# Modeling Non-Uniform Memory Access on Large Compute Nodes with the Cache-Aware Roofline Model

Nicolas Denoyelle, Brice Goglin, Aleksandar Ilic, Emmanuel Jeannot, Leonel Sousa

# Modeling Non-Uniform Memory Access on Large Compute Nodes with the Cache-Aware Roofline Model

Nicolas Denoyelle*‡, Brice Goglin*, Aleksandar Ilic†, Emmanuel Jeannot*§, Leonel Sousa†
* Inria – Bordeaux - Sud-Ouest, Univ. Bordeaux, France
{nicolas.denoyelle, brice.goglin, emmanuel.jeannot}@inria.fr
† INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
{aleksandar.ilic, leonel.sousa}@inesc-id.pt
‡ Atos Bull Technologies – 38130, Echirolles, France
§ LaBRI - Laboratoire Bordelais de Recherche en Informatique, Inria Bordeaux - Sud-Ouest

**Abstract**—NUMA platforms, emerging memory architectures with on-package high bandwidth memories bring new opportunities and challenges to bridge the gap between computing power and memory performance. Heterogeneous memory machines feature several performance trade-offs, depending on the kind of memory used, when writing or reading it. Finding memory performance upper-bounds subject to such trade-offs aligns with the numerous interests of measuring computing system performance. In particular, representing applications performance with respect to the platform performance bounds has been addressed in the state-of-the-art Cache-Aware Roofline Model (CARM) to troubleshoot performance issues. In this paper, we present a Locality-Aware extension (LARM) of the CARM to model NUMA platforms bottlenecks, such as contention and remote access. On top of this, the new contribution of this paper is the design and validation of a novel hybrid memory bandwidth model. This new hybrid model quantifies the achievable bandwidth upper-bound under above-described trade-offs with less than 3% error. Hence, when comparing applications performance with the maximum attainable performance, software designers can now rely on more accurate information.

**Index Terms**—Roofline Model, Cache-Aware Roofline Model, Heterogeneous Memory, Knights Landing, Skylake, Platform Modeling, Benchmarking, NUMA, Multi-core/single-chip multiprocessors, Shared Memory, Memory Bandwidth

## 1 Introduction

The increasing demands of current applications, both in terms of computation and amount of data to be manipulated, and the modest improvements in the performance of processing cores have led to the development of large multi-core and many-core systems [1]. These platforms embed complex memory hierarchies, spanning from registers to private and shared caches, local main memory, and memory accessed remotely through interconnection networks. In these systems, memory throughput is not uniform, since they embed several kinds of memories and the distance between processor and memories varies. On such Non-Uniform Memory Access (NUMA) architectures, the way data is allocated and accessed has a significant impact on performance [2].

Recently, the latest Intel Xeon Phi processor, code-name Knights Landing (KNL) [3], traces the NUMA roadmap with a processor organized into 4 Sub-NUMA Clusters (SNC-4 mode). Usually, NUMA platforms include several sockets interconnected with processor-specific links (*e.g.* Quick Path Interconnect [4]) or by custom switches, such as SGI NUMAlink or Bull Coherent Switch [5]. However, the KNL interconnects NUMA clusters at the chip scale (through a 2D mesh of up to 36 dual-core tiles). Though the software may see both types of systems as similar homogeneous NUMA trees, the strong architectural differences between NUMA sockets and KNL chips, can impact application performance in different ways and motivate the joint study of both systems.

Additionally, each cluster of the KNL may feature traditional DDR memory as well as 3D-stacked high-bandwidth memory named MCDRAM, which can be used as hardware-managed cache or additional software-managed memory. Managing heterogeneous memories in runtime systems brings another level of complexity and makes performance analysis harder and even more necessary. Hence, being able to understand the impact of the memory hierarchy and core layout on application performance, as well as on attainable performance upper-bounds, is of most importance and interest. This is especially true when modeling the architecture and tuning applications to take advantage of the architecture characteristics in order to improve performance. As a reference point, on-chip bandwidth varies from several orders of magnitude ($\simeq$ 20 times) between the caches closest to the cores and the local main memory (*cf.* Section 4). Moreover, purely remote memory access across a tested QPI link delivers half of the local memory throughput, thus pushing further the data transfer time. Additionally, increasing the number of concurrent requests on a single memory also increases the contention and can slow down the perceived local memory bandwidth down to $\simeq$46% of its maximum value on a tested Broadwell system (*cf.* Section 4). On the KNL system with 64 cores, the same

memory access pattern decreases the bandwidth down to $\simeq 25\%$ of its maximum value (*cf.* Section 5). Hence, it is clear that the memory access pattern has a significant impact on the delivered throughput. Finally, unlike cross QPI bandwidth, our experiments in Section 5 show a nearly uniform bandwidth on the interconnection network of the KNL chip. Therefore, it is obvious that the chip layout has a significant impact on the achievable performance.

Tuning application performance and inferring their ability to fully exploit the capabilities of those complex systems require to model and acquire knowledge about their realistically achievable performance upper-bounds and their individual components (including the different levels of memory hierarchy and the interconnection network). The Cache-Aware Roofline Model [6] (CARM) has been recently proposed –by some of the authors of this paper- as an insightful model and an associated methodology aimed at visually aiding performance characterization and optimization of applications running on systems with a cache memory subsystem. CARM has been integrated by Intel into their proprietary tools, and it is described as "*an incredibly useful diagnosis tool (that can guide the developers in the application optimization process), ensuring that they can squeeze the maximum performance out of their code with minimal time and effort.*"[1] However, the CARM refers to systems based on a single-socket computational node with uniform memory access, which does not exhibit the NUMA effects that can also significantly impact performance.

To address these issues, we have proposed, firstly in our previous contribution [7] extended in this paper, a new methodology to enhance the CARM unsightliness and provide locality hints for application optimization on contemporary large shared memory systems, such as multi-socket NUMA systems and Many Integrated Core processors with heterogeneous memory technologies and multiple hardware configurations [7]. However, we noticed that some highly optimized synthetic benchmarks are not correctly characterized yet by our model. Though they are designed to characterize the system bandwidth limits, they do not reach the bandwidth bounds described by the proposed NUMA extension [7]. This observation suggests that additional system features have to be modeled in order to successfully characterize the system bandwidth. Hence, on top of the pillars of this work, our contribution is the design and validation of a new heterogeneous bandwidth model. This model aims at providing a more realistic upper bound of the achievable memory bandwidth when an application has to use several kinds of memories.

The remainder of this paper is organized as follows: Section 2 provides an in-depth overview of the Cache Aware Roofline Model to make the model usable for NUMA and KNL architectures. Our first contribution, Section 3 deep dives into the methodology to measure performance upper-bounds for those systems. Our next contribution, in Sections 4 and 5, details the Locality Aware Roofline Model (LARM) construction and validation for a Xeon E5-2650L v4 NUMA system composed of 4 NUMA nodes and the latest Xeon Phi many-core processor. Our last contribution, in Section 6, discusses the main limitations of the model, instantiates and validates a heterogeneous bandwidth model, able to overcome such limitations. Section 7 gives an overview of the state-of-the-art related works, while Section 8 draws the final conclusion of this research work.

1. Intel® Advisor Roofline - 2017-05-12: https://software.intel.com/en-us/articles/intel-advisor-roofline

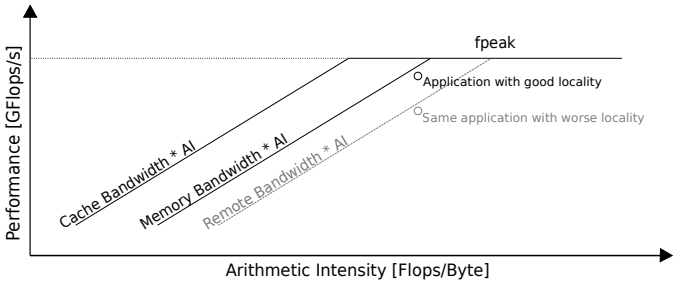



Fig. 1: CARM chart of a hypothetical compute node composed of one cache level.

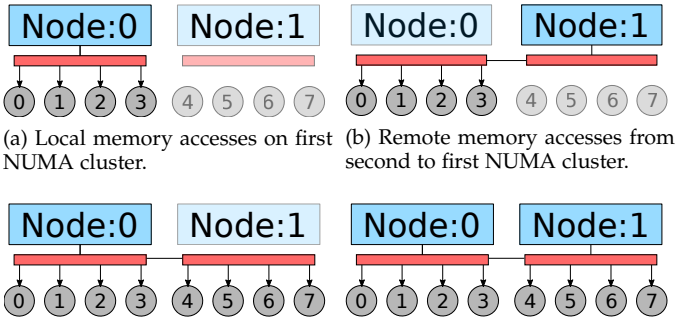
## 2 Locality Aware Roofline Modeling

In general, the Roofline modeling [8] is an insightful approach to represent the performance upper-bounds of a processor micro-architecture. Since computations and memory transfers are simultaneously performed, this model is based on the assumption that the overall execution time can be limited either by the time to perform computations or by the time to transfer data. Hence, from the micro-architecture perspective, the overall performance can be limited either by the peak performance of the computational units or by the capabilities of the memory system (*i.e.* bandwidth).

To model the performance limits of contemporary multi-core systems, the Cache-Aware Roofline Model (CARM) [6] explicitly considers both the throughput of computational engine and the realistically achievable bandwidth of each memory hierarchy level[2]. With this purpose, the CARM (see Figure 1) includes several lines representing the system upper-bounds (Roofs). Oblique lines (representing the memory bandwidths) cross the horizontal lines (representing the peak compute performance), bounding the ridge of memory and compute regions. Applications are characterized according to their features and achieved performance in one of these two regions. The CARM introduces a detailed and meticulous methodology for benchmarking platforms, which is inherited and extended in this paper to NUMA systems.

In contrast to the other roofline approaches [8], the CARM perceives the computations and memory transfers from a consistent micro-architecture point of view, *i.e.* the cores that issue instructions. Hence, when characterizing the applications, the CARM relies on the performance (in GFlop/s) and the Arithmetic Intensity (AI), *i.e.* the ratio of performed compute operations (flops) over the total volume of requested data (in bytes). The CARM is presented in the log-log scale, where the x-axis refers to the AI (in flops/byte) and the y-axis to the performance (in GFlop/s).

2. main memory and several cache levels.




(a) Local memory accesses on first NUMA cluster.

(b) Remote memory accesses from second to first NUMA cluster.

(c) One-to-all: Each core accesses simultaneously a single NUMA node. Contended bandwidth of first NUMA cluster is the bandwidth seen by cores

(d) All-to-all: Each core accesses simultaneously all NUMA nodes. Congested bandwidth of first NUMA cluster is the bandwidth seen by cores 0-3.

Fig. 2: Modeled memory access patterns on a system featuring two memories in blue, Node:0 and Node:1, interconnected by the chips memory controller in red.

### Proposed NUMA Layer Extension to the CARM

From the application perspective, the memory of modern computing systems is abstracted as a flat address space. However, it is made of heterogeneous and/or remote physical memories. In order to fully exploit those system capabilities, current software interfaces [9] [10] [11] require an explicit data allocation policy and/or thread binding policy, in order to reach good performance [12] [13].

