Case Studies on the Impact and Challenges of Heterogeneous NUMA Architectures for HPC

Lilia Zaourar, Mohamed Benazouz, Ayoub Mouhagir, Carlos Falquez, Antoni Portero, Nam Ho, Estela Suarez, Polydoros Petrakis, Manolis Marazakis, Francesco Sgherzi, Ivan Fernandez, Romain Dolbeau, and Dirk Pleiter

[1] *Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France*
`{lilia.zaourar, mohamed.benazouz, ayoub.mouhagir}@cea.fr`
[2] *Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich GmbH, Jülich, Germany*
`{c.falquez, a.portero, n.ho, e.suarez}@fz-juelich.de`
[3] *Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH), Heraklion, Greece*
`{ppetrak,maraz}@ics.forth.gr`
[4] *Barcelona Supercomputing Center (BSC), Barcelona, Spain*
`{francesco.sgherzi, ivan.fernandez}@bsc.es`
[5] *SiPearl*, Rennes, France
`romain.dolbeau@sipearl.com`
[6] *KTH Royal Institute of Technology, Stockholm, Sweden*
`pleiter@kth.se`

Abstract. The memory systems of High-Performance Computing (HPC) systems commonly feature non-uniform data paths to memory, i.e. are non-uniform memory access (NUMA) architectures. Memory is divided into multiple regions, with each processing unit having its own local memory. Therefore, for each processing unit access to local memory regions is faster compared to accessing memory at non-local regions. Architectures with hybrid memory technologies result in further non-uniformity. This paper presents case studies of the performance potential and data placement implications of non-uniform and heterogeneous memory in HPC systems. Using the gem5 and VPSim simulation platforms, we model NUMA systems with processors based on the ARMv8 Neoverse V1 Reference Design. The gem5 simulator provides a cycle-accurate view, while VPSim offers greater simulation speed, with a high-level view of the simulated system. We highlight the performance impact of design trade-offs regarding NUMA node organization and System Level Cache (SLC) group assignment, as well as Network-on-Chip (NoC) configuration. Our case studies provide essential input to a co-design process involving HPC processor architects and system integrators. A comparison of system configurations for different NoC bandwidths shows reduced NoC latency and high memory bandwidth improvement when NUMA control is enabled. Furthermore, a configuration with HBM2 memory organized as four NUMA nodes highlights the memory bandwidth performance gap and NoC queuing latency impact when comparing local vs. remote memory accesses. On the other hand, NUMA can result in an unbalanced distribution of memory accesses and reduced SLC hit ratios, as shown with DDR4 memory organized as four NUMA nodes.

Keywords: Non-Uniform Memory Access (NUMA), co-design, simulation, High Performance Computing (HPC), benchmarking.

1 Introduction

High Performance Computing (HPC) has become increasingly relevant in various important fields, such as weather modeling, drug discovery, financial simulations, and genomics. Meeting the specific demands of these disciplines requires optimization and possibly customization of HPC resources. On the hardware side, this involves using multi-core CPUs, accelerators like GPUs or FPGAs, and even specialized AI processors to offer higher levels of processing power to execute complex simulations, data analytics, and machine learning tasks. HPC systems require substantial memory capacity and bandwidth to handle large datasets efficiently. High-bandwidth and low-latency networks are essential for fast data communication between nodes in a cluster or across different clusters for distributed computing. On the software side, HPC applications are designed to utilize parallel processing capabilities effectively, and systems should scale efficiently to handle larger workloads.

Various changes in hardware design, software optimization, and innovative architectural approaches have been proposed to overcome the limitations of current CPUs and Network On Chip (NoC) architectures regarding memory access performance. Among them, NUMA is the prevalent computer architecture design used in HPC systems. It is a memory architecture that provides multiple processors or computing nodes with access to a shared memory pool but with varying access latencies. System with NUMA control allows division of the memory space in distinct regions where for each CPU one region can be considered local. The CPUs or computing nodes are connected through an interconnect, which allows them to communicate and access the shared memory pool. However, the access time to the shared memory is not uniform for all processing units. Therefore, the key goal for NUMA control is to allow each processor or computing node to exploit faster access to its local region, rather than accessing far memory. Non-uniformity arises due to distance, latency, and bandwidth differences between the processor and the memory regions. NUMA architectures are designed to optimize memory access and minimize the impact of memory latency on performance. By placing the data closer to the processors that frequently access it, NUMA reduces the latency and enhances overall system performance. Therefore, NUMA offers substantial advantages in HPC situations where parallel computing and high memory bandwidth are crucial. It facilitates efficient memory sharing among multiple processors, minimizing latency and maximizing data locality. This leads to enhanced scalability, reduced communication overhead, and improved overall performance in HPC workloads. However, we should note that NUMA designs in HPC are a necessity, as the complexity of handling non-uniform memory accesses is substantial. More uniform designs for attaching memory would be preferable, but are more expensive and difficult to implement. In part it is also due to market realities. For HPC, single-socket designs would be a reasonable option (eg. as in BlueGene and Fugaku), but the much bigger cloud market dictates multi-socket (most commonly dual-socket) designs. Many compromises must be made to balance memory placement and computing resources to obtain good performance. It involves modeling these aspects at early design stages and benchmarking to perform efficient hardware/software co-design.

