# D2.5

# Report on the HPC application bottlenecks

| | | | |
|---|---|---|---|
| **Workpackage:** | 2 | Co-Design for Exa-scale HPC System | |
| **Author(s):** | Kazi Asifuzzaman | | BSC |
| | Milan Radulovic | | BSC |
| | Petar Radojkovic | | BSC |
| **Authorized by** | Petar Radojkovic | | BSC |
| **Reviewer** | Said Derradji | | Atos |
| **Reviewer** | Denis Dutoit | | CEA |
| **Reviewer** | Luca Benini | | ETH |
| **Dissemination Level** | Public | | |

| Date | Author | Comments | Version | Status |
|---|---|---|---|---|
| 2017-03-07 | Milan Radulovic | | V1.0 | Draft |
| 2017-03-27 | Milan Radulovic | Updates based on the comments by the reviewers. | V1.1 | Final |
| | | | | |

# Executive Summary

Designing a balanced HPC system requires understanding of the dominant performance bottlenecks of present state-of-the-art HPC systems and workloads. In the context of the ExaNoDe Task 2.5, we analysed main performance bottlenecks in relevant industrial HPC applications running on a state-of-the-art HPC platform.

We base our analyses on three important bottlenecks in modern state-of-the-art HPC systems: percentage of execution time spent in inter-process communication, memory bandwidth congestion and FLOPs performance. The presented analysis points out the degree of performance degradation for each of the bottlenecks, while running realistic HPC workloads.

Inter-process communication may account for a significant portion of the overall execution time, and may be one of the main limitations for the application scalability. As applications scale-out to higher number of processes (strong-scaling case), our analysis shows that the percentage of time spent in inter-process communication increases, reaching more than 50% in some cases. Therefore, in the context of the project, it is important to deploy new approaches for inter-process communication such as UNIMEM, and also to decrease the load imbalance, using various programming models which are presented in WP3.

Memory bandwidth became one of the main performance bottlenecks in current HPC systems. We measured memory bandwidth usage of relevant HPC applications and the results show that majority of applications experience memory bandwidth congestion. Therefore, our recommendation would be to look toward high-bandwidth memories, such as High Bandwidth Memory (HBM) or Hybrid Memory Cube (HMC), in future ExaNoDe architectures. Additionally, we show the importance of the methodology used to measure application bandwidth – the analysis of average (end-to-end) bandwidth measurements could mislead, and fail to detect segments of applications that are bandwidth intensive. Afterwards, we analyzed the effects of scaling-out on memory bandwidth and found that strong scaling reduces pressure on memory bandwidth, mainly due to the increment in the inter-process communication time, and better utilization of the on-chip caches.

In addition to memory bandwidth congestion, it is important to understand FLOPs vs. memory bandwidth ratio, and which parameter is the dominant performance bottleneck. We analyze FLOPS vs. memory bandwidth ratio for HPC applications by using a roofline model. Our results show that the majority of HPC applications is not bounded by the CPU processing capacity, i.e. that memory bandwidth is a dominant performance bottleneck. When scaling out, due to strong scaling, on-chip caches are utilized better so application operational intensity and GFLOPs performance increase. Again, using average values for memory bandwidth and GFLOPs performance is misleading and can show wrong trends. For future Exascale HPC platforms, we recommend building a balanced system, with higher Byte-per-FLOP ratio, as it becomes more and more important to assess performance of HPC systems based on memory system.

# Table of Contents

1 Introduction ..... 4
2 Experimental ..... 5
2.1 Hardware platform ..... 5
2.2 Applications ..... 5
2.3 Methodology ..... 6
3 Inter-process communication time ..... 9
3.1 Introduction and background ..... 9
3.2 Results ..... 9
3.3 Summary ..... 11
4 Memory bandwidth ..... 12
4.1 Introduction and background ..... 12
4.1.1 Memory wall ..... 13
4.1.2 Memory latency vs. memory bandwidth ..... 13
4.2 Results ..... 15
4.3 Summary ..... 17
5 Memory bandwidth vs. FLOPs analysis ..... 18
5.1 Introduction and background ..... 18
5.2 Methodology: Plotting the roofline ..... 19
5.3 Results ..... 20
5.4 Summary ..... 24
6 Conclusions ..... 25
7 References ..... 26

# 1 Introduction

Designing a balanced HPC system requires understanding of the dominant performance bottlenecks. In the context of the ExaNoDe Task 2.5, we analysed main performance bottlenecks in relevant industrial HPC applications on a main-stream HPC platform – inter-process communication time, memory bandwidth usage and FLOPs versus memory bandwidth analysis.

The rest of the document is organized as follows:

In Section 2 we give a brief description of the experimental platform on which experiments were performed and the applications used in the study. We also describe the methodology used to conduct the experiments, such as input data sets, number of processes and measurements granularity.

Section 3 describes inter-process communication time. Since it may account for a significant portion of the overall execution time, it could be one of the main limitations for the application scalability. We quantify the portion of the overall execution time that is spent in inter-process communication and analyze how it changes as the applications scale, in term of number of processes and input dataset size.

In Section 4, we analyze memory bandwidth as a performance bottleneck. We remind of the memory wall and analyze the dependency between memory latency and bandwidth. Then we measure memory bandwidth usage of the applications under study and analyze whether applications experience severe collision in memory bandwidth. We further expand our analysis on the effects on memory bandwidth when scaling out.

In Section 5 we analyze memory bandwidth versus FLOPs performance. To understand which parameter is the dominant performance bottleneck, we use a roofline model. We measure FLOPs performance and memory bandwidth of applications under study, and analyze the locations of application segments on the roofline plot. Afterwards we expand the analysis when the applications scale-out.

Finally, in Section 6 we summarize the main conclusions of our analysis.

# 2 Experimental

This section describes the experimental environment, the hardware platform and applications, used in the study. We also summarize the experimental methodology, discussing input datasets, number of processes, selection of the representative application segments, measurements granularity, and application scaling.

## 2.1 Hardware platform

All experiments in this deliverable were performed on MareNostrum supercomputer [1], which is constituted of:

- Compute node features:
    - CPU
        - Dual socket Intel SandyBridge E5-2670 (released in 2012)
        - 8 cores per socket (16 cores per node)
        - Disabled hyperthreading
        - 2.6GHz frequency
        - 8 double precisions floating point operations per cycle, per core
        - Caches:
            - L1i: 32 kB, L1d: 32 kB
            - L2: 512 kB
            - L3: 20 MB (shared by all cores in socket)
    - Memory
        - Four DDR3 memory channels per socket
        - 16 GB of DDR3-1600 RAM memory per socket (4 DIMMs x 4GB)

- Interconnect:
    - InfiniBand FDR-10 (40 Gb/s)
    - Non-blocking two-level fat-tree topology

## 2.2 Applications

We analyze four HPC applications from Unified European Application Benchmark Suite (UEABS), and two widely used HPC benchmarks – High-Performance Linpack (HPL) and High-Performance Conjugate Gradients (HPCG). UEABS applications represent set of production applications and datasets designed for benchmarking the European PRACE HPC systems for procurement and comparison purposes [2]. All UEABS applications are parallelized using MPI, and are regularly run on hundreds or thousands of cores. For all applications, we run MPI-only versions, and always execute one MPI process per core. The applications from UEABS suite used in this report are: ALYA, BQCD, CP2K and NEMO. Table 1 summarizes the applications' area of science, problem size and number of processes used in our experiments.

| Application | Area of science | Problem size | No. of processes used |
|---|---|---|---|
| ALYA | Computational mechanics | 27 million element mesh | 16–1024 |
| BQCD | Particle physics | $32^2 \times 64^2$ lattice | 64–1024 |
| CP2K | Computational chemistry | Energy calculation of 1024 waters | 128–1024 |
| NEMO | Ocean modelling | 12° global configuration 4322×3059 grid | 512–1024 |

**Table 1: List of the UEABS applications used in the study**

The High-Performance Linpack (HPL) benchmark is the most widely recognized and discussed metric for ranking of HPC systems for more than 20 years. HPL measures the sustained floating-point rate (GFLOP/s) for solving a dense system of linear equations using double-precision floating-point arithmetic [3]. The linear system is randomly generated, with a user-specified size, so that the user can scale the problem size to achieve the best performance on a given system. The documentation recommends setting a problem size that uses approximately 80% of available memory.

High-Performance Conjugate Gradients (HPCG) benchmark is introduced as a complement to HPL and TOP500 rankings, since the community questions whether HPL is a good proxy for production applications. HPCG is based on an iterative sparse-matrix conjugate gradient kernel with double-precision floating-point values [4]. HPCG is a good representative of HPC applications governed by differential equations, which tend to have much stronger needs for high memory bandwidth, low latency, and accessing data using irregular patterns. As with HPL, the user can scale the problem size to achieve the best performance on a given system.

## 2.3 Methodology

**Input dataset:** Most of the applications from UEABS package come with two input datasets. Smaller datasets (Test Case A) are deemed suitable for Tier-1 systems up to about 1000 strong x86 cores, and larger datasets (Test Case B) target Tier-0 systems up to about 10,000 cores. In most of our experiments we used smaller dataset (Test Case A).