Figure 2 depicts such a system, including two sockets with their local memory (also named NUMA node) and a set of cores. On these systems, the bandwidth is not uniform across the network, and it influences memory access performance. Hence, when modeling, the source and destination of memory access (*i.e.* from a core to a NUMA node, *e.g.* local access as in Figure 2a or remote access as in Figure 2b) should be taken into account to analyze application performance. Moreover, such large-scale systems contain a high number of cores whose pressure on NUMA nodes can cause data accesses to be serialized, for example when several of them are accessing a single memory. We qualify this situation as Contention, and is depicted it in Figure 2c. Finally, the network connecting the NUMA nodes to the cores can be subject to Congestion, when several data paths from the cores to the memory, cross the same link. In Figure 2d, we consider the case where data is distributed across all memories. Although there is no contention, each core will access data located over the whole system NUMA nodes and will eventually create Congestion because of the interleaved data paths. In the remainder of this paper, we use the term Cluster to refer to a set of neighbor cores and their local NUMA node(s)[3].

CARM metrics are consistent across the whole memory hierarchy of a single cluster (as illustrated for the first cluster in Figure 2a). However, from the core perspective, memory access performance is not consistent across the system: bytes transferred from one NUMA node to a cluster are not transferred at the same speed to other clusters. This implies that the legacy CARM can only handle a single multi-core cluster, failing to characterize accurately the cases illustrated in Figures 2b-2d. Yet, as Figure 1 shows, without proper (here remote) bandwidth representation in the CARM, locality issues are not obvious since the performance loss can come from many different sources: no vectorization, sparse memory access, *etc.*

To tackle this issue, we have proposed (in our previous contribution [7]) to extend the CARM with the Locality-Aware Roofline Model (LARM), providing the lacking NUMA insights represented in Figure 2. It expresses the three main throughput bottlenecks, as the characteristics of this type of hardware: non-uniform network bandwidth (Figure 2b), node contention (Figure 2c) and network congestion (Figure 2d). The locality aware extension focuses on each cluster perspective, while new bottlenecks are characterized by different memory access patterns. For each cluster, local and remote bandwidths are measured using the cluster alone. More intricate aspects of the interconnection network are characterized using all the clusters simultaneously and with the usual memory allocation policies from the perspective of each cluster. As a result, the LARM representation is a set of CARM charts, (*i.e.* one per cluster of cores) representing the above-described bottlenecks. The remote roofs set the reference upper-bound of achievable bandwidth from remote nodes to a cluster. The congestion roof sets the bandwidth achieved by a cluster when all cores are accessing simultaneously memory regions located across all NUMA nodes in a round-robin fashion. Finally, the contention roof characterizes a cluster granted bandwidth when all system cores are accessing simultaneously a single NUMA node.

In this paper, we propose a new contribution to simplify the representation and provide additional insights when a given application accesses several different memories simultaneously. In Section 6, we replace the local and remote roofs with a novel heterogeneous bandwidth model. This new roof is application specific and accounts for the application locality pattern to derive the maximum achievable bandwidth with an equivalent pattern. Not only does it simplify the model representation and interpretation, but it also overcomes a limitation of the LARM, which presents strictly hardware bottlenecks that do not necessarily match the application performance when several memories and bandwidths are involved.

To the best of our knowledge, there is no work using the CARM to characterize NUMA platforms. Hence, besides the contribution of extending the CARM, we present the following work and results in the remainder of this paper: 1) a tool based on CARM methodology and add-ons to automatically instantiate and validate the model on multi-socket systems and Knights Landing (KNL) Xeon Phi (for various memory configurations); 2) through several validation stages LARM model has been validated for both systems; 3) after observing model issues to match some synthetic benchmark's performance with the system roofs, we propose a heterogeneous bandwidth model able to overcome those issues.

3. On usual platforms, a cluster is identical to the widely used definition of a NUMA node. On KNL, there can exist two local NUMA nodes near each core (DDR and MCDRAM), hence two NUMA nodes per cluster.

## 3 Methodology for Memory and Micro-Architecture Throughput Evaluation

Initially, Cache-Aware Roofline Model [6] is built with two main sets of parameters: micro-architecture instruction throughput and attainable memory bandwidth. The former provides the peak floating point performance while the latter is used to construct a set of local memory roofs (*i.e.* L1, L2, L3, local DRAM bandwidths). LARM details per NUMA Cluster performance and provides additional rooflines. The identification of available Clusters, cores and memory layout required to build the CARM is performed with hwloc library [10]. The hierarchical representation of the machine allows us to enrich the CARM with a per Cluster representation of the system performance bounds, including the proposed NUMA roofs.

The performance of a micro-architecture, in terms of throughput, can be obtained either by relying on the theoretical hardware characteristics, or by experimentally benchmarking the micro architecture. In the former case, the peak floating point performance can be computed as:

$$\underbrace{Fpeak}_{GFlop/s} = \underbrace{Throughput}_{Instructions/Cycle} * \frac{Flops}{Instruction} * N * \underbrace{Frequency}_{GHz}, \quad (1)$$

where $Throughput$ is the number floating point instructions retired per cycle by one core, $Flops/instruction$ is the number of floating point operations performed in each instruction (*e.g.* 2 for `FMA` instruction and 1 for `ADD` instruction), and $N$ is the number of cores. Similarly, the peak bandwidth of the Level 1 cache can be computed as:

$$\underbrace{Bandwidth}_{GByte/s} = \underbrace{Throughput}_{Instructions/Cycle} * \frac{Bytes}{Instruction} * N * \underbrace{Frequency}_{GHz}. \quad (2)$$

Sometimes, the theoretical throughput is not disclosed by the manufacturer or does not match the experimental throughput. Therefore, we use the prior CARM methodology [6] to implement highly optimized micro-benchmarks and build the roofs.

Our methodology for NUMA-specific bandwidth evaluation relies on a hierarchical description of the system topology as provided by the hwloc library. We focus on deep and heterogeneous memory level evaluation, rather than on micro-architectures tightly coupled with caches already analyzed in the original CARM paper [6]. Since the model needs to provide insights on possible bottlenecks of NUMA systems, the model includes the bandwidth roofs described in Section 2 and depicted in Figure 2, *i.e.* local accesses, remote accesses, accesses with congestion and accesses with contention. In order to model local and remote bandwidths of a cluster, we designed a benchmark performing contiguous and continuous memory accesses, as in the CARM, but on each NUMA node individually. One thread is spawned per core of the target cluster, then for each roof (*i.e.* local and remote), the workload is iteratively allocated on each NUMA node, as depicted in Figures 2a and 2b. We do not look at individual links, but rather at pairs of cores+NUMA node, even though sometimes there are multiple (unknown) hops between clusters. The contended bandwidths are obtained similarly to the local and remote bandwidths, but populating all cores with threads (Figure 2c) instead of just the cores of a single Cluster. Each cluster granted bandwidth is associated with the source contended node to build the contended roofs on each cluster chart. Finally, the congested bandwidth is obtained by performing memory accesses from all the cores, contiguous on the virtual address space, but with pages physically allocated in a round-robin fashion across the system NUMA nodes, and with a private data set for each thread. Once again, the bandwidth perceived by each cluster is modeled as the congested roof in its local CARM. Though we call it congestion, it differs from the standard definition[4]. However, it fits a more practical and easy way to reproduce the memory access pattern, *i.e.* the one implied by using the Linux interleave memory allocation policy.

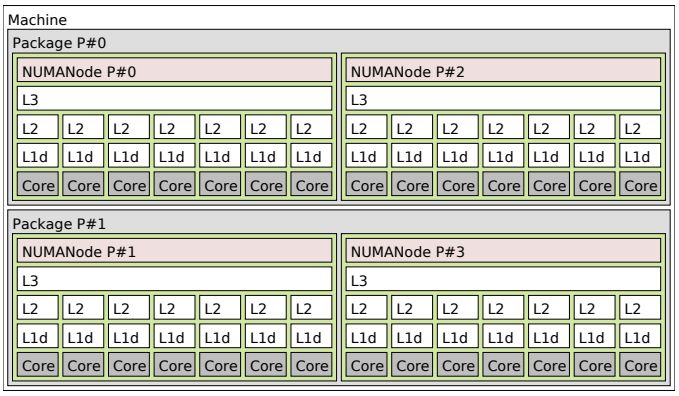



Fig. 3: hwloc topology representation of, Joe0, a dual-socket Xeon E5-2650L v4.

In this paper, we only show the bandwidth for `LOAD` instructions because it suits better our use cases. However, the software tool we have developed is able to evaluate `STORE`, non-temporal `STORE` and a mix of instructions for all memory levels.

## 4 Model Instantiation and Validation on Multi-Socket System

In order to set up and validate the model, we use a dual-socket NUMA system, named Joe0 and presented in Figure 3. It is composed by two Broadwell Xeon E5-2650L v4 processors (at 1.7 GHz), configured with the cluster-on-die mode and exposing the 4 NUMA nodes to the system. Each NUMA node of the system topology (Figure 3) implements 7 cores, with hyper-threading disabled.

### 4.1 Platform Evaluation and Model Instantiation

By relying on the testing methodology proposed in [6], it was possible to reach near theoretical compute and L1 cache bounds on the Intel Broadwell micro-architecture, as presented in Table 1. Each core throughput is derived using the number of operations per instruction and the processor frequency to obtain the peak FMA floating point performance (reaching 190 GFlop/s for a single cluster).

4. Network congestion in data networking and queuing theory is the reduced quality of service that occurs when a network node is carrying more data than it can handle.

| Instruction Throughput | Load | Store | ADD | MUL | FMA |
|---|---|---|---|---|---|
| Theoretical | 2 | 1 | 1 | 2 | 2 |
| Experimental | 1.99 | 0.99 | 0.99 | 1.99 | 1.99 |

TABLE 1: Joe0 core instructions throughput (Instructions/-Cycle)

| Memory Level | Bandwidth (GByte/s) |
|---|---|
| L1 | 760.1 |
| L2 | 309.2 |
| L3 | 154.0 |
| NUMANODE:0 (local) | 36.1 |
| NUMANODE:1 (remote) | 17.5 |
| NUMANODE:2 (remote) | 15.0 |
| NUMANODE:3 (remote) | 14.3 |
| NUMANODE:0 (contended) | 16.7 |
| NUMANODE:1 (contended) | 8.3 |
| NUMANODE:2 (contended) | 6.8 |
| NUMANODE:3 (contended) | 6.2 |
| All NUMANODES (congested) | 18.1 |

TABLE 2: Joe0 bandwidth roofs to the first NUMA cluster, *i.e.* NUMANODE:0.

As discussed in Section 3, a comprehensive evaluation aims to extensively benchmark the memory subsystem with several memory access patterns, *i.e.* the remote/local bandwidth between each pair (cluster, NUMA node), as well as the contended/congested ones. The results obtained are presented in Table 2 for the first NUMA cluster of the system. Unless specified, the model presented herein is restricted to a single NUMA cluster, due to the bandwidth symmetry between clusters.

The performed measures (Tables 1, 2) are used to build the proposed model depicted in Figure 4 for the first cluster of Joe0. Besides the roofs for local caches, this CARM chart also includes all the unveiled memory bandwidths, namely local, remote, contended and congested roofs.

### 4.2 Model Validation

#### 4.2.1 Micro-Benchmarks

This validation step consists in micro-benchmarking the system with several arithmetic intensities, *i.e.* memory and compute instructions in the platform under evaluation. It assesses the ability to reach roofs while performing both computations and memory accesses. The roofs fitness is evaluated through the relative root mean squared error[5] of validation points to the roof performance for a realistic range of arithmetic intensities. Errors and deviations (too small to be visible) for each validation point, and for each bandwidth roof of a single cluster, are presented in Figure 4. As the error in the legend is small (less than 2% on average for every roof), the validation enforces that measured bandwidths are attainable by programs with different arithmetic intensities.