In this work, we model high-level and microarchitecture aspects to reflect the ability to take NUMA effects into account on a complex and representative processor architecture for HPC using the gem5 [13] and VPSim [5] simulation platforms. Then, we present case studies of the performance potential and data placement implications of NUMA in HPC systems using simulation models, in both gem5 and VPSim. These case studies provide input to a co-design process involving HPC System on a Chip (SoC) architects and system integrators.

2 Background

The concept of NUMA traces back to the 1990*s*. Historically, what is now understood as NUMA was called ccNUMA (Cache Coherent Non-Uniform Memory Access), and was commercially introduced in the Convex Examplar [23] and SGI Origin [12]. As the entire system is cache coherent, the non-uniformity in NUMA is one of performance. While all cores can access all memory, they observe different performance for different address ranges.

Up to the 2010s, NUMA was commonly a system-level or a board-level property. Introduced in the large systems of the 1990s, NUMA became a common feature in servers with the introduction of the AMD *Opteron* [10] and its integrated memory controller. Systems with more than one *Opteron* CPU would have a NUMA effect when a core in one CPU would access memory connected to the memory controller of a different CPU. Although this was not desirable from a software point of view, it made the available bandwidth scalable with the number of CPU, a highly desirable feature. Intel followed suit with the micro-architecture code-named *Nehalem* [19], and all multi-CPU systems since are NUMA. Modern operating systems are NUMA-aware, i.e. leverage for themselves and expose to user code the underlying NUMA architecture. With the ever-expanding number of cores inside a CPU, full homogeneity and symmetry of behavior inside a CPU is becoming difficult to maintain. The Intel micro-architecture code-named Haswell [7] introduced Cluster-on-Die [8, 16], which exposes two NUMA nodes in each CPU. It exposes a more explicit relationship between the cores and the two memory controllers inside a socket.

The Intel *Xeon Phi* 72*xx* (micro-architecture code-named Knights Landing) [21] was an even more explicit example, exposing up to four NUMA nodes inside the socket in its SNC-4 (Sub-NUMA Clustering) mode [11]. This micro-architecture did not support multi-socket systems, hence being the first NUMA design to be entirely single-socket. This architecture established HPC architectures with nodes that feature main memory tiers based on heterogeneous memory technologies. MCDRAM- and DDR4-based tiers could be used within a single address space. With such a hybrid-memory design, existing mechanisms of NUMA-awareness in system software could be applied to control non-uniformity and heterogeneity in the memory system.

This evolution of NUMA is not a requirement from software - a uniform system behavior is much easier for most users to deal with. It is the result of hardware implementation requirements, which, when too strong, must be exposed to higher levels (firmware, operating system, user code) so that they can be taken into account by software. Historical Symmetric MultiProcessors (SMPs) would put all the CPUs on a shared bus on which the memory controller also connected, creating a uniform memory architecture. Early large systems needed a dedicated network connecting multiple shared-bus to support coherency between groups of CPU. From the AMD *Opteron* onward, this shared bus was removed in favor of integrated memory controllers and dedicated socket-to-socket communication channels (HyperTransport, QuickPath Interconnect).

The examples of Haswell and Knights Landing are a prelude to the current situation, where the network required by the large systems of the 90s is now needed inside each CPU to connect the cores. The number of cores in a single CPU has become so large that the distance (topological and physical) between a core and a memory controller has become significant and cannot be uniform. The bandwidth requirements are also so large that multiple memory controllers are also needed in each socket. Thus, there are many possible (core, memory controller) pairs, each with its own latency and specific bandwidth bottlenecks. A design like the Ampere Altra offers 80 cores. It can expose their 8 memory channels as four NUMA nodes inside a single socket. The advent of in-package High Bandwidth Memory (HBM) significantly increases available bandwidth, but also exacerbates the effect of NUMA inside a socket. The Fujitsu $A64FX$ processor [18] uses in-package HBM2 memory and always exposes a NUMA node per HBM stack, four in total. Exploiting such memory resources requires careful management of potential bandwidth bottlenecks. It also involves optimization of latency, as excessive latency limits the usable bandwidth.

