He has authored or coauthored more than 250 papers appearing in journals and international conferences, and has edited four special issues of international journals. His research interests include VLSI architectures, computer architectures, parallel computing, computer arithmetic, and signal processing systems. Prof. Sousa is a Fellow of the IET, and a Distinguished Scientist of the ACM.