**Number of processes:** The experiments were executed for various number of processes, from 16 up to 1024 by powers-of-two, i.e., 16, 32, 64, 128, 256, 512, and 1024. The 16 processes correspond to a single MareNostrum node, while 1024 processes are marked by the benchmark developers as the scalability limit for the input dataset Test Case A. BQCD, CP2K and NEMO applications have memory capacity requirements that exceed the available memory on a single node. This limits the lowest number of processes we use in the experiments, e.g., BQCD cannot be executed with less than 64 processes (four nodes). For each UEABS application the process range used in the experiments is listed in Table 1.

**Representative application segments:** In order to perform complex numerical computations in a reasonable time, HPC applications use numerous simultaneous processes. Trace collection and simulation of entire HPC application that comprises thousands of processes is infeasible. Therefore, first we had to analyze the application structure to detect relatively smaller application segments that are good representative of its overall behavior.

Figure 1 illustrates a visual representation of an HPC application's execution (ALYA). For different application processes (Process 1–1024), the figure shows repetitive appearance of MPI Barrier — the iterating function of the application. At the beginning of the execution (up to approximately 17s in Figure 1), in the *pre-processing phase*, HPC applications divide and distribute input data over a large number of processes. Then, in the application *main computational phase*, through a series of computation bursts and inter-process communication steps, the intermediate calculations are combined into final results. In production runs of HPC applications, duration of the pre-processing phase is negligible, so the analysis of HPC applications is primarily focused on the main computational phase. Since the computation naturally follows repetitive patterns, characterizing of a sample of the iterations is sufficient to characterize the entire application execution [5]. For each group of experiments (in each of the following sections), we specify whether we analyze the whole main computational phase or select a number of its iterations. This depends mainly on the amount of data that profiling generates, and the overhead that it introduces. While coarse-grain measurements are performed on the whole main computational phase, the detailed measurements cover a selected number of its iterations.

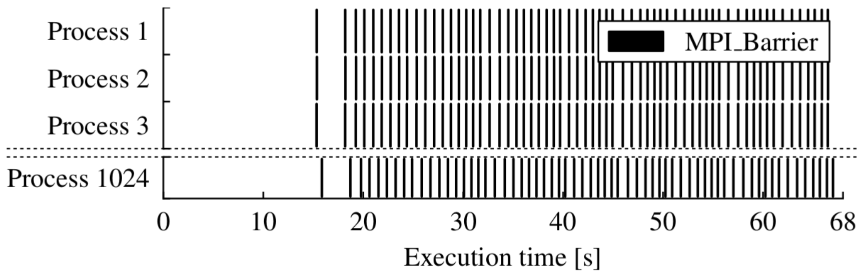

**Figure 1: Repetitive behavior of HPC applications: ALYA, 1024 processes**

**Measurements granularity:** Each HPC application process is comprised of a series of computation bursts and inter-process communication steps, MPI calls in the case of the UEABS applications, as illustrated in Figure 2. Computation bursts and communication calls have a different nature and use of the CPU resources. Communication calls in general show much lower FLOPs and memory bandwidth utilization w.r.t. the computation bursts, lowering the average applications utilization of these resources. This would mean that an application with FLOPs or memory bandwidth *intensive computational bursts* could still have a *moderate or low average utilization* of these resources, e.g. if significant portion of the overall execution time is spent in the communication. Therefore, in addition to the average application behavior based on the end-to-end measurements, we also determine the behavior at computation burst level. For each application process, we generate a series of timestamped measurements and gather more than 10000 samples per process.

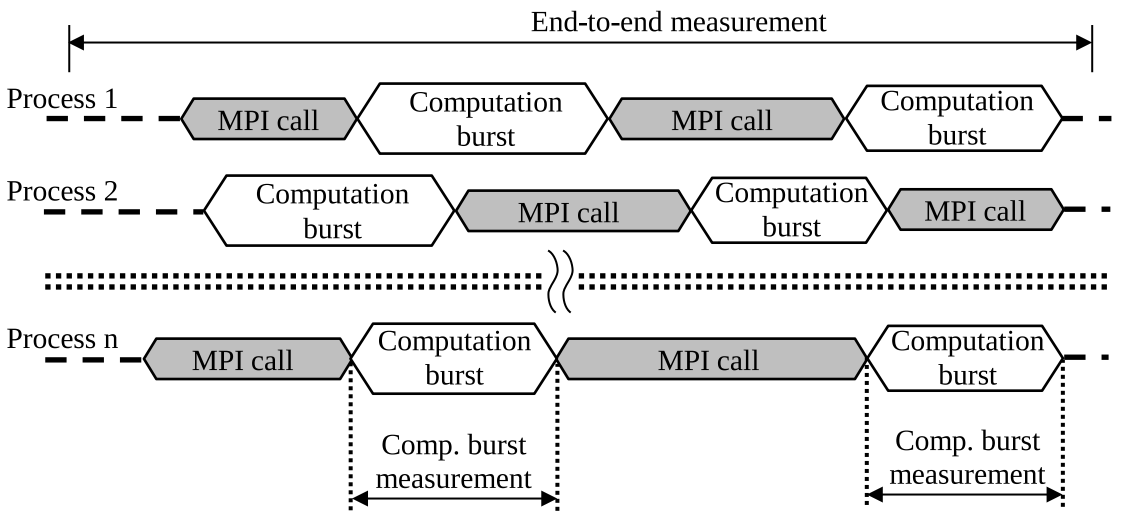

**Figure 2: We perform measurements at two levels of granularity – end-to-end measurements (in a selected interval) and series of computational burst measurements.**

**Weak and strong scaling:** For each number of processes, HPL and HPCG benchmark input problem size is set to achieve best performance at the given scale of execution, in a general case trying to keep the same problem size per process. This approach is referred to as *weak scaling* of HPC applications.

When scaling UEABS applications from minimum number of processes to 1024 processes, we keep constant the input dataset size. This refers to the *strong scaling* case of production HPC applications. In addition to this, it is also important to perform a weak scaling analysis, i.e., to analyze applications when both the number of processes and input dataset size are increased — similar as for HPL and HPCG benchmarks. Since the problem inputs are specified by the benchmark suite, such analysis requires either that the benchmark suite supports a user-defined problem size (as for HPL and HPCG) or that it provides a set of inputs specifically intended for weak scaling analysis. We are not aware of such a real application benchmark suite. UEABS for instance has just two problem sizes, Test Case A and Test Case B, and in many cases the problems being solved are fundamentally different, making them unsuitable for weak scaling analysis.

# 3 Inter-process communication time

## 3.1 Introduction and background

Inter-process communication may account for a significant portion of the overall execution time, and may be one of the main limitations for the application scalability. Inter-process communication time is influenced by two main factors. First one is the communication latency, and it accounts for the latency of the communication hardware (e.g. links and switches) and system software (e.g. MPI library) [6]. Second factor is the load imbalance between concurrently running application processes, making some processes to wait for the others [7].

In this section we quantify the portion of the overall execution time that is spent in the inter-process communication for a set of UEABS applications running on the MareNostrum supercomputer. We also analyze how the process communication time varies as the applications scale, in term of the number of processes and the input dataset size.

The time spent in the inter-process communication is measured with the Limpio [8] and Extrae [9] instrumentation tools, and the measurements focus on the application main computational phase avoiding pre-processing, as described in Section 2.

## 3.2 Results

Figure 3 plots the portion of the total execution time that is spent in the inter-process communication for the UEABS applications under study. The figure represents a strong scaling case;  results cover various number of application processes, from 16 to 1024, while maintaining the constant input dataset (see Section 2 for the details). The results presented in Figure 3, show that even on the lowest number of processes, the communication time corresponds to non-negligible portion of the overall execution time, from 13% for the BQCD up to 25% for the CP2K application. As the applications scale (strong scaling), the communication time increases, and ranges from 31% of the overall execution time for NEMO, up to 56%, 61%  62% for BQCD, CP2K and ALYA when comprising 1024 processes.