Despite the substantial benefit in sharing the memory capacity of all nodes, the relative distances of cores and memory elements lead to a performance challenge. The Linux kernel manages memory movement in NUMA systems based on the `node distance` mechanism, which has been proven to have several inefficiencies [4, 24]. The `node distance`s are provided by the system manufacturer via the System Locality Information Table (SLIT) table, hence being a hard-coded value specified at boot-time suggesting an ordering between NUMA node distances, rather than an actual distance. Moreover, `node distance` values do not consider memory characteristics. Today's CPUs support several memory-like devices that can be attached at will and serve as additional memory [17, 26], as well as supporting various Dynamic RAM (DRAM) kinds as main memory [20]. Blindly trusting `node distance` implies that the operating system might decide that using an HBM stack in a *far* NUMA node might be less beneficial than using a *near* Double Data Rate (DDR), even if HBM would yield higher bandwidths for streaming-like accesses. The introduction of HMAT (Heterogeneous Memory Attribute Table [1]) in the ACPI (Advanced Configuration and Power Interface) standard might help by providing more comprehensive information in the future, but they are not yet commonly available in hardware platforms.

3 NUMA architecture: modeling and exploration

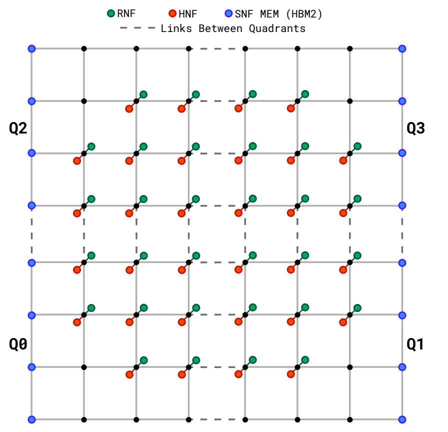
We consider in this work a General Purpose Processor (GPP) based on the ARMv8 Neoverse V1 Reference Design, with $2x256$ bits Scalable Vector Extension (SVE) instructions [22]. The core has separate 64 KiB L1 data and instruction caches and a private unified data and instruction 1 MiB L2 cache. It is interconnected through a $8x8$ Mesh NoC of four Quadrants (NUMA nodes), with 64 CPU@2.4 GHz cores, 2 cores per Request Node (RNF), for a total of 16 per Quadrant, 1 MiB System Level Cache (SLC) slices, 2 slices per Home Node (HNF), and 32 HBM2@1.6 GHz channels, (8 per Quadrant, with a combined bandwidth of 307.2 GB/s), as shown in Figure 1(a). The NoC@2.0 GHz follows the AMBA-CHI protocol and has four Virtual Networks (VNETs), and one link per VNET, with 64B link width and one packet per cycle data rate. For DDR4 memories, we reuse the models already available in gem5, with the latency of links connecting two Quadrants (inter-NUMA links) set to 3 cycles. In the two following subsections, we present how the reference system can be modeled in two simulators: gem5 and VPSim. This approach of combining the two simulation platforms, as presented in [27] has been extended to model NUMA-related aspects and assess the concrete impact on both HW and SW performance Indeed, the gem5 simulator models computer system architecture at cycle-level accuracy, but at the cost of increased simulation time. While VPSim's accuracy is diminished, a trade-off exists between simulation time and supporting simulation of larger systems. We demonstrate how gem5 and VPsim can be used in a complementary manner to study co-design concerns.

3.1 Modeling NUMA topologies with gem5

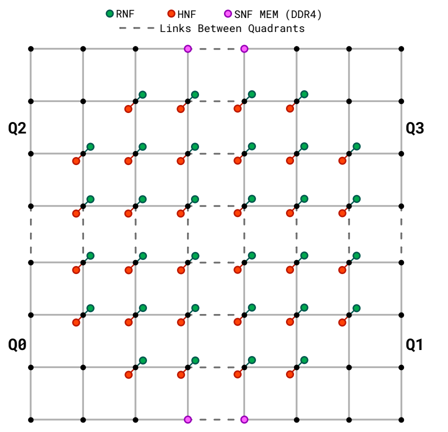
We have implemented the capability of modeling distinct NUMA node topologies with the gem5 simulator [13]. A definite NUMA topology can be defined by:

– Labeling each CPU and memory controller with a distinct NUMA identifier,
– Defining groups of SLCs (ARM System Cache Groups [2]) assigned to specific memory controllers.