#### 4.2.2 Synthetic Benchmarks

Figure 5 shows the LARM instantiated on the first socket of Joe0. For each NUMA cluster, a CARM chart includes local

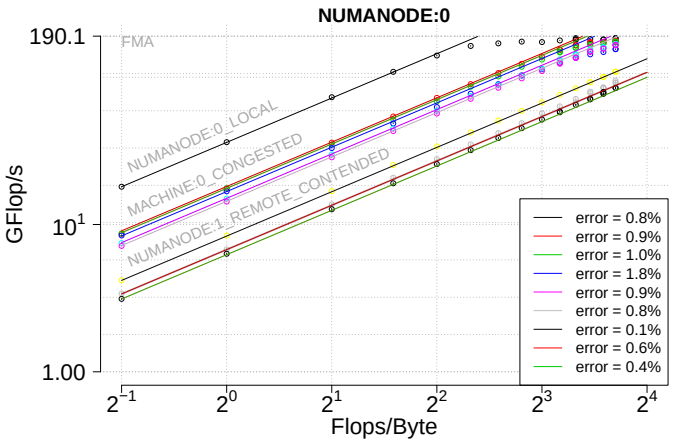


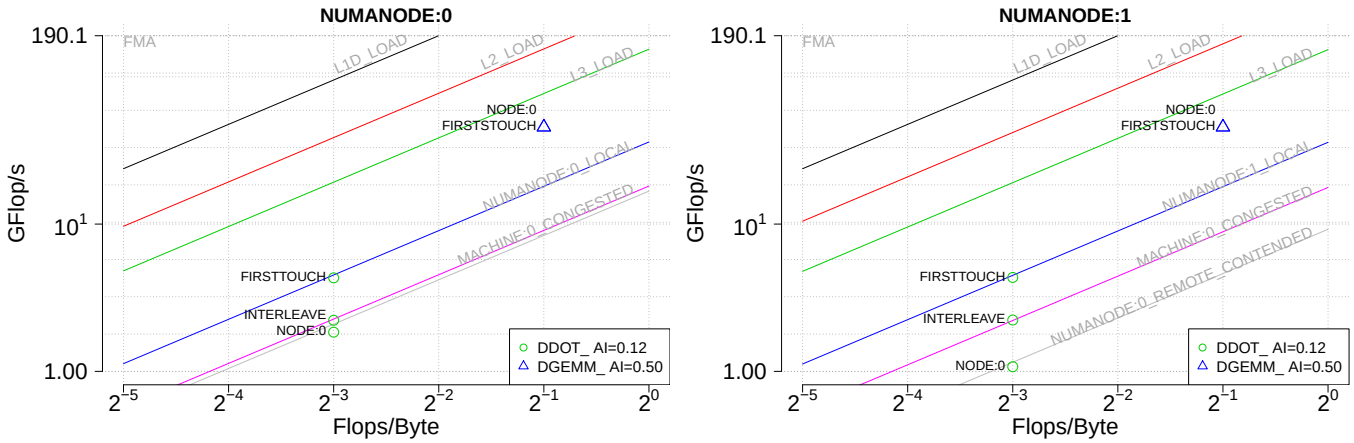

Fig. 4: CARM validation of one NUMA cluster of Joe0 platform. Validation points are visible along the roofs. Finally, the model error for each roof is in the legend.

caches bandwidth, local node bandwidth, the bandwidth under congestion[6] and the bandwidth of the first NUMA node under contention (which is different whether it is measured from the first or the second cluster). Figure 5 also illustrates the memory-bound *ddot* kernel and the compute-bound *dgemm* kernel from the BLAS package, under several scenarios, showing the model ability to pinpoint locality issues. For each scenario, threads are bound in a round-robin fashion and data allocation policy is one of: firsttouch (*i.e.* data in memory close to threads), interleave (*i.e.* data spread on all nodes), Node:0 (*i.e.* data on a single memory node). Each thread performs the same amount of work though the allocation policy on a single node may create an asymmetry when observing their performance across different NUMA nodes (Figure 5). The modeled applications run on the full system, *i.e.* 28 threads (1 thread per core), however, only the model for a single socket is presented to minimize the amount of redundant data. On the chart (Figure 5), *ddot* and *dgemm* are represented each with its own constant arithmetic intensity (*i.e.* the code is unchanged between scenarios) but with different values for runtime parameters.

The *ddot* case with allocation on Node:0 has a different performance whether we look at the first or the second cluster. Hence, the kernel characterization shows the model ability to spot asymmetries. Even if asymmetries do not come from the program, they can come from the data distribution and significantly impact the performance [14]. Congested and contended roofs also successfully characterize similar bottlenecks in *ddot* application. Indeed, in Figure 5, *ddot* kernel with interleaved access (*i.e.* inducing congestion) and access on a single node (*i.e.* inducing contention) match with appropriate memory bandwidths. Optimized compute-intensive applications do not suffer from locality issues. Indeed, as presented in Figure 5, the *dgemm* execution is not affected by non-uniform memory access, it achieves the same performance on each node even if data is allocated with different policies. This may be due to

5. The error is computed as $\frac{100}{n} \times \sqrt{\sum_{i=1..n} \left(\frac{y_i - \hat{y}_i}{\hat{y}_i}\right)^2}\%$ where $y_i$ is the validation point at a given arithmetic intensity, and $\hat{y}_i$ is the corresponding roof.

6. Remote memory bandwidths are very close to congested bandwidths on this system and we omit the former in the chart to improve its readability.




(a) CARM of the first NUMA cluster.

(b) CARM of the second NUMA cluster.

Fig. 5: LARM chart of linear algebra kernels on one socket of Joe0 system.

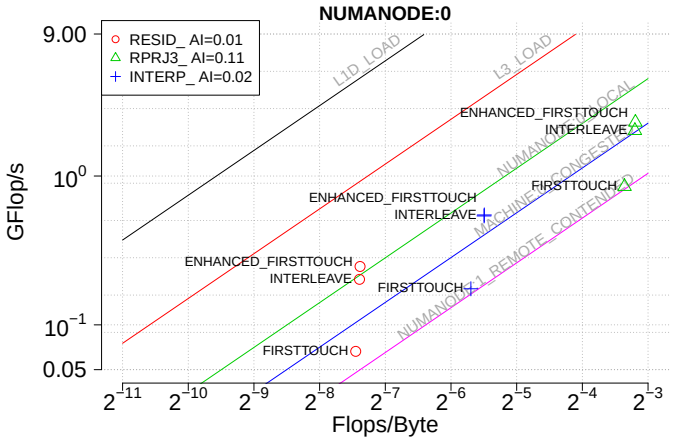



Fig. 6: NAS-3.0 MG Functions Characterization on Joe0 first cluster.

the high cache efficiency of the code allowing the system to prefetch the required data into the cache before it is actually required, thus avoiding the local and remote memory access asymmetries.

In a nutshell, data allocation policies applied to synthetic benchmarks affect the performance in a way that is foreseeable. It matches expectations from their characterization in the LARM, thus validating the proposed roofs relevance.

#### 4.2.3 NAS MG Parallel Benchmark

This step aims to show that the model insights can help to flush out performance bottlenecks, *i.e.* application characterization with the new roofs can help to pinpoint potential execution bottlenecks. For this purpose, we ran a C version[7] of the NAS-3.0 MG benchmark with one thread per core, bound in a round-robin fashion on the system cores. We extract the LARM of this system with hardware counters at the core level, and aggregate the results at the Cluster level. As presented in Figure 6, three functions from the MG benchmark are characterized on the first cluster of the system with several memory allocation strategies. In the first scenario, the default Linux policy firsttouch is used for data allocation. The characterization of these functions (labeled with firsttouch) reaches near the contention roof performance, and suggests the usage of interleave allocation policy to balance memory accesses over the NUMA nodes in order to decrease contention. Indeed, this latter policy improves significantly the performance above the congestion roof. However, it is unlikely that the interleave policy surpasses the firsttouch policy with such significance. Hence this observation also suggests that firsttouch actually allocates memory on a single node. Indeed, once parallelized, the previously sequential memory allocations enable the firsttouch policy to allocate data on all NUMA nodes near appropriate threads. This solution, labeled with enhanced_firsttouch slightly improve against the performance of interleave policy.

To sum up, the LARM characterization of the memory-bound MG benchmark, matches the contended roofs when data allocation is serialized, *i.e.* data is allocated on a single contented node, and improves above the congestion roof once the contention issue is solved, validating hereby the proposed roofs.

## 5 Model Instantiation and Validation on Knights Landing Processor

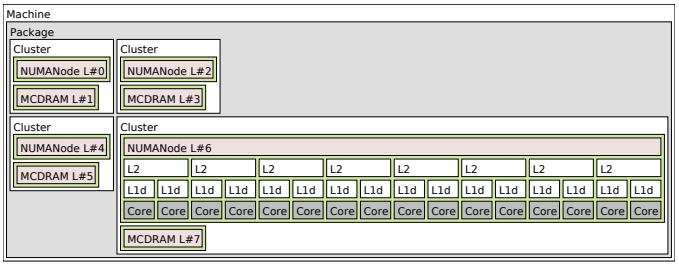



Fig. 7: hwloc model of KNL topology in SNC-4 flat mode. Only the fourth cluster is detailed, for simplifying representation. Other clusters have a similar topology.

7. https://github.com/benchmark-subsetting/NPB3.0-omp-C

When in SNC-4 mode [3], the KNL is a special case of a multi-Socket system. Each socket has an additional fast memory (MCDRAM) to the conventional DRAM memory also specified as NUMA, which is addressable in Flat mode or configurable as a last level cache in Cache mode. Whether flat or cache mode is used, the system may yield different bandwidths, performances and application execution times. The proposed model accounts for this behavior when characterizing KNL platform performance.

For our experiments, we used Knights Landing 7230 chips, with 64 cores operating at 1.3GHz. The KNL topology, when configured with SNC-4 and flat settings, is shown in Figure 7, where the complete topology is provided for the last type of cluster. The mesh interconnection network (Figure 8) between L2 tiles of the chip is widely different from conventional multi-socket systems [15] and motivates additional observations when compared to the previously analyzed system.

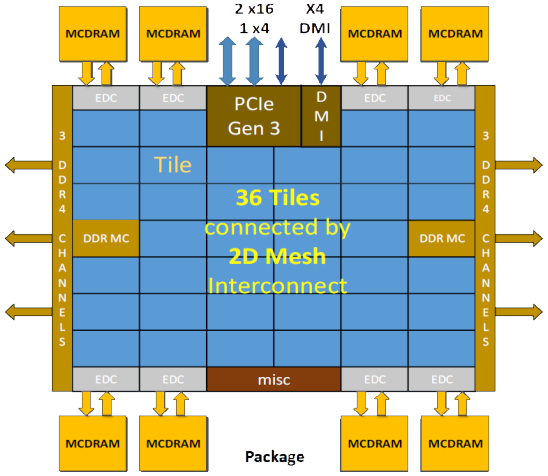



Fig. 8: Knights Landing mesh interconnect with DRAM and MCDRAM memory controllers (Source: Intel). Only 32 of these 38 tiles are actually enabled in our experimentation platform. The number of tiles enabled can reach up to 36 tiles, though 38 are present.