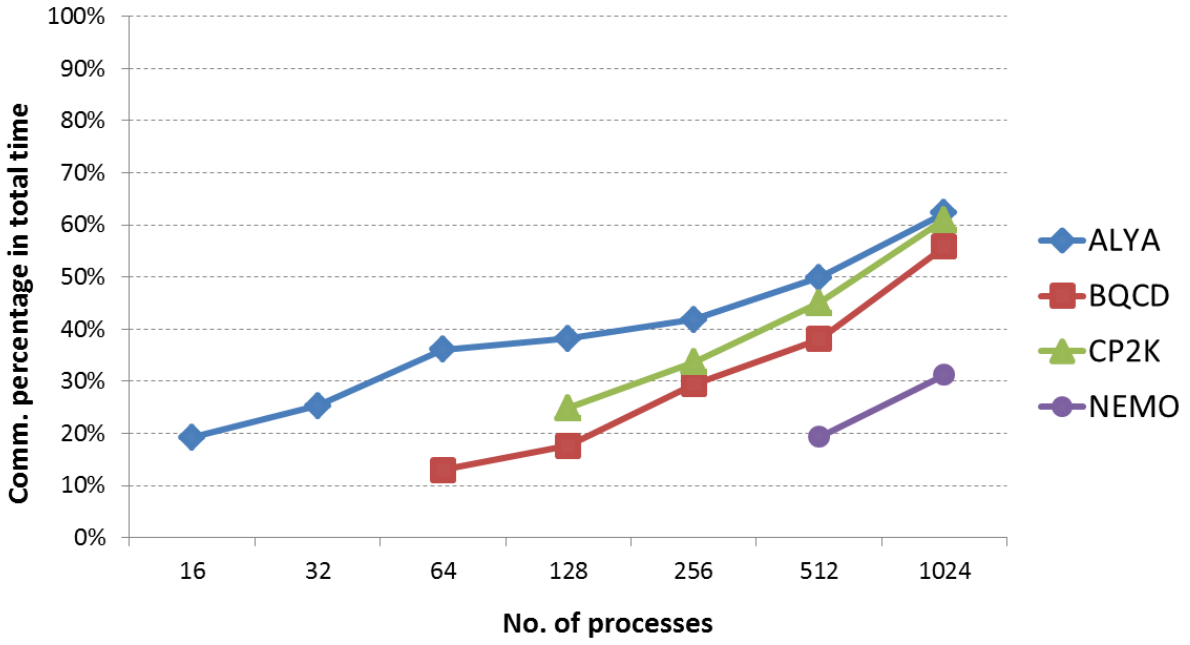

**Figure 3: Portion of the total execution time that is spent in the inter-process communication. UEABS applications, Test Case A input dataset (strong scaling).**

In addition to strong scaling analysis, presented in Figure 3, it would be also important to perform a weak scaling analysis, i.e., to analyze application communication time when both the number of processes and input dataset size are increased. This, however, is not a trivial task, as discussed in Section 2.3. The best that we could do toward the weak scaling analysis was to analyze UEABS applications solving two problem sizes, Test Case A and Test Case B.[1] Out of UEABS applications under study, we successfully installed the Test Case B only for ALYA.

Figure 4 plots the inter-process communication time for ALYA application running Test Case A and Test Case B. The Test Case B results start from the 256 processes (16 compute nodes) because the input dataset is so large that it does not fit into fewer nodes.[2] The results in Figure 4 show that the Test Case B results keep the same trend as for Test Case A, but the curve is shifted toward larger number of processes. This conclusion is aligned with the analysis of Zivanovic et al. [10] that analyze scalability of UEABS applications running both input datasets.

[1] Please, note that, for various UEABS applications, Test Case A and Test Case B solve fundamentally different problems, making them not 100% suitable for the weak scaling analysis.

[2] It could not fit into fewer power-of-two nodes, i.e. it could not fit into 8 nodes (128 processes).

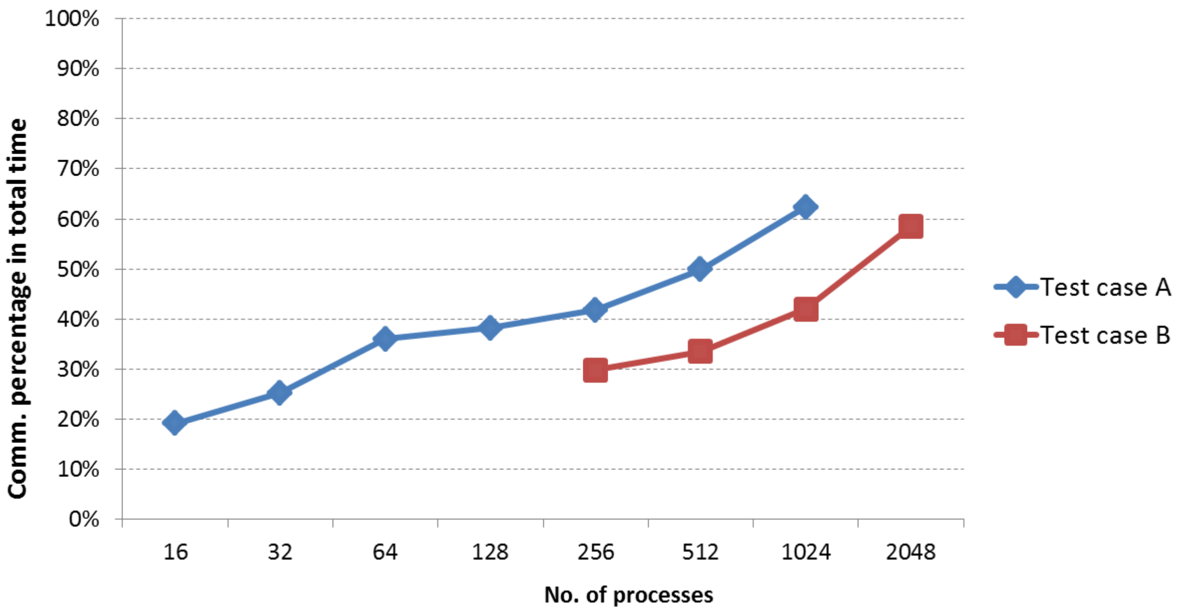

**Figure 4: Portion of the total execution time that is spent in the inter-process communication. ALYA application, Test Case A and Test Case B input datasets (weak-like scaling).**

## 3.3 Summary

Our results show that inter-process communication time accounts for a significant portion of the overall execution time, between 13% and 62% for the UEABS applications running on the MareNostrum supercomputer. Our experiments also show that application scale-out increases the portion of the time spent in the communication, having a negative impact on the application scalability.

# 4 Memory bandwidth

## 4.1 Introduction and background

In this section, we analyze whether memory bandwidth is indeed a performance bottleneck for the production HPC applications running in the current HPC system. This analysis becomes increasingly important in the recent years. One reason for this is the community was very active in advocating that the processing power (FLOPs) is not the only parameter that matters in HPC system design. An example of this movement is the release of the High-Performance Conjugate Gradients (HPCG) benchmark [11] as a complement to the FLOPs-bound HPL. Various studies analyze behavior and bottlenecks of HPCG benchmark and show that, in state-of-the-art HPC systems, memory bandwidth is the main performance bottleneck [10], [12], [13]. All this reopened a discussion on the desired ratio between CPU power (FLOPs per second) and memory bandwidth (bytes per second) in the HPC systems. Another reason for the recent popularity of the memory bandwidth discussions comes from the hardware side. High-bandwidth 3D-stacked DRAM products are hitting the market, and their manufacturers are promising significant performance boosts over the standard DDRx DIMMs. Some independent studies, however, show some doubts whether and how much will high-bandwidth memory benefit the HPC applications [14].

For some benchmarks, such as HPCG, it is easy to detect that the memory bandwidth is the main performance bottleneck. For each floating-point operation (FLOP), HPCG requires a transfer of at least 4 bytes from main memory, i.e., HPCG byte-per-FLOP ratio is higher than 4 [10], [13]. In state-of-the-art HPC systems, the byte-per-FLOP ratio is below 1.[3] This means that the HPCG will use all the available memory bandwidth, reaching performance directly proportional to it.

Production HPC applications sometimes require weighty memory-to-CPU data movements, but seldom reach 100% of memory bandwidth utilization.[4] In this case, analysis whether the memory bandwidth collisions could have a performance impact, requires an intermediate step – understand of the relation between the used memory bandwidth and main memory access time. In this section, first we remind the reader of the memory wall, and analyze the dependency between memory access latency and bandwidth. Then, we measure the memory bandwidth usage of the representative HPC applications, and we detect the applications that experience severe collision in the memory bandwidth. We perform two sets of experiments, one measuring average bandwidth, and other analyzing the distribution of memory bandwidth usage in time, and we show that the approach could significantly impact the results and the conclusions of the study. Finally, our results indeed show that memory bandwidth is a serious bottleneck in state-of-the-art HPC systems. Our analysis also shows that as the applications scale-out (strong scaling), their pressure to the memory bandwidth reduces, mainly due to the increment in the inter-process communication time, and the fact that smaller per-process data may show better utilization of the on-chip caches.

[3] Our compute node comprises two 8-core Sandy Bridge sockets and each core can execute up to 8 double-precision FLOPs per cycle. Each socket has four 64-bit wide, 1.6 GHz memory channels. Therefore byte/FLOP = $8\times(8 \text{ bytes})\times(1.6 \text{ GHz}) / 16\times(8 \text{ FLOP})\times(3 \text{ GHz}) = 0.27$

[4] As we seldom detect 100% utilization of any hardware resource.

### 4.1.1 Memory wall

*Hitting the memory wall* refers to the fact that memory latency (relative to the CPU frequency) is so large that the processor spends significant number of cycles waiting for the data from the memory [15]. First time defined in 1995, the memory wall still imposes a fundamental limitation to system performance.

One set of arguments maintained that latency-tolerance techniques like out of order execution, wider instruction issue, and speculative techniques such as hardware prefetching would bridge the processor-memory performance gap. Even in combination, though, such approaches can mask the latency of only so many memory requests — the exact numbers depend on the sizes of the hardware structures. The cost and complexity of implementing larger and larger structures proved prohibitive, and although latency tolerance pushed the memory wall back, it did not save us.