Inter-node communication latencies when traversing NUMA boundaries can be modeled by adjusting the latency of cross-NUMA links (links connecting two Mesh routers that belong to adjacent Quadrants). Experimental results from this GPP are shown in Section 4.



((a)) gem5: 4 Quadrants/NUMA nodes GPP



((b)) VPSim: 4 Quadrants/NUMA nodes GPP

Fig. 1: NoC Designs

3.2 NUMA modeling in VPSim

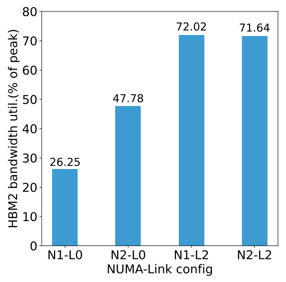
We brought NUMA support to VPSim environment, which involved modifying several parts of the tool. The custom QEMU machine used to support cores simulation in VPSim [9] has been extended to expose NUMA systems to the guest Operating System (OS). Depending on the components of the modeled platform, VPSim automatically generates device tree files. The generator has been updated for correct transcription of CPUs and memory address spaces bindings to NUMA nodes. As for the memory hierarchy modeled in SystemC, most work was naturally ported to VPSim NoC performance model [14]. This model has a whole view of the connected components and, as such, centralizes all information about NUMA configuration. It was extended with NUMA support to enable redirecting requests to the right components, such as SLCs, and DDR memory controllers. Moreover, cache group functionality was also implemented in VPSim. It can be turned on or off when NUMA support is activated. For benchmarking purposes, counters were added to quantify inter-node communications. For each couple *(source, destination)* of NUMA nodes, the number of packets crossing from *source* to *destination* is monitored. We also keep track of Inter-node memory accesses, *i.e.* DDR controllers' accesses coming from other NUMA nodes than the one to which the DDR controller is bound.

4 Evaluation

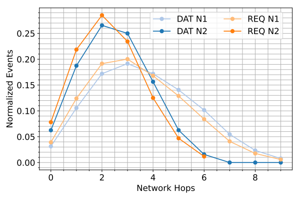
In this section, we evaluate the impact on the network performance of different NUMA node and SLC group assignments. Consider the NoC divided into 4 quadrants as shown in Figure 1(a). A *single NUMA node* topology defines a single NUMA node containing all quadrants, with all CPU and memory controllers labeled with the same NUMA id, and SLC slices interleaved over the available memory controllers. A *4-NUMA node* topology defines each quadrant as a local NUMA node, with each CPU and memory controller labeled according to which quadrant it belongs to. Furthermore, all SLCs within a quadrant are assigned the memory address range corresponding to the memory controllers in the quadrant. This section uses the STREAM-TRIAD benchmark [15] (with SVE vectorization, and parallelized using OpenMP directives) to characterize the performance of different configurations. The selected problem size is 10 million double-precision elements per array, i.e. 228.88 MiB for all three arrays.

4.1 Performance for different NoC bandwidths

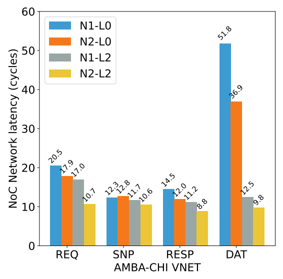
Here we discuss the effect on memory bandwidth and network performance of distinct NUMA topologies with varying inter-router NoC bandwidth. We restrict our simulations to the lower 2-Quadrant subset of the 4-Quadrant NoC, either as a single NUMA node or 2-NUMA node topology, with each quadrant defining a local NUMA node as described before. We label these configurations *N1* and *N2*, respectively. We consider two distinct NoC configurations: one with a single link carrying all VNET packets, labeled *L0*, and one with 2 links for each VNET (for 8 links in total), labeled *L2*. We run a 32-thread STREAM-TRIAD benchmark over all available CPUs to quantify the effect of different NoC and NUMA configurations on memory bandwidth and network performance.