### 5.1 Platform Evaluation and Model Instantiation

By relying on the micro-architecture evaluation methodology from Section 3, the highest experimentally achieved throughput with carefully designed micro-benchmarks is slightly lower than theoretical values (see Table 3). However, a performance of 2.2 TFlop/s for 64 cores is still achieved.

| Instruction | Load | Store | ADD | MUL | FMA |
|---|---|---|---|---|---|
| Theoretical throughput | 2 | 1 | 2 | 2 | 2 |
| Experimental throughput | 1.66 | 0.96 | 1.70 | 1.70 | 1.70 |

TABLE 3: Theoretical and experimental instruction throughput (in instructions per cycle) for a single core of the KNL platform.

Table 4 presents the bandwidth evaluation between clusters solo (*i.e.* by fully exercising memory units within a single cluster) for flat mode only. The evaluation is presented for the first two clusters since the others yield a similar bandwidth. Contrary to the multi-socket system, remote and local DRAM attain similar bandwidths, which suggests the high efficiency of KNL interconnection network. However, significant and less predictable variations can be noticed for MCDRAM, which would require the disclosure of more architectural details to fully explain the mesh interconnection behavior.

| | | from | | | | |
|---|---|---|---|---|---|---|
| | | NUMA:0 | MCDRAM:1 | NUMA:2 | MCDRAM:3 | |
| to | Cluster:0 | 38.1±0.1 | 92.0±0.5 | 38.0±0.8 | 86.6±0.4 | |
| | Cluster:1 | 38.1±0.1 | 91.5±0.4 | 38.2±0.1 | 92.8±0.4 | ... |
| | Cluster:2 | 37.8±0.1 | 90.6±0.5 | 38.1±0.1 | 83.7±0.6 | |
| | Cluster:3 | 38.0±0.2 | 82.8±0.4 | 38.0±0.1 | 90.8±0.3 | |

TABLE 4: KNL load bandwidth (GByte/s) from first and second clusters memories to cores in flat mode. Other clusters are omitted because of similar results.

In Table 5, we also compare the bandwidth for `LOAD` instructions granted to the first cluster when the data set is allocated into the first cluster DRAM and MCDRAM under different scenarios. The first row of the table corresponds to the reference when the cluster runs solo as in Table 4 for the cache and flat modes. In the cache mode, bandwidths of both types of memories (*i.e.* DRAM and MCDRAM) decrease, probably due to overheads induced by the MCDRAM caching mechanism. In both modes, DRAM bandwidth (NUMA:0) reduces when using all clusters simultaneously (*i.e.* local with 64 versus 16 threads), whereas this is less obvious for MCDRAM bandwidth. The presence of only two DRAM memory controllers shared among 4 clusters to access DRAM, whereas there are 8 EDC controllers (two per cluster) to access MCDRAM (see Figure 8), is a possible cause of this behavior.

| | | flat | | cache | | |
|---|---|---|---|---|---|---|
| | | NUMA:0 | MCDRAM:1 | NUMA:0 | MCDRAM:1 | threads |
| Cluster:0 | local | 38.1 ±0.1 | 92.0 ±0.5 | 22.9 ±0.7 | 85.4 ±3.0 | 16 |
| | local | 21.7 ±0.7 | 90.9 ±1.2 | 20.0 ±0.7 | 83.3 ±2.0 | 64 |
| | congested | 19.8 ±0.3 | 77.6 ±2.0 | 17.0 ±0.4 | NA | 64 |
| | contended | 10.7 ±0.0 | 21.5 ±0.5 | NA | NA | 64 |

TABLE 5: KNL load bandwidth (GByte/s) from first cluster memories.

In cache mode, DRAM bandwidth drops when using all clusters simultaneously. However, the fall is not as significant as the drop in flat mode, probably because of data reuse in MCDRAM cache, which redirects a part of the traffic via the EDC channels and absorbs a part of the contention on DRAM memory controllers. Congestion already happens for interleaved memory access on DRAMs, provoking further bandwidth reduction when compared to local memory accesses. As expected, contention is the worst-case scenario, resulting in a dramatic bandwidth reduction. When congestion and contention are experienced, they imply a need for a locality to achieve good performance. Several Non-Achievable values stand in Table 5. One of them, contention on NUMA:0 in cache mode cannot be observed with the Section 3 methodology. Indeed, private data accessed by each cluster in NUMA:0 memory would actually fit into the 4 MCDRAMs and result in MCDRAMs benchmark instead of NUMA:0 benchmark.

Based on the characterization provided above, the LARM is constructed for a single cluster and presented in Figure 9, where the chip is configured in (SNC-4) flat

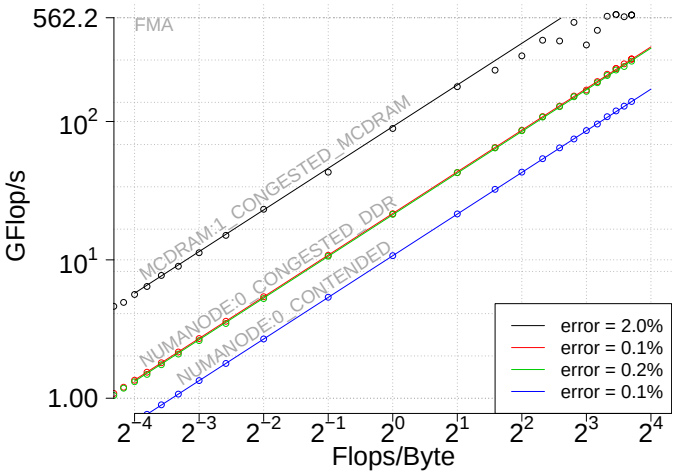
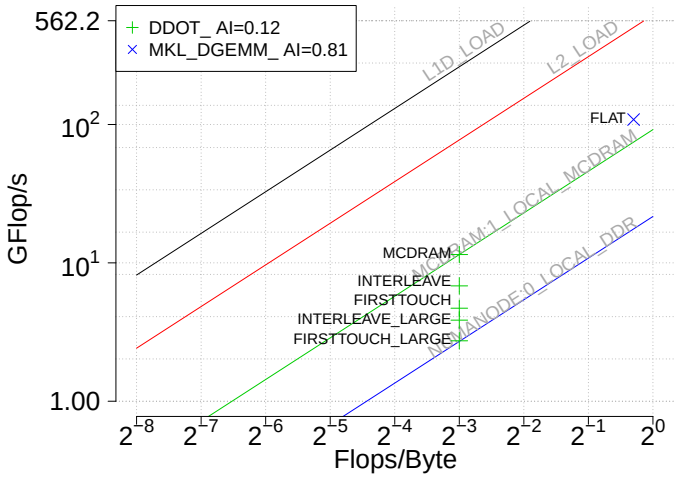
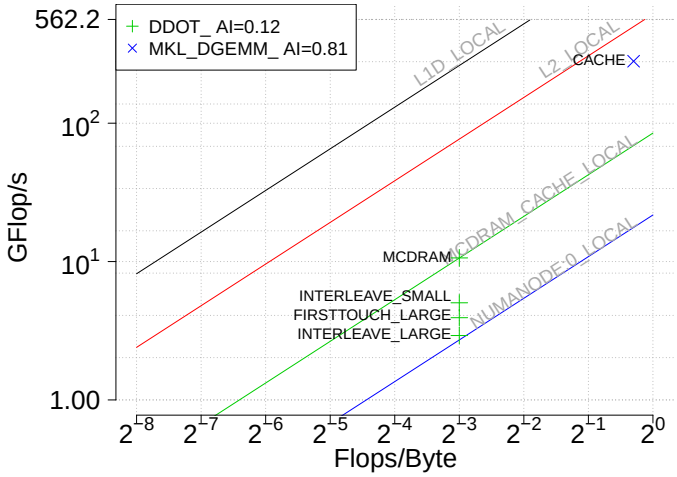



Fig. 9: CARM validation of one sub-NUMA cluster of KNL platform in SNC-4 flat mode.




(a) CARM of the first cluster in flat mode.




(b) CARM of the first cluster in cache mode.

Fig. 10: CARM of KNL first cluster with synthetic benchmarks running either in flat mode or in cache mode.

mode. In contrast to the previous platform, it also includes the MCDRAM roofs siblings of DRAM roofs. Bandwidths of remote nodes are hidden for clarity, because they have the same order of magnitude as local bandwidth and thus overlap in the chart.

## 5.2 Model Validation

### 5.2.1 Micro-Benchmarks

We use again the previous methodology to validate the model in Figure 9 for one cluster (equivalent to the others). KNL micro-benchmark validation fits the model with an average error below 5% for all roofs. Most of these errors are due to the points located near the ridge on L1 and MCDRAM bandwidths roofs. Otherwise, it fits nearly perfectly the roofs in memory-bound and compute-bound regions.

### 5.2.2 Synthetic Benchmarks

As previously referred, validation with synthetic benchmarks aims to verify that well-designed synthetic benchmarks are able to hit the roofs. For this purpose, we characterize again *ddot* and *dgemm* BLAS kernels in the KNL CARM chart (Figure 10) for the first cluster. Herein we focus on MCDRAM usage rather than classical memory allocation policy, which was already analyzed for the multi-socket system. Hence, several data allocation strategies and sizes, as well as the flat and cache configurations of the chip are compared.

The *ddot* function is compared under both flat and cache modes and by adopting various allocation strategies and data-set sizes. The small data set (labeled small in Figure 10 fits into MCDRAM, whereas the large data set (labeled large) does not. MCDRAM policy allocates the whole small data set into MCDRAM. Interleave exploits Linux implementation of the policy, allocating pages of the data set across all the nodes, *i.e.* MCDRAM and DRAM nodes. Finally, the Linux policy firsttouch allocates data on the DRAM near the first thread writing the corresponding page.

In flat mode, allocation into MCDRAM allows for the performance to reach approximately the MCDRAM roof, as visually assessed through the model. Unlike allocations into MCDRAM, large data sets with firsttouch policy are allocated into slow memory and also reach approximately the DRAM roof performance. With interleave policy, the data set is mixed across memories and thus the performance stands in between local DRAM roof and local MCDRAM roof, with a higher influence of the slowest DRAM memory (the point is closer to this roof). In cache mode, the small data set is cached in MCDRAM, thus reaching MCDRAM bandwidth roof. Although MCDRAM bandwidth is lower in cache mode, the performance of large data sets with firsttouch policy is higher than DRAM roof. This is because the large MCDRAM cache size allows reusing a significant part of the data, thus improving achievable performance. Interleaving memory accesses decreases performance when compared to firsttouch policy, because of the congestion hereby generated.

The *dgemm* function uses a data set too large to fit into MCDRAM and is compared in flat and cache mode. As expected, the intrinsic temporal locality of this kernel allows the cache mode to yield a better performance. When choosing the data set size or location (DRAM or MCDRAM),

the synthetic benchmark performance still corresponds to our expectation and validates the model insights on KNL system. When choosing the system configuration (flat or cache), the model also shows that kernels with good data reuse, *i.e.* with a performance over MCDRAM roofs, benefit from the hardware cache mechanism and also validates the model relevance.