Another set of arguments maintained that new memory technologies like Intelligent RAM (IRAM) and Rambus Direct DRAM (RDRAM), or high-bandwidth 3D-stacked DRAM, such as High Bandwidth Memory (HBM) or Hybrid Memory Cube (HMC). DRAM manufacturers and IP providers continue to deliver high-performance DRAM technology, but they have not (yet) removed the memory wall. Main memory latency remains limited by the speed of the underlying storage technology, and even increases in the complex memory system solutions.[5]

To summarize, technological evolutions and revolutions notwithstanding, the memory wall (main memory access latency) has imposed a fundamental limitation to system performance for more than 20 years.

### 4.1.2 Memory latency vs. memory bandwidth

Although memory latency and bandwidth are often described as independent concepts, they are inherently interrelated. Actually, understanding the relation between the memory latency and bandwidth is essential for understanding the memory bandwidth impact on the performance, as we will describe in this section.

When analyzing memory access latency, we distinguish between single-access latency in an idle system and latency in a system with many concurrent memory accesses. *Idle-system memory latency* includes time spent in the CPU (load/store queues, cache memory, and on-chip memory controller), memory channel, and main memory. *Full-system memory latency* considers shared-resource contention among concurrent memory requests. Figure 5 shows the impact of such contention on memory latency [16]. The x-axis shows application memory bandwidth, and the y-axis shows the corresponding latency. This bandwidth-latency curve has three regions that are limited by the maximum sustainable bandwidth (which is 65-75% of the maximum theoretical bandwidth). First, when application bandwidth utilization is low, memory latency is constant and equals idle-system latency. This region has few concurrent memory requests and negligible contention for shared hardware resources. The memory latency is constant until application bandwidth reaches approximately 40% of the maximum sustainable bandwidth. After this point, increases in bandwidth needs also increase contention for shared resources, which, in turn, increases memory latency. Memory latency increases

[5] In D2.2 we analyze main memory latency of the KNL system comprising 3D-stacked DRAM.

linearly with application bandwidth usage in the region between 40% and 80% of the sustainable bandwidth.

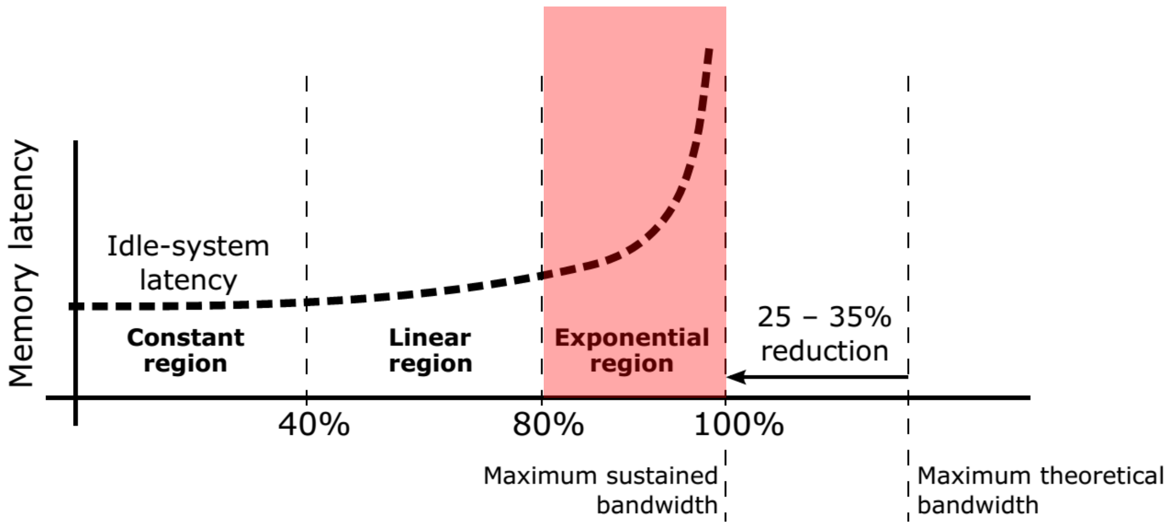

**Figure 5: Memory access latency versus used memory bandwidth**

Further increases in application bandwidth needs cause severe collisions among concurrent memory requests, and thus memory latency increases exponentially.[6]

In the context of our analysis, it is sufficient to understand that if the application memory bandwidth is below approximately 40% of the sustained one, the collision between concurrent memory requests (memory bandwidth collision) has no impact on the main memory latency and the overall performance. If the application memory bandwidth exceeds the 40% of the sustained one, the memory bandwidth starts to impact the memory latency and the overall performance. When the application uses more than 80% of the sustained memory bandwidth, the collisions between concurrent memory requests increases memory latency severely, i.e. memory bandwidth becomes a serious performance bottleneck.

In order to understand whether memory bandwidth is a performance bottleneck in the state-of-the-art HPC systems, we measure the memory bandwidth usage of the representative HPC applications and position it in the corresponding region of the memory latency vs. bandwidth curve.

[6] The dependency between memory latency and bandwidth could be also explained by the queueing theory; the trend presented in Figure 5 corresponds to the mean system response time as a function of request arrival rate.

## 4.2 Results

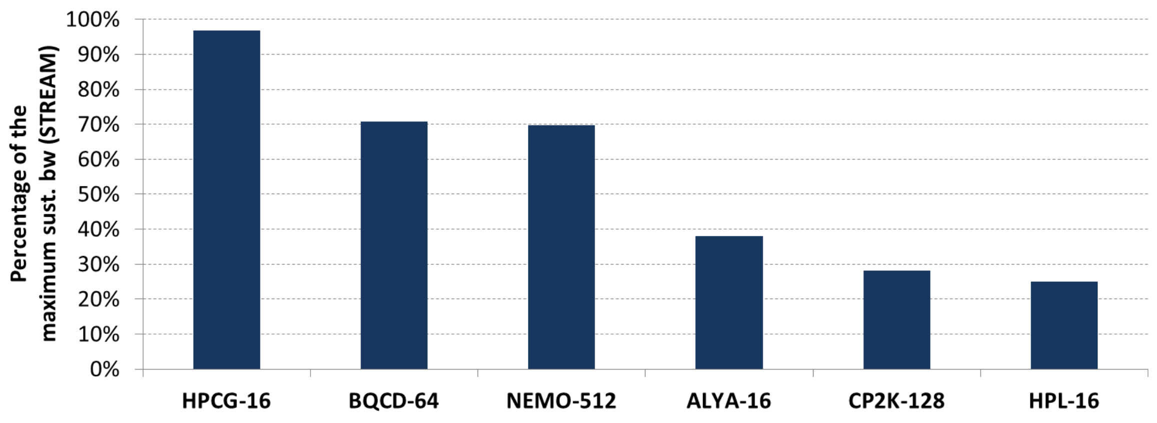

**Figure 6: Average memory bandwidth of HPC applications under study**

Figure 6 plots the memory bandwidth usage of the HPL, HPCG and the UEABS applications. HPL and HPCG are executed on a single MareNostrum server (16 processes), while UEABS applications are executed on the lowest possible number of cores (servers), 16 cores for ALYA, 64 for BQCD, 128 for CP2K and 512 for NEMO [10]. The memory bandwidth usage is plotted relative to the sustained memory bandwidth.

HPCG has high bandwidth utilization, 97% of the sustained, which confirms the analysis of [10] that memory bandwidth is a serious bottleneck. On the other extreme, HPL has a low bandwidth utilization, 25% of the sustained one (HPL is CPU bound application that makes a good use of the on-chip caches). ALYA and CP2K use 38% and 28% of the sustained bandwidth meaning that the memory bandwidth should have no performance impact. BQCD and NEMO, reach around 70% of the sustained bandwidth, meaning that the memory bandwidth collision could lead to some performance penalties; however, they are still out of the exponential part of the bandwidth latency curve (>80% of the sustained memory bandwidth) in which the bandwidth becomes a serious performance bottleneck.

The presented results are based on the average application memory bandwidth.[7] However, since the application behavior may change during the execution, we also analyze in-time memory bandwidth measurements (sampled on computational bursts, as explained in Section 2.3).

In our experiments, we measured used memory bandwidth per process, at CPU computation burst granularity. We compare average bandwidth usage and burst memory bandwidth usage. This way, the analysis shows that even when average bandwidth appears low, periods of high memory bandwidth bursts present a performance bottleneck.

[7] In the main computational phase, see the Section 2.