((a)) HBM2 Bandwidth utilization



((b)) Network Hops



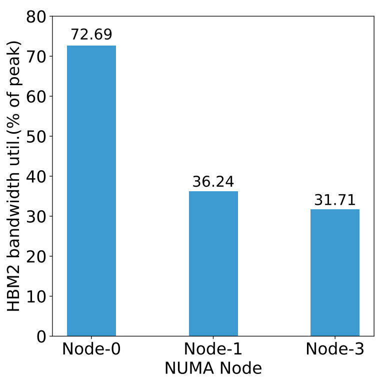
((c)) Network latency per VNET

Fig. 2: STREAM-TRIAD - (1 vs 2 NUMA nodes, 1 vs 8 links)

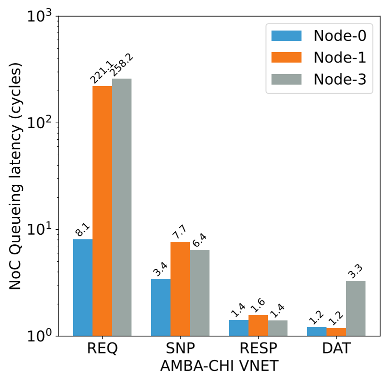
The HBM2 memory bandwidth utilization for the 4 possible configurations is shown in Figure 2(a). For the single-link configurations L0, NUMA partitioning causes a great improvement in available memory bandwidth. For the 8-link configurations L2, NUMA partitioning does not seem to have an effect on the already saturated memory bandwidth utilization. Figure 2(b) shows the effect on Data (DAT) and Request (REQ) VNETs *network hop distribution* for the N1 and N2 NUMA topologies. NUMA partitioning clearly shifts the hop distribution to the left since the network traffic becomes mostly constrained to its local quadrant. The effects of this traffic reduction can also be seen on the *average network packet latency*, shown in Figure 2(c). The average network packet latency T_N is defined as $T_N \equiv T_a - T_i$, where T_i is the time the packet was injected in the network, and T_a is the time the packet arrived at its destination node. For all protocol VNETs, the partitioned NUMA topology N2 always has a reduced network latency compared to the single-node configuration N1, the effect being greater for the lower bandwidth configurations.

4.2 STREAM TRIAD on a 4-NUMA node GPP

We now model a complete 4-NUMA nodes GPP, with 64 cores, 64 SLC slices, and 32 HBM2 controllers, as depicted in Figure 1(a). We run STREAM-TRIAD with 16 threads, always bound on Node-0 (Q0 in Figure 1(a)), and with the allocated memory attached on a different NUMA node each time. We examine three different cases. Initially, the allocated memory buffer is placed on Node-0 (the same node as the OpenMP threads), then on Node-1 (Q1), and finally on Node-3 (Q3), which is the furthest away from Node-0. By allocating the buffer on multiple NUMA nodes, we can measure the impact of memory locality on memory bandwidth and NoC latency.



((a)) HBM2 bandwidth utilization



((b)) NoC Queueing latency per VNET

Fig. 3: STREAM-TRIAD, 4 NUMA nodes

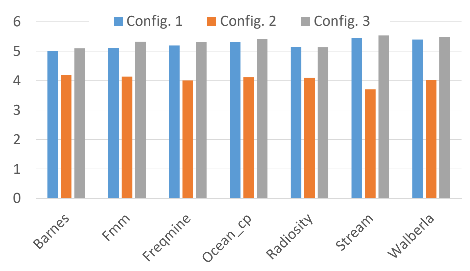
As shown in Figure 3(a), 16 OpenMP STREAM-TRIAD threads can utilize 72.69% of the peak HBM2 bandwidth when the buffer is placed in Node-0. This utilization drops down to 36%, when the buffer is moved to Node-1 and only 32% when the buffer is placed on Node-3. Regarding the NoC performance concerning the allocated buffer location, we examine the NoC Queueing latency for all four AMBA-CHI VNETs (Request, Snoop, Response, Data), as shown in Figure 3(b). When the buffer is placed on NUMA Node-0, the average Request (REQ) VNET Queueing latency is about 8 cycles. When the buffer is moved to Node-1 and Node-3, the REQ VNET Queueing latency is increased to 221 cycles and 258 cycles, respectively, signifying a more congested NoC. For the remaining three VNETs, the NoC Queueing latency increase is insignificant. From these two graphs, we can clearly see the impact of data locality on the memory bandwidth performance and the NoC latency increase, for a specific VNET. The simulation tools allow us to have a clear view of the underlying subsystems performance and limitations. This information is then communicated to the hardware architects in order to improve their models accordingly.