#### 5.2.3 Lulesh proxy-applications

For NUMA KNL system, we focus on Lulesh application because of its sensitivity to memory bandwidth. The aim of this validation step is to show that the model helps managing memory, *i.e.* either performance can be improved with the chip configuration or a good memory allocation policy. From Lulesh, we pick the three memory-bound hot spots of the application (*i.e.* the ones bounded below NUMANode:0 roof), namely CalcFBHourGlassForElems, IntegrateStressForElems functions, and a loop in the main function. Due to the lack of required hardware counters, arithmetic intensity, performance and application profile are collected with the Intel Advisor tool[8]. In our experiments, we ran the application using a working set size large enough[9] not to fit into MCDRAM. Hence, target memory needs to be carefully chosen to fulfill size constraints and special care needs to be taken when addressing memory allocation to get good performance.

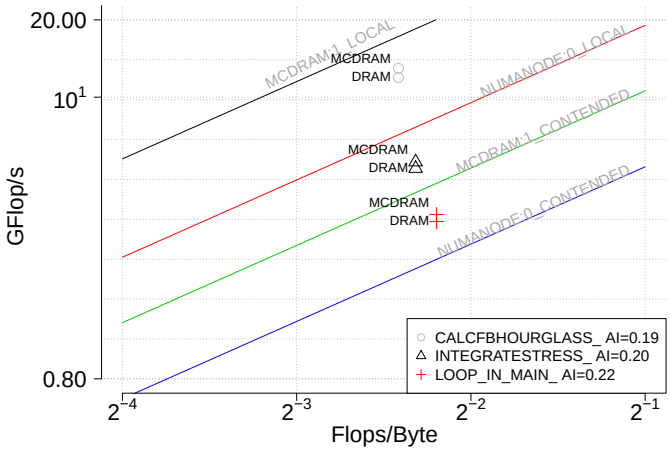
The first run allocates all data into regular DRAM memory (labeled DRAM), and aims at characterizing the application to find potential allocation improvements. We then customize dynamic allocations in those hot spots by replacing the usual allocator from the standard C library with memkind [16] allocator to target fast memory (labeled as MCDRAM in Figure 11a) instead of the traditional DRAM (labeled as DRAM). Finally, instead of forcing MCDRAM allocations, we let the interleave policy (labeled as interleave) to choose which data to put into MCDRAM for all allocations visible in the file lulesh.cc. Summarizing, each hereby found hot spot is executed using three different policies, *i.e.* DRAM allocation (labeled DRAM), custom allocations (labeled MCDRAM), and interleave policy, in flat mode (see Figure 11a).

The second chart in cache mode (see Figure 11b) contrasts the performance of hand-tuned allocations into MCDRAM with hardware management of the fast MCDRAM cache. As expected, MCDRAM allocations provide improved performance in flat mode. However, in cache mode hot spots characterization reaches comparable performance to top achieved performances in flat mode, denoting hardware efficiency to manage data locality. In comparison, interleave memory policy performs poorly, probably because spread allocation forces threads to access remote nodes, thus congesting the mesh.

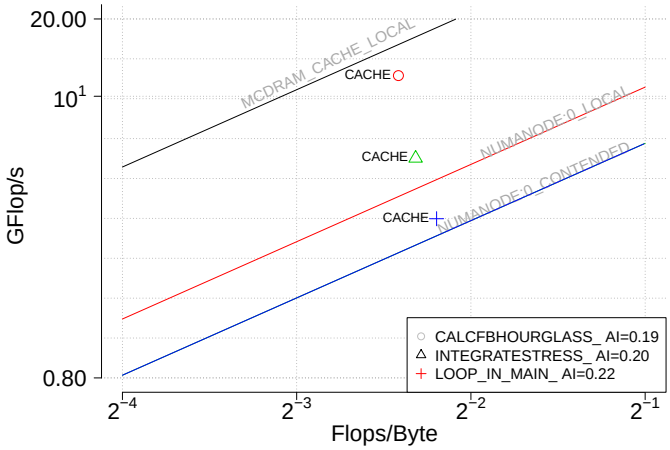
To summarize, the use of the LARM for data allocation policy choice on the KNL, we saw with lulesh case that cache mode brings no significant performance improvement. Hopefully, LARM characterization gave us this insight because it pointed out that the application performed

8. Product version: Update 2 (build 501009).

9. Lulesh application run parameters: -i 1000 -s 60 -r 4. The application is compiled with ICC 17.0.2 and options: -DUSE_MPI=0 -qopenmp -O3 -xHost




(a) KNL first cluster in flat mode.




(b) KNL first cluster in cache mode.

Fig. 11: CARM chart of the KNL first cluster. The 3 main hot spots as detected by Intel Advisor are represented both in cache and flat modes.

below MCDRAM roofs. Nevertheless, data allocation policy changes performance significantly between interleave and other policies. However, the causes of this behavior arises are not clear. In the model, *interleave* points are widely spread on the performance axis. This suggests a different issue than memory allocation policies. Unfortunately, some other bottlenecks than the ones presented here may lower overall performance when changing such a very high-level parameter as the memory allocation policy. In the next section, we explore one among possible causes for this lack of accuracy in the LARM and propose a novel heterogeneous bandwidth model.

## 6 Beyond Memory Bounds: Modeling Heterogeneous Memory Bandwidth.

As shown in previous sections, performance characterization of some applications may not perfectly match the bandwidth upper-bounds defined by the model. When streaming a data set spanning several memories (*e.g.* MCDRAM + DRAM), the maximum achievable bandwidth does not adhere to a single memory bound. The LARM characterizes achievable bounds as one of those memory bounds, while

using several memories simultaneously leads to a different bandwidth upper-bound. In the real world, this scenario might happen for different reasons: using Linux interleave allocation policy, allocating a large data set spanning several memories, or even application-level allocation management as offered by memkind interface [16]. Our proposal is to characterize application physical memory access patterns and infer the maximum achievable bandwidth following this pattern. Therefore, instead of modeling raw memory bandwidth (local and remote), we model a bandwidth parametrized by the locality scheme spanning across several memories. Such hybrid roof is represented in the original LARM (Figure 13) for a KNL platform. In between MCDRAM:1 and NUMANODE:0 load bandwidths, possible positions of hybrid roofs are characterized by a color gradient. This gradient represents the continuity of the achievable bandwidth parametrized by the amount of MCDRAM and NUMANODE:0 accesses.

The main memory subsystem features several bandwidths depending on the memory type and distance to cores, as shown in the LARM [7]. The combination of all those bandwidths is subject to a superposition of different hardware mechanisms involved in serving memory requests. Starting from instruction pipeline, data requests trigger the prefetchers and cache coherency systems, crossing the cache hierarchy, then the chip interconnection network (mesh for KNL, ring for Broadwell, *etc.*), memory controllers, off-chip interconnect and finally memory banks. Every component of the non-exhaustive herein described data path implies complex policies. These may not even be publicly disclosed, which makes hardware-based modeling a hard task. Thus, unlike it is performed in the CARM construction methodology, we apply herein a systemic approach, by considering unknown details of the main memory subsystem as a black box, to model the main memory subsystem (not including the caches). Our goal is to expose a simple model out of significantly complex hardware mechanisms involved.

In this section, we provide a thorough analysis of performance upper-bounds for memory-bound application, which is actually multi-bandwidth-bound. We first show that, in practice, the effective bandwidth exploited by an application can be in between several system memory bandwidths, if data span across several memory levels. We propose to characterize this performance bound with an application specific hybrid roof instead of the previously proposed static NUMA roofs. This bound is assessed through a set of bandwidth benchmarks by considering a wide range of memory access patterns. We show that the herein proposed memory bound can provide a better approximation of an application achievable bandwidth upper-bound when compared with traditional modeling approaches. Finally, the overall goal of designing application specific hybrid roof is achieved. On top of these observations, we characterize achieved bandwidth limits with an insightful model, which quantifies the system ability to overlap some data transfers across several memories.

### 6.1 Motivation: Use Cases

This subsection carries out runs of several STREAM benchmarks with several memory access patterns. It aims to show that highly optimized memory-bound applications may run at a lower bandwidth than raw system bandwidth. Furthermore, the *ddot* synthetic benchmark is chosen for demonstrating effects of using heterogeneous memory bandwidth, while *triad* and *scale* benchmarks are chosen to show variations in memory bandwidth when serving a different amount of load and store requests in main memory. Hence, these applications are good candidates for showcasing benefits of including a hybrid roof in the LARM.

We implemented a custom version of these kernels with ISA specific instructions optimized for AVX512 capable micro-architectures and with non-temporal stores. This implementation is compiler agnostic and ensures to exercise the full potential of tested platforms. A code snippet of critical regions of *ddot* and *triad* are exposed in Figures 12a, 12b.

*ddot* is tailored to approach target memory load bandwidth. However, in certain cases, data accesses can span several memories. For instance, within this application, allocating a large data set into MCDRAM, exceeding its capacity, will result in utilizing the slower DRAM as well. Another way to allocate the memory without explicit awareness of the hybrid memory system is to use an interleave policy. In this case, pages of allocated data are pinned in a round-robin fashion on the system memories. In Figure 13, the LARM performance characterization is presented for the same *ddot* function but with different allocation policies: data into MCDRAM memory (MCDRAM), data into DRAM memory (firsttouch), and data spanning both memories (interleave). As it can be observed, when data is allocated into several memories with different bandwidths, the total aggregated bandwidth does not match any of the system memories bandwidth. *triad* and *scale* benchmarks also exhibit different bandwidths than the system memories bandwidth, even though data is allocated into a single local memory, due to different load/store patterns when compared to *ddot*.

For the aforementioned cases, with the very same arithmetic intensity but varying memory access patterns, the performance was affected even though those synthetic benchmarks are designed to reach memory bandwidth bounds. To some extent, memory access pattern is a constraint that has to be fulfilled, though it is considered as sub-optimal in the LARM. Hence, the herein proposed work aims at enriching the model with a roof accounting for applications specific memory access pattern. The next subsection discusses the construction of such a hybrid roof, along with its integration in the LARM.

### 6.2 Overlapping Memory Bandwidths Model

In its current form, the CARM model is based on the assumption that the performance of an application can be bound either by compute throughput or memory bandwidth depending on the application arithmetic intensity. Every bandwidth associated to cache levels and memory kinds are represented as possible bottlenecks. The LARM splits the system in NUMA domains and stacks on top of each local CARM bandwidths, local and remote bandwidths, bandwidth under congestion and bandwidth under contention. With the new proposed bandwidth model, we remove local and remote bandwidths in the LARM, and replace them with a hybrid roof. For each application,

```
vpxor %%ymm0, %%ymm0, %%ymm0
loop_ddot:
vmovupd (%1), %%zmm1
vmovupd (%2), %%zmm2
vfmadd231pd %%zmm2, %%zmm1, %%zmm0
vmovupd 64(%1), %%zmm3
vmovupd 64(%2), %%zmm4
vfmadd231pd %%zmm4, %%zmm3, %%zmm0
vmovupd 128(%1), %%zmm5
vmovupd 128(%2), %%zmm6
vfmadd231pd %%zmm6, %%zmm5, %%zmm0
vmovupd 192(%1), %%zmm7
vmovupd 192(%2), %%zmm8
vfmadd231pd %%zmm8, %%zmm7, %%zmm0
add $256, %1
add $256, %2
sub $32, %3
jnz loop_ddot
```

(a) *ddot* assembly code snippet.

```
vmovupd (%[s]), %%zmm0
loop_triad:
vmovupd (%[a]), %%zmm1
vmovupd (%[b]), %%zmm2
vfmadd231pd %%zmm0, %%zmm2, %%zmm1
vmovntpd %%zmm1, (%[c])
vmovupd 64(%[a]), %%zmm3
vmovupd 64(%[b]), %%zmm4
vfmadd231pd %%zmm0, %%zmm4, %%zmm3
vmovntpd %%zmm3, 64(%[c])
vmovupd 128(%[a]), %%zmm5
vmovupd 128(%[b]), %%zmm6
vfmadd231pd %%zmm0, %%zmm6, %%zmm5
vmovntpd %%zmm5, 128(%[c])
vmovupd 192(%[a]), %%zmm7
vmovupd 192(%[b]), %%zmm8
vfmadd231pd %%zmm0, %%zmm8, %%zmm7
vmovntpd %%zmm7, 192(%[c])
add $256, %[a]
add $256, %[b]
add $256, %[c]
sub $32, %[n]
jnz loop_triad
```

(b) *triad* assembly code snippet.