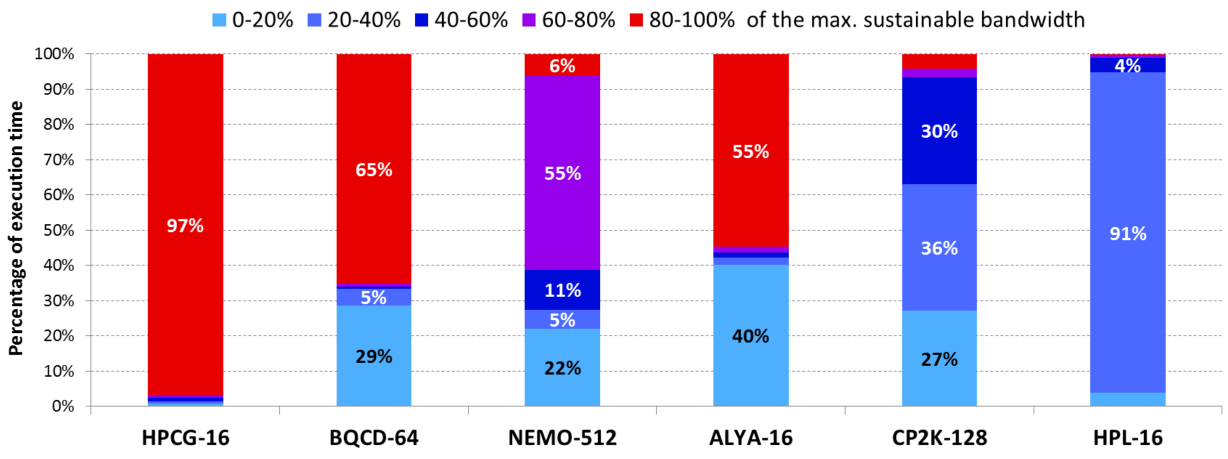

**Figure 7: Memory bandwidth usage of HPC applications on computation burst granularity**

Figure 7 shows memory bandwidth measurements for the HPL and HPCG benchmarks, and UEABS applications, similar to the Figure 6 (previous one). However, instead of showing average memory bandwidth, in Figure 7 we plot the percent of the execution time that the application spends in a given bandwidth utilization rate. Red portion of the bars represents the percentage of total execution time where the applications use high bandwidth (more than 80% of the sustainable limit). Purple section of the bars represent moderately high bandwidth usage period of the execution. Blue and lighter blue shades indicate the time periods where applications use lower bandwidth.

As depicted in Figure 7, HPCG has a serious bottleneck with 97% of the execution time in the segment 80-100% of the sustained bandwidth. HPL has no memory bandwidth problems, with >90% of the execution time in 0-40% segment. BQCD and NEMO suffer high and moderately high (respectively) bandwidth congestion during more than half of the total execution time. CP2K shows no major bandwidth bottlenecks during most of its execution time. Perhaps, ALYA has the most interesting scenario among all the applications. In Figure 6 (end-to-end measurements), ALYA seemed to be a bandwidth insensitive application. But, the detailed in-time measurements (sampled on computational bursts, as explained in Section 2.3) show 55% of its execution time, it is a bandwidth bound, with application bandwidth of above 80% of the sustained limit.

Therefore, we can emphasize to have the in-time measurements. Average end-to-end measurements could be misleading. Two out of four UEABS applications experience performance penalties due to the memory bandwidth collision. BQCD and ALYA are memory bandwidth bound in most of their execution time.

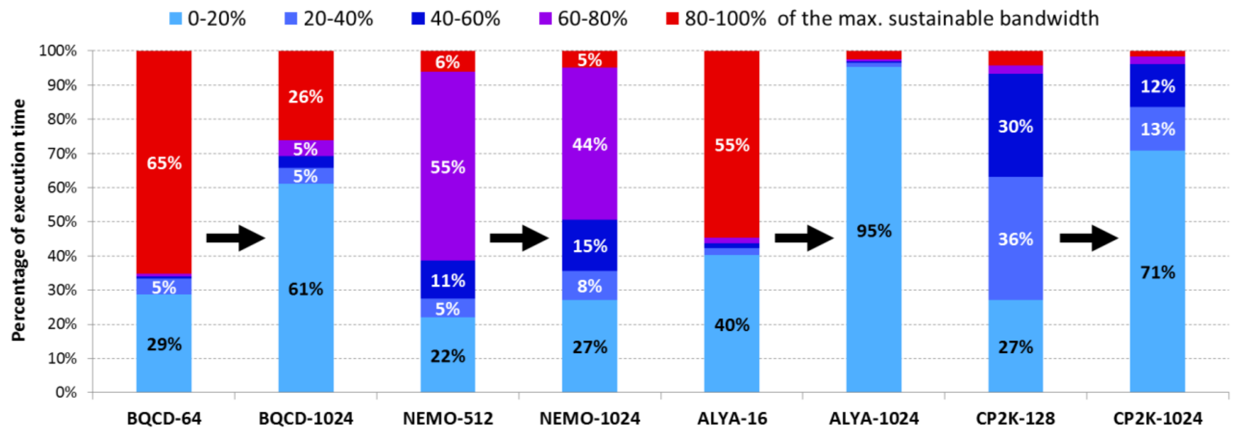

**Figure 8: Memory bandwidth usage when scaling-out, at computation burst granularity**

We further analyze application's bandwidth usage by scaling out. Figure 8 shows the impact on bandwidth usage when applications are executed on 1024 cores. All applications suffer less bandwidth congestion when executed on 1024 cores. BQCD's bandwidth bound execution period decreased to 26% from 65%. NEMO's bandwidth stress improves slightly when scaled out to 1024 cores. ALYA eliminates almost its entire bandwidth bottleneck by scaling out to 1024 cores. CP2K also shows promising improvement in bandwidth usage after scaling out.

## 4.3 Summary

Our results indeed show that memory bandwidth is a serious bottleneck in state-of-the-art HPC systems. In addition to this, we show the importance of methodology used to measure application bandwidth – the analysis of average (end-to-end) bandwidth measurements could mislead, and fail to detect applications that are bandwidth intensive in some of their segments. Our analysis also shows that as the applications scale-out (strong scaling), pressure on memory bandwidth reduces, it is mainly due to the increment of the inter-process communication time, and the fact that smaller per-process data may show better utilization of the on-chip caches.

# 5 Memory bandwidth vs. FLOPs analysis

## 5.1 Introduction and background

In addition to traditionally considered FLOPs, memory bandwidth is recognized to be an important system parameter. Therefore, building a balanced HPC node requires understanding of the desired FLOPs vs. memory bandwidth ratio, and understanding which parameter is the dominant performance bottleneck. In this section, we analyze FLOPS vs. memory bandwidth ratio for the HPC applications by using a roofline model [17]. The roofline is a bound and bottleneck model – it specifies the performance bounds of an application running on a given system, and it determines the dominant performance bottleneck (the processing power or the memory bandwidth).

We illustrate an example of the roofline model in Figure 9. The X-axis of the model refers to the *operational intensity* – the number of FLOPs that the application executes per byte of data transferred between the main memory and the CPU. The applications with low operational intensity perform small number of floating point operations per amount of data transferred from the memory and vice versa. They Y-axis of the model show the application performance (in FLOPs per second). The roofline first defines the performance bounds of a target compute node by plotting the memory bandwidth (inclined line) and the FLOPs roof (horizontal line), see Figure 9. The memory bandwidth and FLOPs roofs essentially show the system performance bounds for a given application operational intensity. Applications with low operational intensity are ultimately limited by the memory bandwidth roof (sloped), while the applications with high operational intensity are ultimately limited by the FLOPs roof (horizontal).

Then, the model determines the position, operational intensity and performance, of an application (or application segment) on the target compute node's roofline. Based on the application's position on the plot, we can determine its dominant bottleneck. Briefly, the application can be characterized as memory bandwidth or FLOPs bound, if it is close to the corresponding roof.

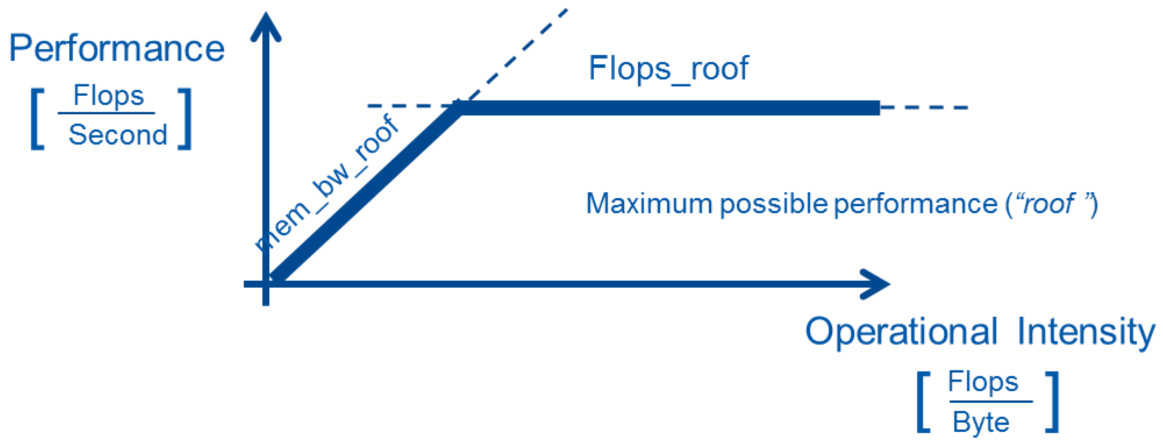

**Figure 9: Representation of the roofline model. The sloped line represents the performance ceiling limited by the memory bandwidth; the horizontal line represents performance ceiling limited by the platform floating point performance.**