4.3 Application performance characterization using VPSim

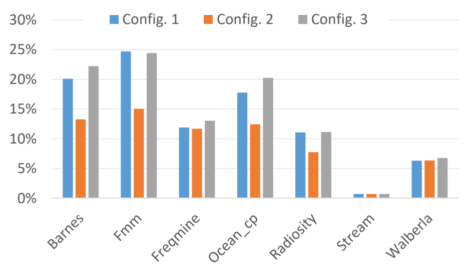
In the following, we demonstrate the ability of high-level virtual prototyping tools, such as VPSim, in analyzing the impact of enabling features, such as NUMA support and SLCs grouping, on memory hierarchy performance. The objective is to help software developers make aware choices at early stages of the hardware design. Thanks to the fast simulation speed of VPSim, we consider a large set of HPC-oriented benchmarks consisting of multi-threaded applications from the PARSEC [3] and SPLASH-2 [25] suites as well as two OpenMP applications: STREAM and WaLBerla's UniformGrid benchmark [6]. For OpenMP applications, OpenMP directives were employed to bind 64 threads on a one-to-one basis to the 64 cores. Experiments are run under the default NUMA local memory allocation policy (no `numactl` directives), known as first-touch policy, *i.e.* threads will preferably allocate memory in their own NUMA node's memory.

While awaiting for the integration of a new HBM model into VPSim, we simulated the same setup as previously, except that instead of HBM stacks, we considered 4 DDR memory controllers (one per Quadrant) as depicted in Figure 1(b). Since the emphasis is on analyzing data locality and accesses across NUMA nodes, replacing HBM with DDR should not alter the observations and the outcomes exposed in this section. Three configurations were compared. In Configuration 1, NUMA support is disabled, and the single memory address space is interleaved among all SLCs and DDR memory controllers. Configuration 2 enables NUMA support. Four homogeneous NUMA nodes are considered. The physical address space is then segmented equally into 4 contiguous memory regions. Cache groups functionality is also enabled; *i.e.* each memory region is interleaved among the SLC slices of its NUMA node. Interleaving at DDR level is not applicable since each node has only one DDR controller. Configuration 3 resembles Configuration 2 except SLCs are not grouped. Thus, the whole physical address space is interleaved among all SLC slices, similar to Configuration 1.

Figure 4(a) depicts the impact of our configurations on the average distance of NoC packets. The behavior is consistent no matter the tested application. When NUMA support is enabled, local allocation policy reduces up to 32% of



((a)) Average Distance of Packets (hops)



((b)) SLC Hit ratio (%)

Fig. 4: NoC Designs

the average distance of packets under the condition that SLCs are also grouped per NUMA node (Configuration 2). Otherwise, if SLC cache groups functionality is disabled (Configuration 3), this benefit is lost and we measure comparable or slightly higher values than those of Configuration 1, despite having DDR controllers in NUMA mode. Figure 5 shows the impact of the three configurations on

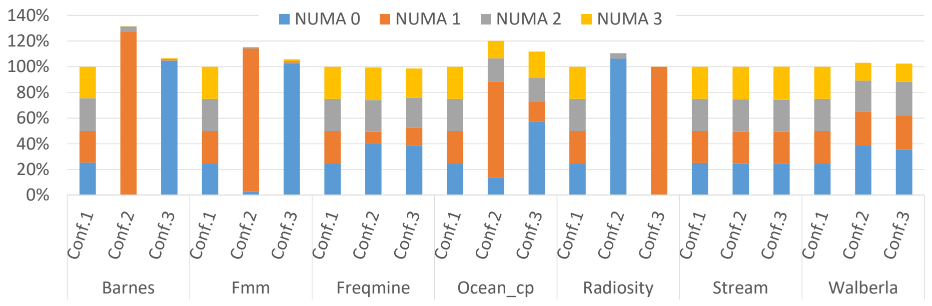


Fig. 5: DDR Memory Activity (Reads and Writes). Values are normalized to the result of Configuration 1.

DDR memory activity. This includes read and write operations. For each application, values are normalized to Configuration 1 result. No matter the application, the activity is well distributed among the four DDR controllers when interleaving is at work (Configuration 1). As such, interleaving offers an easy way to fully exploit the available memory bandwidth. On the other hand, when NUMA is enabled, unless data allocation is well distributed between threads (such as STREAM and WaLBerla), multi-threaded applications can hardly exploit DDR controllers equally. Indeed, NUMA might result in an unbalanced usage of the memory address space. For three out of the seven benchmarks (Barnes, Fmm, Radiosity), more than 95% of accesses are concentrated on one memory region, the one bound to the NUMA node running the main thread. Concentrating load on a single DDR controller instead of 4 increases the average access latency.