Fig. 12: Sample of KNL optimized assembly of *ddot* and *triad* kernels.

we characterize its physical memory access pattern and match the best achievable bandwidth following a similar pattern. The latter comes in replacement of the previous local and remote bandwidths. In Figure 13, we represent the LARM chart of a single NUMA domain of a KNL system, with local MCDRAM and DRAM bandwidths. The range of possible values for herein proposed hybrid roofs are represented as a color gradient spanning from MCDRAM bandwidth to DRAM bandwidth. Projecting this roof for each represented application would correspond to replacing the color gradient with parallel oblique lines crossing each application point, if it is indeed limited by the matching hybrid bandwidth.

When accessing several memories (in the main memory subsystem), a bandwidth-bound application needs to wait

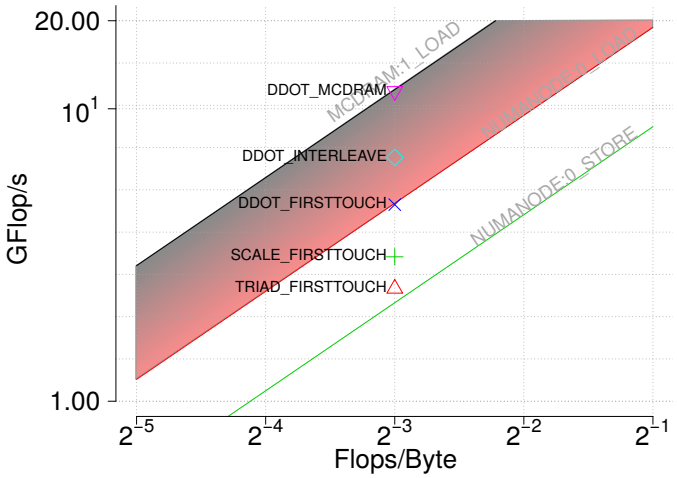



Fig. 13: Several STREAM benchmarks with different load/-store patterns and different memory allocations. Between MCDRAM:1_LOAD and NUMANODE:0_LOAD roofs is represented as a color gradient the possible scalar values for the hybrid roof depending on the physical memory access pattern between each hereby characterized memory.

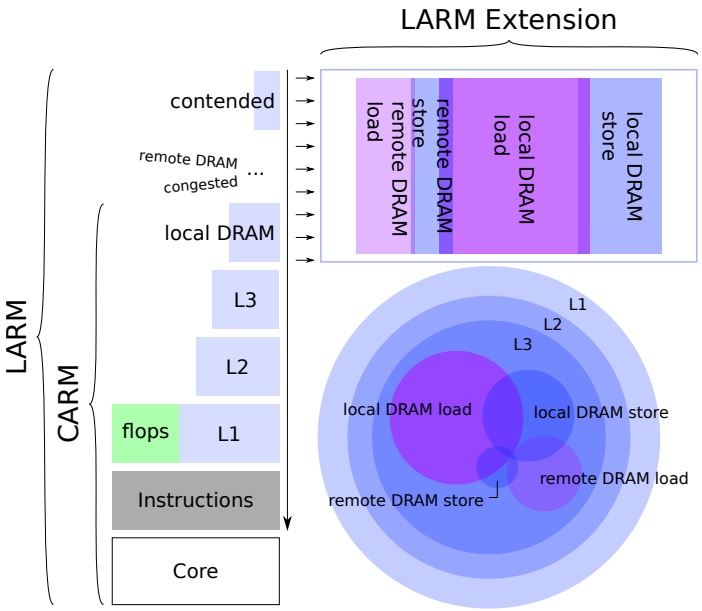



Fig. 14: LARM extension model compared with LARM and CARM models.

(at least) for the slowest data transfer to finish, thus defining an upper bound of the achievable bandwidth. It is also a reasonable hypothesis to state that the application has to wait (at most) the time of sequentially accessing each of those memories, thus setting a lower bound of the achievable bandwidth. As it will be shown in the next subsection, the realistically achievable bandwidth is in between those two bounds. Thus, data transfers can overlap as depicted in Figure 14, and our goal is to quantify this overlap in order to provide a complete characterization of the proposed hybrid roof.

### 6.3 Dissecting Nodes Memory Bandwidth

In order to understand and quantify how the overall application bandwidth is affected by the use of different memories with different bandwidths, we run a large set of micro-benchmarks tailored to reach system bandwidth upper-bounds following the legacy methodology from the CARM [6]. For this evaluation, we use two NUMA memories, and one cluster of cores; the memory with the highest bandwidth is noted as fast and the one with the least bandwidth is noted as slow. For instance, on a Knights Landing processor, the slow memory stands for the DRAM, while the fast memory stands for the MCDRAM. Hence, this methodology remains the same whether it is applied to the KNL system with a fast and a slow memory per cluster of cores, or if it is applied to a single cluster of a multi-socket system with near fast memory and far slow memories. Unlike it was considered previously, the herein proposed approach accounts for the application locality and load/store patterns to infer the achievable bandwidth. This is done by measuring the sustained bandwidth when varying the amount of load and store instructions, as well as the proportion of the data set in fast and slow memories. Obtained benchmark results are represented by a surface, where the bandwidth measured by a cluster of cores is a function of $\frac{\text{load}}{\text{load+store}}$ proportion and $\frac{\text{fast}}{\text{fast+slow}}$ proportion of data accessed. Each benchmark bandwidth can be finally matched with (load, store) and (slow, fast) ratios of an application (obtained with hardware counters) to deduce a new upper-bound of the application achievable bandwidth.

For each $\frac{\text{load}}{\text{load+store}}$ value to test, we generate a highly tuned micro-benchmark to measure the platform bandwidth. As for constructing the CARM roof, the critical section of each benchmark uses the highest available vector extension (*i.e.* AVX512 here) for load and store instructions. Furthermore, since we do not intend to measure cache effects, we bypass the cache hierarchy with the special non-temporal store instructions `VMOVNTPD` . We do not use non-temporal loads `VMOVNTQDA` yielding a similar bandwidth as cache-able loads `VMOVUPD`. Such an instruction flow is depicted on Figure 15. The latter also shows how memory locality is coupled with loads and stores which is detailed below.

Since data allocation spans several memories, we have to set the allocation scheme maximizing the achievable bandwidth. The trade-off is between spatial locality when allocating one continuous block for each memory (or balance) and parallelism when interleaving pages on each memory. It is worth to note that the system prefetcher is able to successfully deal with regularly spaced patterns of interleaved memory allocations and mitigate spatial locality issues. Hence, we choose to use the interleave allocation pattern as illustrated in Figure 15 with a read buffer and a write buffer interleaving pages of slow and fast memories. If the amount of each memory used is not balanced, then, remaining pages which cannot be interleaved are allocated in a single memory. Unlike pages allocation strategy, benchmarks load and store instructions are interleaved in a balanced way to mimic application behaviour. For instance, if there are twice as much load instructions than store instructions, the pattern interleaves two loads with one store as depicted in Figure 15. Indeed, because of the limited amount of registers in the processor, applications usually have to store computations results soon after they are performed in order to release the registers for new data.

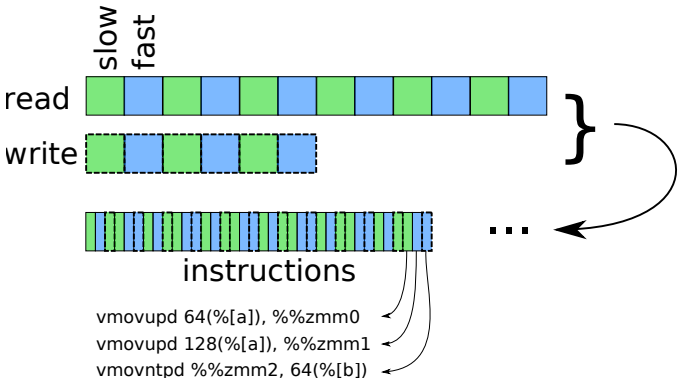



Fig. 15: Example of a memory benchmark pattern where half of buffers size is in the fast memory, *i.e.* slow/fast ratio = 1, and one third of the instructions are store instructions, *i.e.* load/store ratio = 2

### 6.4 Results and Models

We applied the proposed methodology for a wide range of (load, store) and (slow, fast) ratios on two different NUMA systems: the Knights Landing processor and a bi-socket Xeon processor from Skylake generation, in order to evaluate the target memory bandwidths. These bandwidths are noted with the label "benchmark" in Figure 16: the x axis denotes the proportion of fast memory used on the overall allocations, while the y axis represents the achieved bandwidths. The total allocated size is constant for every sample and large enough to hide caching effects.

In order to extend the comprehensiveness of the results, we also plot a grey area noted Theoretical limit bounding the theoretically achievable bandwidth. Upper and lower bounds of this area are computed from the theoretical total aggregated bandwidth when all memories can be simultaneously used (see equation 4) and the total theoretical aggregated bandwidth when each memory is sequentially used (see equation 3). In the first case, the execution time is bound by the longest of the memory access, whereas in the second case the time is bound by the sum of the memory access time. The "Model" curve is a comprehensive model built on the top of system bandwidths. It decomposes the total bandwidth into each memory bandwidth involved and weights their contribution in the final execution time. Hence, the model weights bridge the gap between the hardware characteristics and the execution context imposed by the application.

Toward the explanation of the model we introduce few notations (quantities are expressed in Bytes):

- $q_{ls}$, the **q**uantity of data to **l**oad from the **s**low memory,
- $q_{ss}$, the **q**uantity of data to **s**tore into the **s**low memory,
- $q_{lf}$, the **q**uantity of data to **l**oad from the **f**ast memory,
- $q_{sf}$, the **q**uantity of data to **s**tore into the **f**ast memory,

| | $\theta^{sf}_{lf}$ | $\theta^{ls}_{lf}$ | $\theta^{ss}_{lf}$ | $\theta^{sf}_{ls}$ | $\theta^{lf}_{ls}$ | $\theta^{ss}_{ls}$ | $\theta^{lf}_{sf}$ | $\theta^{ls}_{sf}$ | $\theta^{ss}_{sf}$ | $\theta^{sf}_{ss}$ | $\theta^{ls}_{ss}$ | $\theta^{lf}_{ss}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Skylake | 0.966 | 0.600 | -0.102 | 0.465 | 0.294 | 0.373 | 0.912 | 0.067 | 0.337 | 0.059 | 0.293 | 0.54 |
| KNL | 0.238 | 0.722 | 0.985 | 0.956 | 0.611 | 0.564 | 0.183 | 0.953 | 0.797 | 0.726 | 0.571 | 0.65 |

TABLE 6: Model parameters fitted for the Skylake and KNL systems, obtained from Figure 16.