## 5.2 Methodology: Plotting the roofline

**Theoretical vs. sustained performance:** The roofline model can be plotted based on the maximum theoretical or measured sustainable memory bandwidth and the floating point performance. Theoretical performance can be obtained from the product datasheets. Maximum sustainable values of the floating point performance and memory bandwidth are be measured by DGEMM and STREAM benchmarks, respectively. DGEMM [18] is a floating point intensive routine that calculates the product of double precision matrices:$C \leftarrow \alpha A \times B + \beta$. It represents Level 3 Basic Linear Algebra Subprograms (BLAS) routine. STREAM benchmark [19] performs operations on arrays that are several times larger than the last level cache, and therefore puts high pressure to the memory bandwidth. It comprises of four kernels: Copy, Add, Scale and Triad. In this deliverable, we report the results of the Triad operation, since it is the most similar to kernels used in HPC applications. In this document, we plot the roofline based on the maximum theoretical values and sustained values of the FLOPs and memory bandwidth. Also, based on the analysis presented in Section 4, we include the performance boundary for 80% of the maximum sustained memory bandwidth, indicating that applications positioned above these lines are strongly limited by the available memory bandwidth.

**Core vs. socket level:** In our experimental platform (and in a general case) each core has its private pipeline including floating point units, while the memory bandwidth is dynamically shared by all processes concurrently running on the socket. Therefore, we plot the FLOPs roof based on the theoretical performance of a single core, or measured (sustained) performance of the single-threaded DGEMM benchmark. Memory bandwidth is plotted based on the socket-level memory bandwidth, theoretical and sustained (multi-threaded STREAM execution, using all cores on the socket). Applications are profiled correspondingly – FLOPs are measured for each process independently, while memory bandwidth is always measured at the socket level, summing the memory traffic of all concurrently running processes.

**Measurement granularity:** We perform detailed FLOPs and memory bandwidth measurements at the CPU burst level, for all the application processes, covering several iterations of a main computation phase (see Section 2). For each CPU burst measurement, we plot a point on the roofline model. The point's transparency is reversely proportional to the duration of the burst – longer bursts are less transparent (darker) and vice versa.
Plotting the semi-transparent points also helps us to distinguish between a single point on a given position in the model (light color) and multiple overlapping points (dark color). Finally, we also plot the application position in the roofline based on its average FLOPs and memory bandwidth (light green triangle). We plot this point because, similarly to our memory bandwidth analysis, we want to emphasize that the analysis of the average values may lead to a misleading conclusions.

**Log-2 axis:** Model X-and Y-axis are plotted in the log2 scale, as in most of the studies that use the roofline model. This is because applications show a large range of operational intensity and performance (two or three orders of magnitude). To display such a variety of performance, it is easier on a log-log scale than on a linear scale.

## 5.3 Results

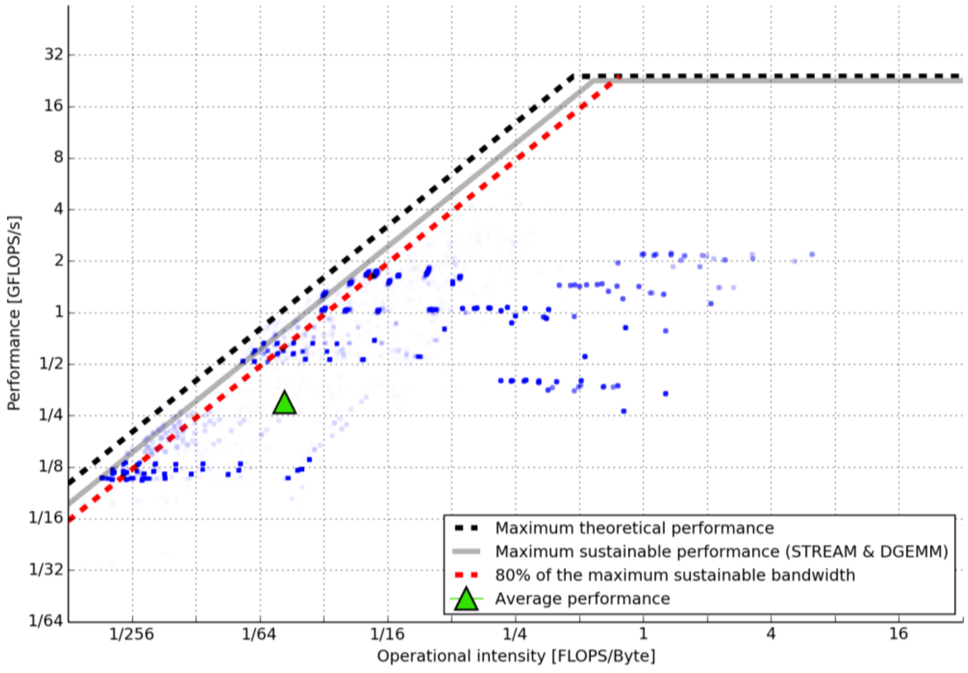

**Figure 10: ALYA application bursts on a roofline plot**

Figure 10 shows ALYA application bursts on a roofline model. As it can be seen, all the computational bursts are well below the FLOPs roof (remember the log scale), indicating that the core performance is severely underutilized. Various bursts are located above the inclined roof of the 80% and even 100% of the sustainable bandwidth, suggesting that they are clearly memory bandwidth bound.

Looking into the ALYA position based on the average FLOPs and memory bandwidth is again misleading – it does not detect the CPU bursts that are clearly memory bandwidth bound.

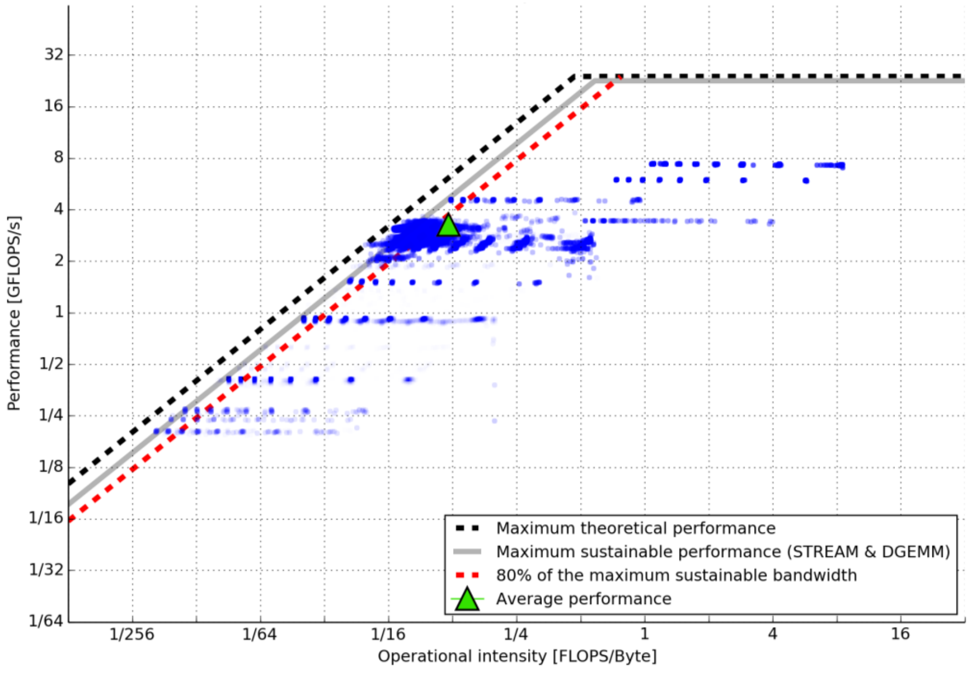

**Figure 11: BQCD application bursts on a roofline plot**

BQCD application bursts are displayed on Figure 11. As for ALYA, all the CPU bursts are well below the FLOPs roof, indicating that the core performance is severely underutilized. Numerous bursts are located above the inclined roof of the 80% and even 100% of the sustainable bandwidth, suggesting that they are clearly memory bandwidth bound.