These poorly behaving benchmarks are known for optimal data distribution being challenging [25]. By clustering threads into separate nodes, a NUMA architecture make it even more noticeable. In order to draw more performance out of these benchmarks, we may be tempted to have recourse to software enforced interleaving using `numactl --interleave=all` directive. Eventually, this will balance the usage of DDR memory controllers, and we will observe similar performance to Configuration 1. However, spreading memory accesses across controllers also negates the benefit of NUMA locality, as three out of four memory accesses will cross NUMA nodes boundaries. A better solution would be to create NUMA-aware versions of such benchmarks.

In a different register, we observe for some applications up to 31% of increase in DDR memory activity when both NUMA and SLC cache groups are enabled (Configuration 2). More than 69% and up to 96% of this increase is attributed to additional read operations, which indicates a lower caching efficiency at the SLC level. This is confirmed by analyzing SLC hit ratio as shown in Figure 4(b).

There is a clear correlation between the decrease in hit ratio and the increase in DDR memory activity. Grouping SLCs has the effect of reducing the total cache size available per NUMA node (16MB against 64MB), which results in lower hit rates (up to 39%). This behavior is more pronounced for unbalanced applications with memory activity concentrated in fewer NUMA nodes. Data are evicted more frequently due to cache line replacements, forcing the system to read them again from DDR memory the next time they are accessed.

According to these observations, applications can be classified depending on whether they benefit from NUMA or not. Embarrassingly parallel applications such as STREAM, as well as quite balanced applications that are not affected by the reduced size of cache groups, such as Freqmine and WaLBerla, benefit most from NUMA locality. Thus, these applications are better executed using Configuration 2. Highly unbalanced applications, such as Barnes and Fmm, for which the burden of memory accesses is almost fully transferred to one NUMA node, see their performance decreasing because of reduced available bandwidth per NUMA combined with an increase in DDR memory accesses due to higher miss rate compared to the non-NUMA configuration. Such applications are better executed on Configuration 1 or by enforcing DDR memory interleave using `numactl --interleave=all` directive. Overall, if NUMA support is enabled, we advise against disabling cache group functionality (Configuration 3). Even though turning off this functionality restores the SLC hit ratio and reduces DDR memory activity, it nullifies the effect of local allocation on the packet's average distance without being able to restore a balanced usage of DDR controllers.

4.4 Memory Page Migration in NUMA systems

Since automatic kernel-level management is potentially unaware of memory characteristics, users have to manually managing thread and memory placement using utilities like `libnuma`, `numactl`, `OpenMP 5.0 Memory Allocators` or higher-

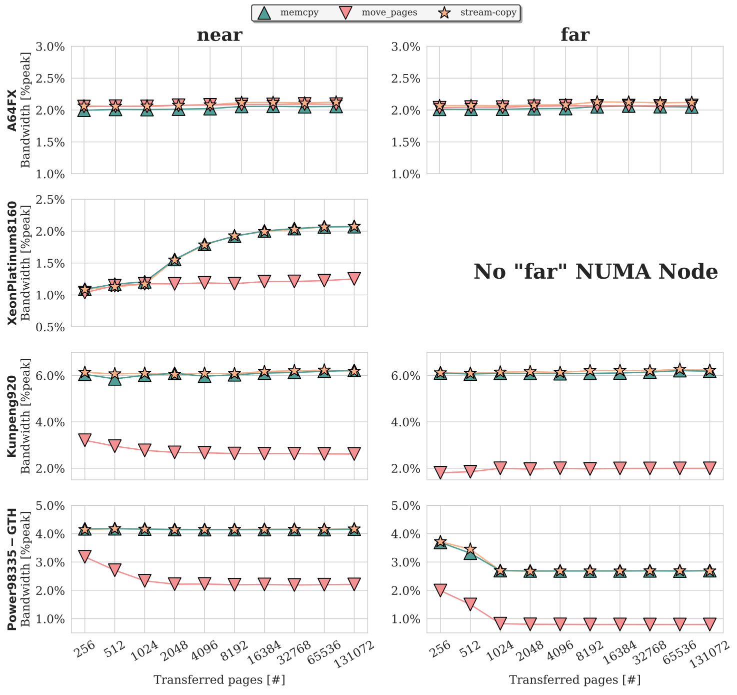


Fig. 6: Percentage of peak interconnect bandwidth utilized on 4 NUMA-enabled platforms when transferring data from one NUMA node to another.