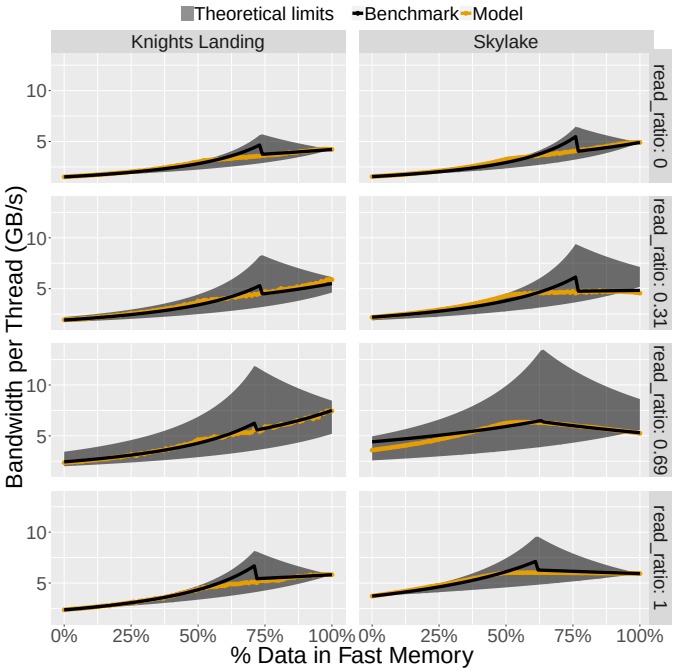



Fig. 16: Bandwidth versus Model for data locality on the KNL and Skylake systems where only stores (or only load) instructions are used.

such that $q_{ls} + q_{ss} + q_{lf} + q_{sf}$ is constant across executions. We also establish several bandwidth notations (expressed in bytes/seconds):

- $b_{ls}$ the **l**oad **b**andwidth of the **s**low memory,
- $b_{ss}$ the **s**tore **b**andwidth of the **s**low memory,
- $b_{lf}$ the **l**oad **b**andwidth of the **f**ast memory,
- $b_{sf}$ the **s**tore **b**andwidth of the **f**ast memory.

As a result, it can be observed that it takes at least:

- $t_{ls} = q_{ls}/b_{ls}$ seconds to load the data set from the slow memory,
- $t_{ss} = q_{ss}/b_{ss}$ seconds to store the data set into the slow memory,
- $t_{lf} = q_{lf}/b_{lf}$ seconds to load the data set from the fast memory,
- $t_{sf} = q_{sf}/b_{sf}$ seconds to store the data set into the fast memory.

$$t_{min} = max(t_{ls}, t_{ss}, t_{lf}, t_{sf}) \quad (3)$$

$$t_{max} = t_{ls} + t_{ss} + t_{lf} + t_{sf} \quad (4)$$

$t_{min}$ (3) is the minimum time to serve memory requests if all of them are simultaneously performed. $t_{min}$ stands for a lower bound of the achievable time. It shall not be confused with the actual transfer time that might be longer. With these notations, the top of the grey area in Figure 16 represents $(q_{ls} + q_{ss} + q_{lf} + q_{sf})/t_{min}$. In contrast, the bottom of the grey area represents $(q_{ls} + q_{ss} + q_{lf} + q_{sf})/t_{max}$, where $t_{max}$ (4) refers to the amount of time required to serve memory requests by each of accessed memories serially. $t_{max}$ stands for an upper bound of the achievable transfer time. It shall not be confused with the actual transfer time that might be shorter.

One can see in Figure 16 that in extreme cases (load_ratio $\in \{0, 1\}$and fast_ratio $\in \{0, 1\}$), the bandwidth matches the limits of the "Theoretical limits" area. Hence, both $t_{min}$ or $t_{max}$ can be used as the intercept term of the linear model of the total transfer time.

We split the model into four parts to treat separately the cases where each of the four bandwidths is dominating the data transfer. They are used to obtain a better curve fitness when there are sudden breaks and the measured bandwidth does not seem differentiable. In order to reach this objective, we define the following set of flags:

- $ismax_{lf} = 1\ if\ max(t_{ls}, t_{ss}, t_{lf}, t_{sf}) = t_{lf}\ else\ 0$,
- $ismax_{sf} = 1\ if\ max(t_{ls}, t_{ss}, t_{lf}, t_{sf}) = t_{sf}\ else\ 0$,
- $ismax_{ls} = 1\ if\ max(t_{ls}, t_{ss}, t_{lf}, t_{sf}) = t_{ls}\ else\ 0$,
- $ismax_{ss} = 1\ if\ max(t_{ls}, t_{ss}, t_{lf}, t_{sf}) = t_{ss}\ else\ 0$,

such that $ismax_{lf} + ismax_{sf} + ismax_{ls} + ismax_{ss} = 1$. With this notation, we can rewrite the intercept term $t_{min} = t_{lf} * ismax_{lf} + t_{ls} * ismax_{ls} + t_{sf} * ismax_{sf} + t_{ss} * ismax_{ss}$ split into four parts.

For each of the four model parts, the slope is defined as the weighted sum of non-dominating bandwidths which do not overlap the dominating bandwidth with a quantity $\theta$. We define $\theta$ weights sub-scripted with the first letters of the dominating bandwidth and super-scripted with the non-dominating bandwidth considered. For instance, $\theta^{ls}_{lf}$ represents the quantity of data transfer loaded from the slow memory and that is not overlapped by the transfer of data loaded from the fast memory. With this notation, the slopes of the four model parts are written as follows:

- $ismax_{lf} * (\theta^{ls}_{lf} * t_{ls} + \theta^{sf}_{lf} * t_{sf} + \theta^{ss}_{lf} * t_{ss})$, the not overlapped transfer times when the whole time is dominated by loads in the fast memory,
- $ismax_{ls} * (\theta^{lf}_{ls} t_{lf} + \theta^{sf}_{ls} t_{sf} + \theta^{ss}_{ls} t_{ss})$, the not overlapped transfer times when the whole time is dominated by loads in the slow memory,
- $ismax_{sf} * (\theta^{ls}_{sf} t_{ls} + \theta^{lf}_{sf} t_{lf} + \theta^{ss}_{sf} t_{ss})$, the not overlapped transfer times when the whole time is dominated by stores in the fast memory,
- $ismax_{ss} * (\theta^{ls}_{ss} t_{ls} + \theta^{lf}_{ss} t_{lf} + \theta^{sf}_{ss} t_{sf})$, the not overlapped transfer times when the whole time is dominated by stores in the slow memory.

Finally, the fit of approximate spent time in data transfers

is expressed as follows:

$$
\begin{aligned}
t_{fit} = \quad & \\
& ismax_{lf} * (\theta^{ls}_{lf} * t_{ls} + \theta^{sf}_{lf} * t_{sf} + \theta^{ss}_{lf} * t_{ss} + t_{lf}) + \\
& ismax_{ls} * (\theta^{lf}_{ls} * t_{lf} + \theta^{sf}_{ls} * t_{sf} + \theta^{ss}_{ls} * t_{ss} + t_{ls}) + \\
& ismax_{sf} * (\theta^{ls}_{sf} * t_{ls} + \theta^{lf}_{sf} * t_{lf} + \theta^{ss}_{sf} * t_{ss} + t_{sf}) + \\
& ismax_{ss} * (\theta^{ls}_{ss} * t_{ls} + \theta^{lf}_{ss} * t_{lf} + \theta^{ss}_{ss} * t_{sf} + t_{ss})
\end{aligned}
\tag{5}
$$

where each $\theta$ parameter is adjusted with a linear regression and quantifies the system ability to overlap a pair of bandwidths when one is dominating the total transfer time. The fitted parameters for the tested Skylake system, as well as for the KNL system, are given in Table 6.

The "Benchmark" curve on Figure 16 is obtained by assessing the system heterogeneous bandwidth along a wide range of memory access patterns. Not only the hereby obtained roof can help to troubleshoot performance issues, but it also provides a simplified interpretation of the system bandwidth. The average model error[10] is below 3% for each platform, thus highlighting the model quality. However, the model shows a protuberance at 50% of the data allocated in fast memory, where it does not break although the real bandwidth seems to. This shortcoming can easily be overcome by introducing additional parameters, *e.g.* by splitting the model to introduce a hypothetical intermediate memory element and estimate its bandwidth. However, this approach would weaken the model interpretability, since this new element would not be built to match a real system component for which we can measure the bandwidth but instead a hypothetical one.

### 6.5 Model Validation with synthetic benchmarks

The final step of this work is to verify the capability of the proposed model to characterize the behaviour of highly optimized memory-bound applications. For this purpose, a set of STREAM benchmarks, expected to approach the modeled roofs, are used. The benchmarks ratio of loads and stores is constrained, however the memory allocation policy is not. Hence, we validate the roof along the locality dimension for the three applications. The buffers for reading and writing are all allocated with the same methodology as the benchmarks are, as it is presented in Figure 15. Figure 17 presents the validation chart, where the synthetic benchmarks are matched with the closest micro-benchmark. For each (slow, fast) ratio, we plot the achieved bandwidth when compared to the micro-benchmark bandwidth, fitted model bandwidth and associated model with $\theta$ parameters (computed with platform micro-benchmarks) from Table 6. We also plot the system raw load and store bandwidths with horizontal lines. The latter clearly shows (see Knights Landing in Figure 17) the importance of providing a hybrid roof when compared to raw bandwidths, especially when extracting the information that a memory bandwidth bottleneck is reached. Though the "Model" is still a good fit to benchmarks, both feature a different bandwidth when compared with *scale* or *triad*, on the Skylake system. Nevertheless, since they perform better than the application, they still can stand as a roof of the achievable bandwidth.

10. The model error is computed as root mean squared error of bandwidth differences between the benchmarks $y_i$ and the model $\hat{y}_i$: $\frac{100}{n} \times \sqrt{\sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{\hat{y}_i}\right)^2}$

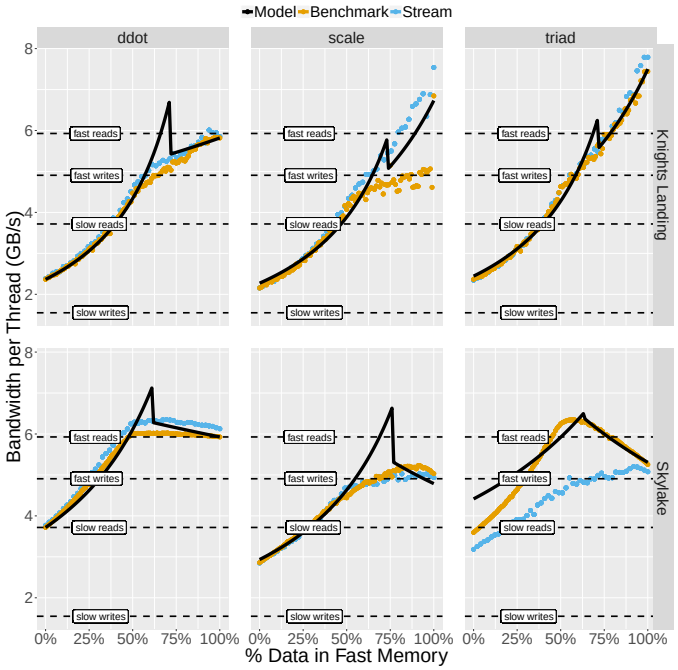



Fig. 17: Comparison of STREAM benchmarks bandwidth (application) with systems bandwidths (benchmark) assuming the same load/store ratio, with the system raw bandwidths as horizontal (roof) lines, and with the model of the system bandwidth (Model).