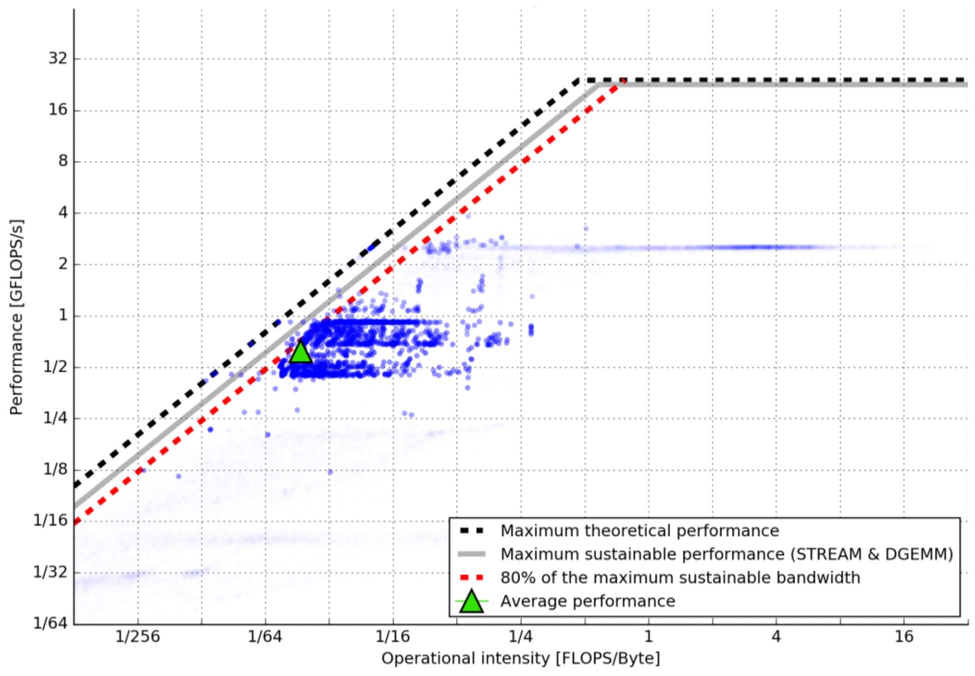

**Figure 12: NEMO application bursts on a roofline plot**

Similar as for ALYA and BQCD, but with less pressure to the memory bandwidth, is the NEMO application bursts on Figure 12. The bursts cluster is touching the 80% of the sustained bandwidth roof, with only few bursts that exceed it.

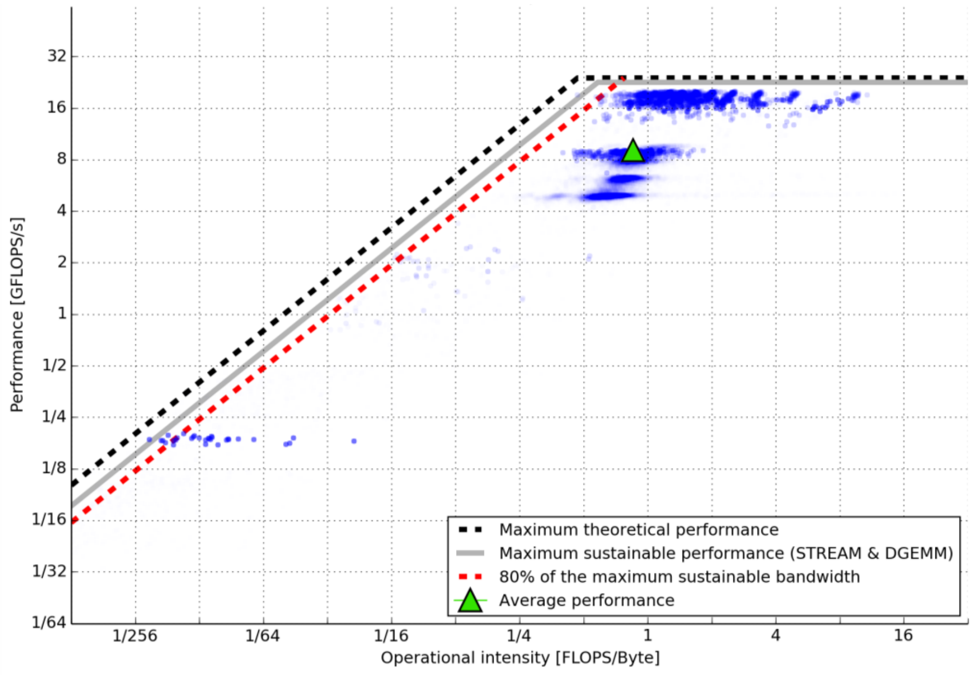

**Figure 13: CP2K application bursts on a roofline plot**

Figure 13 shows CP2K application bursts. Unlike other UEABS applications, CP2K shows low pressure to the memory (only few bursts exceed the 80% sustainable bandwidth roof). More pressure is put to the FLOPs, with one burst cluster that is touching the sustained FLOPs roof.

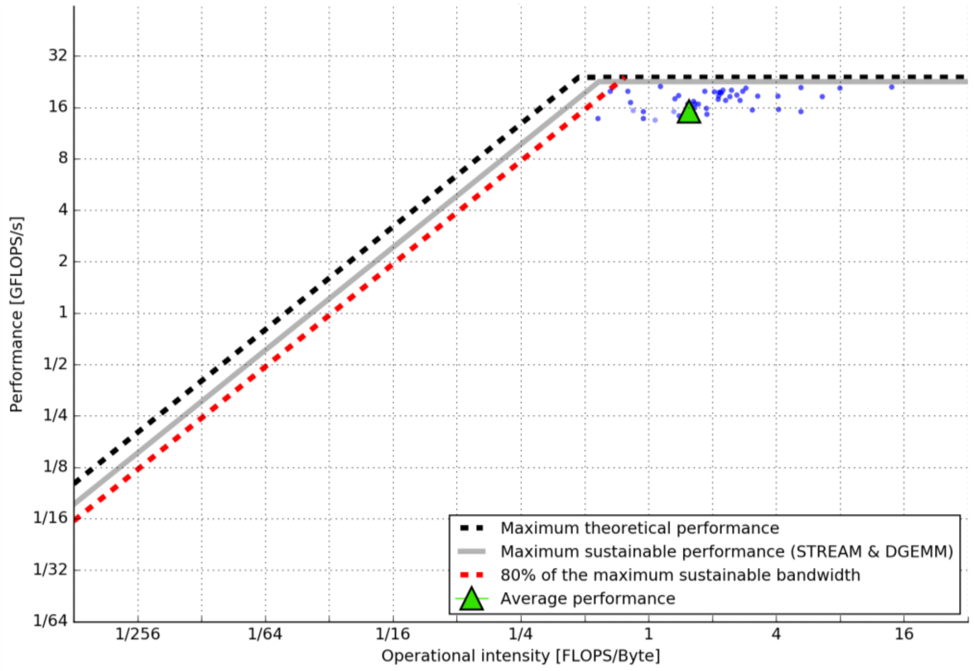

**Figure 14: HPL benchmark bursts on a roofline plot**

Bursts for HPL application are displayed on Figure 14. Since HPL is FLOPs-intensive benchmark, all the bursts are located close below the horizontal roof, some of them crossing the sustained FLOPs performance line. HPL is clearly CPU-bound benchmark.

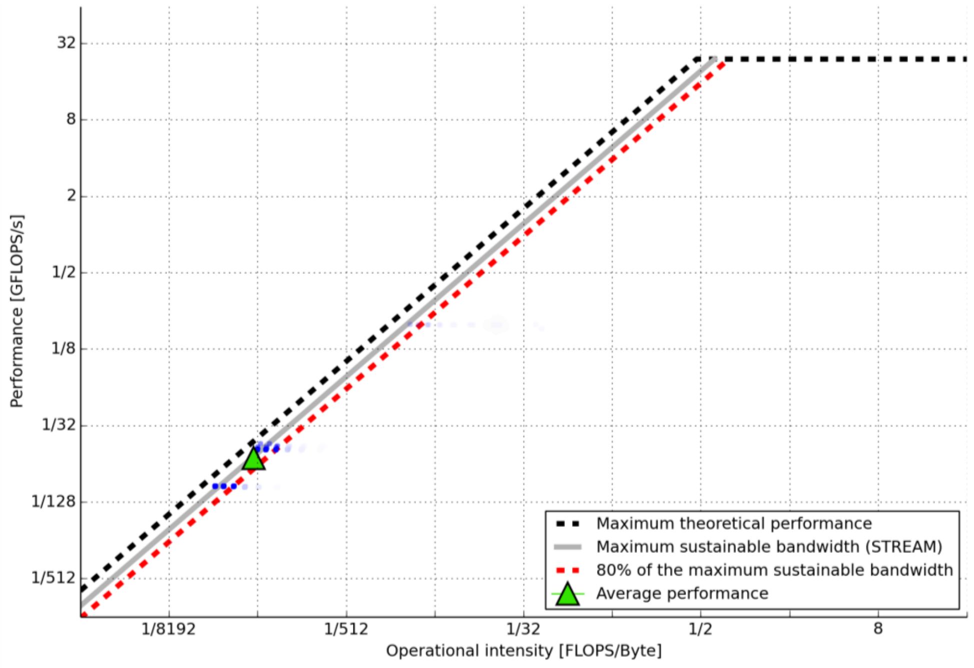

**Figure 15: HPCG benchmark bursts on a roofline plot**

Figure 15 shows HPCG application bursts. HPCG is not computationally intensive application while it puts a high pressure on memory bandwidth. Many bursts cross the 80% of the sustainable memory bandwidth boundary, while FLOPs values are low. Even the value based on average FLOPs and memory bandwidth is positioned above 80% of the maximum sustained memory bandwidth.

The analysis of application bursts when scaling out is displayed on Figure 16 and Figure 17. As in Section 4, because of the strong scaling and better utilization of the caches, bandwidth congestion decreases and some of the bursts from both applications move to the right, i.e. their operational intensity increases. We can also see that, for the same reason, some of the bursts move up, i.e. their GFLOPs performance increases. However, average performance is misleading again, because it shows a decrease of GFLOPs performance in both applications. The reason is the increment in inter-process communication time when scaling out, so end-to-end measurements fail to detect segments of high intensity for both GFLOPs performance and memory bandwidth. We noticed the same trends for the other applications under study.

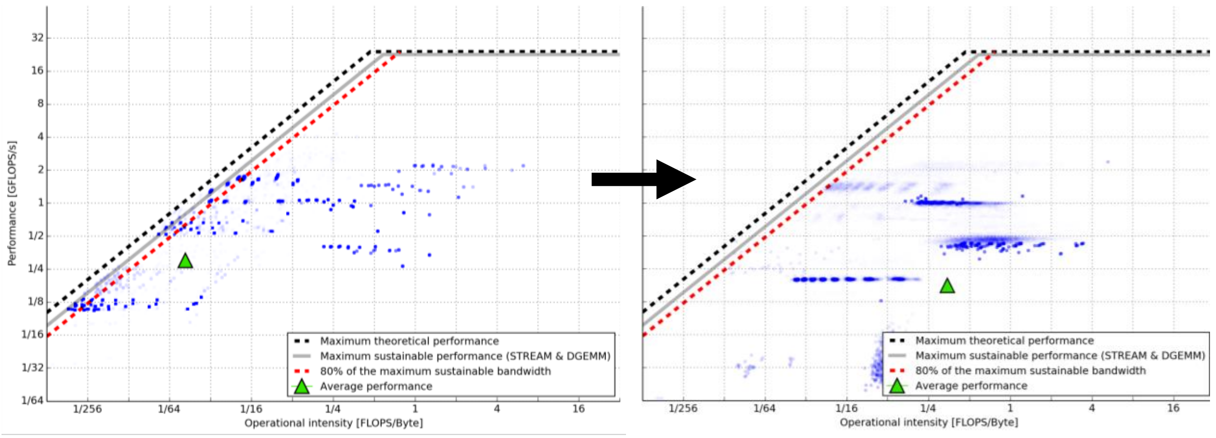

**Figure 16: ALYA application bursts when scaling out, from 16 (left) to 1024 (right) processes**

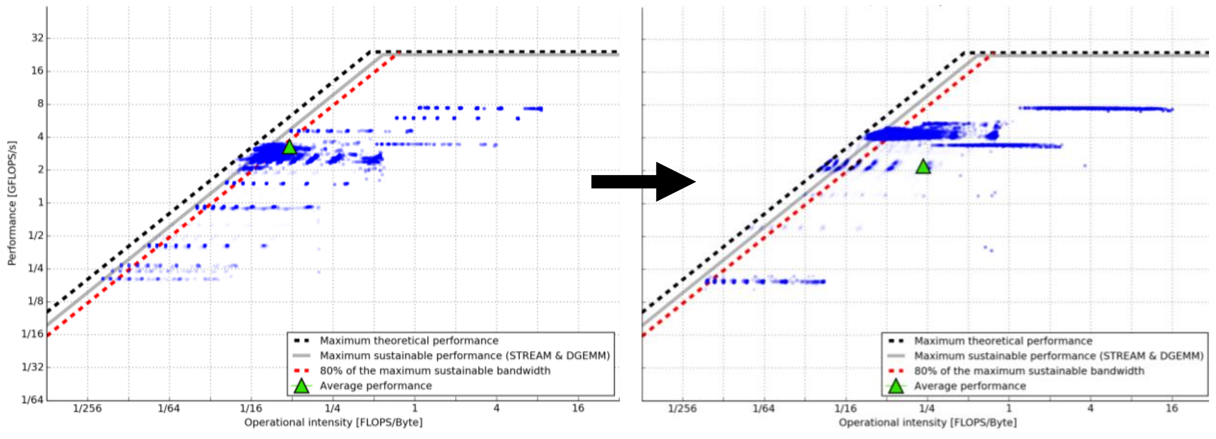

**Figure 17: BQCD application bursts when scaling out, from 64 (left) to 1024 (right) processes**

## 5.4 Summary

As in Section 4, the results confirm that the memory bandwidth presents a serious bottleneck, while FLOPs performance in most of the cases does not present a bottleneck for applications under study. Three out of four UEABS applications are far from horizontal roof while they reside close to the memory bandwidth roof and crossing the 80% of the sustainable memory bandwidth. When scaling out, due to strong scaling, on-chip caches are utilized better so operational intensity and GFLOPs performance of the applications increase. Because of the increment in inter-process communication, average values for memory bandwidth and GFLOPs performance are misleading and can show false trends.

# 6 Conclusions

Designing a balanced HPC system requires understanding of the dominant performance bottlenecks of present state-of-the-art HPC systems and workloads. In the context of the ExaNoDe Task 2.5, we analysed main performance bottlenecks in relevant industrial HPC applications on a main-stream HPC platform.

We base our analyses on three important bottlenecks in modern state-of-the-art HPC systems: percentage of execution time spent in inter-process communication, memory bandwidth congestion and FLOPs performance. The presented analysis points out the degree of performance degradation for each of the bottlenecks, while running realistic HPC workloads.

The main conclusions of our analysis are:

- Inter-process communication may account for a significant portion of the overall execution time, and may be one of the main limitations for the application scalability. Therefore, in the context of the project, it is important to deploy new approaches for the inter-process communication such as UNIMEM, and also to decrease the load imbalance, using various programming models which are present in WP3.

- The majority of applications considered in this report experience memory bandwidth congestion. Therefore, we recommend looking toward high-bandwidth memories in future ExaNoDe architectures.

- In addition to memory bandwidth congestion, it is important to understand FLOPs vs. memory bandwidth ratio, and which parameter is the dominant performance bottleneck. Our results show that majority of HPC applications is not bounded by the CPU processing power, i.e. that memory bandwidth is a dominant performance bottleneck. For future Exascale HPC platforms, we recommend building a balanced system, with higher Byte-per-FLOP ratio.

- The methodology used to measure application bandwidth and FLOPs utilization is very important. The analysis of average (end-to-end) measurements could mislead, and fail to detect applications that are bandwidth or FLOPs intensive in some of their segments, or show false trends.

# 7 References

[1] Barcelona Supercomputing Center, "MareNostrum III System Architecture," Technical Report, 2013.

[2] "Unified European Applications Benchmark Suite," [Online]. Available: www.prace-ri.eu/ueabs/.

[3] "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers.," [Online]. Available: http://www.netlib.org/benchmark/hpl/.

[4] "The HPCG Benchmark," [Online]. Available: http://www.hpcg-benchmark.org/.

[5] T. K. M. S. D. K. B. S. a. D. Q. R. Preissl, "Detecting Patterns in MPI Communication Traces," in *37th International Conference on Parallel Processing*, 2008.

[6] S. Y. a. J. D. B. Vien Dao, "Architectural support for reducing communication overhead in multiprocessor interconnection networks," in *Proceedings Third International Symposium on High-Performance Computer Architecture*, 1997.

[7] D. Böhme, "Characterizing Load and Communication Imbalance in Parallel Applications," Forschungszentrum Jülich GmbH, PhD Thesis, 2013.

[8] M. R. A. R. a. P. R. Milan Pavlovic, "LIghtweight MPI instrumentatiOn," in *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, 2015.

[9] Barcelona Supercomputing Center, "Extrae User guide manual for version 2.5.1," 2014.

[10] M. P. M. R. H. S. J. S. S. A. M. P. M. C. P. R. E. A. Darko Zivanovic, "Main Memory in HPC: Do We Need More, or Could We Live with Less?," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2017.

[11] M. H. a. P. L. Jack Dongarra, "The HPCG Benchmark," 2016. [Online]. Available: http://www.hpcg-benchmark.org/.

[12] J. J. D. a. M. A. Heroux, "Toward a New Metric for Ranking High Performance Computing Systems," Sandia Report SAND2013-4744, Sandia National Laboratories, 2013.

[13] J. G. a. C. W. G. Vladimir Marjanović, "Performance Modeling of the HPCG Benchmark," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, 2014.

[14] D. Z. D. R. B. R. d. S. S. A. M. P. R. a. E. A. Milan Radulovic, "Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC?," in *Proc. of the International Symposium on Memory Systems (MEMSYS)*, 2015.

[15] W. A. W. a. S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Computer Architecture*, 1995.

[16] B. L. Jacob, "The memory system: You can't avoid it, you can't ignore it, you can't fake it," *Synthesis Lectures on Computer Architecture*, 2009.

[17] A. W. a. D. A. P. S. W. Williams, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures," EECS Technical Report UCB/EECS-2008-134, 2008.

[18] J. D. C. S. H. a. I. S. D. J. J. Dongarra, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, 1990.

[19] J. D. McCalpin, "STREAM: Sustainable Memory Bandwidth in High Performance Computers," 1997. [Online]. Available: https://www.cs.virginia.edu/stream/ref.html.