## 7 Related Works

To this date, there are two main approaches for Roofline modeling, namely: the Original Roofline Model (ORM) [8] and the Cache-Aware Roofline Model (CARM) [6]. Unlike the CARM that includes the complete memory hierarchy in a single plot, the ORM mainly considers the memory transfers between the last level cache and the DRAM, thus it provides fundamentally different perspective and insights when characterizing and optimizing applications [17]. Recently, the ORM was also instantiated on the KNL [18], without modifying the original model. The arithmetic intensity (AI) described in ORM is not to be confused with CARM AI because of the difference in the way how the memory traffic is observed. The bandwidth measured also differs from the one measured in this paper, the latter being explicitly load bandwidth. In [18], the authors present several ORM-based optimization case studies, and compare the performance improvements between Haswell processor and KNL, with data in DDR4 memory or MCDRAM, and finally KNL with data in MCDRAM memory. However, the authors do not show how the model can help choosing between memories when working sets do not fit in the fastest one nor they provide a comparison with the cache mode.

An extension to the ORM, named 3DyRM [19], has been proposed to provide locality insights on NUMA systems. This model considers memory accesses from a single last

level cache to any other memory, and not only local memory. It extends the ORM with a latency dimension to characterize the sampled memory access. Not only 3DyRM inherits the distorted perspective of the ORM, when characterizing real-world applications, but it also gives very limited insights on the distance of memory accesses to the NUMA thresholds considered in this paper. Moreover, 3DyRM characterizes applications with sampled memory accesses, without classifying them nor providing a methodology to get the first order insights, which is the main goal of the legacy model.

The *Capability Model* [15] was recently proposed to evaluate KNL realistic upper-bounds and guide applications performance optimization. The authors established a complex model mostly focusing on latency and bandwidth of the mesh interconnect. The Capability Model focuses on communication intensive algorithms (such as barrier synchronization, reduction, sorting, etc), whereas the LARM has a throughput-oriented approach, focusing on computational workloads stressing both compute and memory units. As such, the Capability model suits better message passing programming paradigms to enhance communication-based algorithms, while the LARM suits better shared memory programming paradigms where communications are not explicitly expressed and mixed with computations.

*Execution Cache Memory* (ECM) [20] is also another insightful approach to model performance of memory-bound applications. This model is built under similar assumptions as the CARM when modeling the performance of processing elements and memory levels, e.g., by considering their maximum throughput. However, the ECM aims at predicting the application runtime whereas the CARM aims at providing insights toward application characterization and optimization. Moreover, to the best of our knowledge, there are no studies demonstrating the usability of the ECM for NUMA and heterogeneous memory systems featuring emerging heterogeneous memory technologies.

Our contribution to the CARM can also advance its current implementation in the Intel proprietary tool's, referred as Intel Advisor Roofline [21], and for which some of the authors of this paper published concrete use cases [22]. Unlike Intel Advisor Roofline, we keep track of the MCDRAM bandwidth in several aspects, and provide additional insights about potential bottlenecks and characteristics of NUMA systems. Indeed, we demonstrated that our model improvements can efficiently spot locality-related issues, and provide different interpretation methodology, especially in the case of traditional multi-socket systems.

## 8 Conclusions

The trend of increasing the number of cores on-chip is enlarging the gap between compute power and memory performance. This issue leads to design systems with heterogeneous memories, creating new challenges for data locality. Before the release of those memory architectures, the Cache-Aware Roofline Model offered an insightful model and methodology to improve application performance with knowledge of the cache memory subsystem. We build new roofs characterizing bottlenecks typical of NUMA systems and showed that this additional information can help to successfully spot locality issues arising from different sources, such as data allocation policy or memory configuration.

This paper also focused on overcoming the limitations and extending its ability to pinpoint a wider set of application bottlenecks. The model limitations were highlighted with concrete synthetic benchmark cases. Based on this observation, we built a multi-bandwidth model and methodology to design a heterogeneous roof in the model for replacing some of the previously proposed roofs. By design, the fitted model also quantifies the system ability to overlap bandwidths when simultaneously accessing several memories with different characteristics. Finally, the relevance of the proposed approach was validated by matching a set of applications with it.

A trail toward future works is to provide deeper insights by modeling the memory latency and the different access patterns. So far, we only modeled contiguous memory accesses or typical interleaved patterns. However, it could be interesting to measure the achievable performance gains from a complex memory access pattern that could be simplified. Generalizing the model to other devices and architectures (also embedding heterogeneous memories) represents one of the future research directions with a potentially high interest to the scientific community.

## Acknowledgments

We would like to acknowledge COST Action IC1305 (NESUS), Intel Corporation and Atos for funding parts of this work, as well as Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/50021/2013 and LISBOA-01-0145-FEDER-031901 (PTDC/CCI-COM/31901/2017). Some experiments presented in this paper were carried out using the PLAFRIM experimental test-bed, being developed under the Inria PlaFRIM development action with support from Bordeaux INP, LaBRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux and CNRS (and ANR in accordance to the programme d'investissements d'Avenirs, see https://www.plafrim.fr/).

## References

[1] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," IEEE Signal Processing Magazine, vol. 26, no. 6, pp. 26–37, November 2009.

[2] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A case for numa-aware contention management on multicore systems," in 2011 USENIX Annual Technical Conference, Portland, OR, USA, June 15-17, 2011, 2011.

[3] J. Reinders, J. Jeffers, and A. Sodani, "Intel xeon phi processor high performance programming knights landing edition," 2016.

[4] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "Intel® quickpath interconnect architectural features supporting scalable system architectures," in High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on. IEEE, 2010, pp. 1–6.

[5] "Bull atos technologies: Bull coherent switch," http://support.bull.com/ols/product/platforms/hw-extremcomp/hw-bullx-sup-node/BCS/index.htm.

[6] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware roofline model: Upgrading the loft," IEEE Computer Architecture Letters, vol. 13, no. 1, pp. 21–24, 2014.

[7] N. Denoyelle, B. Goglin, A. Ilic, E. Jeannot, and L. Sousa, "Modeling large compute nodes with heterogeneous memories with cache-aware roofline model," in High Performance Computing systems - Performance Modeling, Benchmarking, and Simulation - 8th International Workshop, PMBS 2017, ser. Lecture Notes in Computer Science, vol. 10724. Denver (CO), United States: Springer, Nov. 2017, pp. 91–113. [Online]. Available: https://hal.inria.fr/hal-01622582

[8] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," Commun. ACM, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[9] C. Cantalupo, V. Venkatesan, J. Hammond, K. Czurlyo, and S. D. Hammond, "memkind: An extensible heap memory manager for heterogeneous memory platforms and mixed memory policies." Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2015.

[10] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: a generic framework for managing hardware affinities in hpc applications," in PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, IEEE, Ed., Pisa, Italy, Feb. 2010.

[11] A. Kleen, "A numa api for linux," Novel Inc, 2005.

[12] B. Lepers, V. Quema, and A. Fedorova, "Thread and memory placement on numa systems: Asymmetry matters," in 2015 USENIX Annual Technical Conference (USENIX ATC 15). Santa Clara, CA: USENIX Association, Jul. 2015, pp. 277–289.

[13] C. Chou, A. Jaleel, and M. K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2014.63

[14] B. Lepers, V. Quéma, and A. Fedorova, "Thread and memory placement on numa systems: Asymmetry matters." in USENIX Annual Technical Conference, 2015, pp. 277–289.

[15] S. Ramos and T. Hoefler, "Capability models for manycore memory systems: A case-study with xeon phi knl," in 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2017, pp. 297–306.

[16] "The memkind library," http://memkind.github.io/memkind.

[17] A. Ilic, F. Pratas, and L. Sousa, "Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores," IEEE Transactions on Computers, vol. 66, no. 1, pp. 52–58, Jan 2017.

[18] D. Doerfler, J. Deslippe, S. Williams, L. Oliker, B. Cook, T. Kurth, M. Lobet, T. Malas, J.-L. Vay, and H. Vincenti, "Applying the roofline performance model to the intel xeon phi knights landing processor," in International Conference on High Performance Computing. Springer, 2016, pp. 339–353.

[19] O. G. Lorenzo, T. F. Pena, J. C. Cabaleiro, J. C. Pichel, and F. F. Rivera, "Using an extended roofline model to understand data and thread affinities on numa systems," Annals of Multicore and GPU Programming, vol. 1, no. 1, pp. 56–67, 2014.

[20] J. Hofmann, J. Eitzinger, and D. Fey, "Execution-cache-memory performance model: Introduction and validation," CoRR, vol. abs/1509.03118, 2015.

[21] Intel. (2017) Intel® advisor roofline. [Online]. Available: https://software.intel.com/en-us/articles/intel-advisor-roofline

[22] D. Marques, H. Duarte, A. Ilic, L. Sousa, R. Belenov, P. Thierry, and Z. A. Matveev, "Performance analysis with cache-aware roofline model in intel advisor," in 2017 International Conference on High Performance Computing Simulation (HPCS), July 2017, pp. 898–907.


**Nicolas Denoyelle** Nicolas Denoyelle is a PhD. student at Atos Bull Technologies, an HPC system manufacturer and Inria Bordeaux, the French public institute for research in computer science. Nicolas researches has been lead on application and platforms modeling for improving data locality in high-performance shared memory systems.

**Brice Goglin** Brice Goglin is research scientist at Inria Bordeaux, the French public institute for research in computer science. He earned his PhD at Ecole normale superieure de Lyon (France) in 2005. He then worked for Myricom, inc. (Oak Ridge, TN) as a software architect for low latency networks. His research interests at Inria now include the management of data locality in many-core HPC platforms as well as high performance I/Os.

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

**Emmanuel Jeannot** Emmanuel Jeannot is a Senior Research Scientist at Inria. He is doing his research at INRIA Bordeaux Sud-Ouest and at the LaBRI laboratory since 2009. From 2005 to 2006 he was researcher at INRIA Nancy Grand-Est. In 2006, he was a visiting researcher at the University of Tennessee, ICL laboratory. From 2000 to 2005 he was assistant professor at the Université Henry Poincaré. During the period from 2000 to 2009 he did his research at the LORIA laboratory. He got his PhD and Master degree of computer science (resp. in 1996 and 1999) both from Ecole Normale Supérieur de Lyon, at the LIP laboratory. His main research interests lie in parallel and high-performance computing and more precisely: process placement, topology-aware algorithms, scheduling for heterogeneous environments, data redistribution, algorithms and models for parallel machines, distributed computing software, adaptive online compression, and programming models.

**Leonel Sousa** Leonel Sousa (M'01–SM'03) received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Tecnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 1996. Since 1996, he has been with IST, where he is currently the Chair of the Department of Electrical and Computer Engineering, and has been a Full Professor since 2010. In 2016, he was a Visiting Professor with Tsukuba University, Tsukuba, Japan, with a JSPS Invitation Fellowship for Research in Japan, and Carnegie Mellon University, Pittsburgh, PA, USA. His research interests include computer architectures, high performance computing, and multimedia systems. He has contributed over 250 papers for international journals and conferences and to the organization of several international conferences. Dr. Sousa is currently an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS. He is also a Distinguished Scientist of the ACM.