level wrappers like `memkind` [4]. While these approaches work if the optimal placement of computational elements is known in advance, they are inapplicable when data movement between NUMA domains is required. To outline the pitfalls of these approaches and the relevance of appropriate NUMA placement, we benchmark several NUMA platforms on a workload that involves moving pages from one domain to another. The benchmark tests three methods for transferring data: `libc`'s `memcpy`, the `move_pages` syscall, and implementation of `stream-copy` using vector intrinsics. All the benchmarks are run on a single core, transferring a range between 2^8 and 2^{17} pages of 64 KiB. Figure 6 displays the percentage of peak theoretical interconnect bandwidth achieved in moving pages from one NUMA node to another in 4 HPC systems. Three of those systems (A64FX, Kunpeng920, Power98335-GTH) present several NUMA nodes at different `node_distance`s. In this setting, we refer to pairs of nodes with the lowest nonzero distance as *near* and, conversely, as *far* if they are the furthest apart. On the other hand, the Xeon Platinum 8160 platform we test has just two NUMA nodes. Hence it has no *far* pair of nodes. Userspace and kernel space management approach incur high overhead, stemming from either (i) having to resort to syscalls (`move_pages`) or (ii) being limited by the number of cores that perform the transfer (`memcpy`). Consequently, current memory management methods cannot perform effective data transfer between NUMA nodes, with data needing to pass through the core being the limiting factor.

5 Conclusion and perspectives

In this paper, we study the performance potential and data placement implications of NUMA in HPC systems. We model a NUMA system architecture with processors based on the ARMv8 Neoverse V1 Reference Design, using the gem5 and VPSim simulation platforms. We then present several case studies to evaluate the performance impact of different NUMA node and SLC group assignments, as well as NoC configuration settings, on microbenchmarks and applications. These case studies, highlight design trade-offs, and serve as input for a co-design process involving HPC SoC architects and system integrators.

The innovation of this paper in regard to gem5, is the modeling of large ARM systems (64 cores), with NUMA capability (NUMA capability was added in gem5 source code in February 2022) and Full System simulation (Linux OS and complete software stack). Exploration of such large systems modeled in gem5, is hard to find in existing literature. Usually, such large systems (16 or more CPU cores) are based on trace-driven simulations, whereas we are making use of a very detailed cycle accurate Out of Order (O3) Processor model, with SVE capability. With a complete GPP chip modeled, our work uncovers performance effects at the NoC level that are less pronounced with smaller configurations.

While we can rely on already existing hardware to do some of the observations described in this paper, we demonstrated the ability of virtual prototyping tools to study the impact of varying design choices. Besides, bringing NUMA capability into these tools is a key milestone towards the exploration of next generation platforms with more advanced and complex memory accesses schemes.

Finally, we envision hardware architects to use both VPSim and gem5 together within the context of co-designing a new CPU. On one side, the rapid simulation capabilities and high scalability of VPSim allows for swift exploratory studies, enabling architects to quickly improve upon their initial design and validate the tooling (i.e., operating systems, drivers and software) that will be available on the final CPU. On the other side, once the architectural structure reaches the final stages, the accuracy of gem5 becomes paramount in determining performance metrics and assessing if the design meets performance expectations. Using the two simulators jointly overcomes the shortcomings of both, while providing the accuracy and flexibility required in simulating modern NUMA systems.

Acknowledgment

This research has received funding from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 (European Processor Initiative) and Specific Grant Agreement No 101036168 (EPI SGA2). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland. The EPI-SGA2 project, PCI2022-132935 is also co-funded by MCIN/AEI /10.13039/501100011033 and by the UE NextGenerationEU/PRTR.

References

1. ACPI HMAT. `https://uefi.org/htmlspecs/ACPI_Spec_6_4_html/05_ACPI_Software_Programming_Model/ACPI_Software_Programming_Model.html`.
2. ARM. Neoverse CMN-650 Technical Reference manual. `https://developer.arm.com/documentation/101481/0200?lang=en`, 2023.
3. C. Bienia, S. Kumar, J. Pal Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, page 72–81, 2008.
4. Christopher Cantalupo, Vishwanath Venkatesan, Jeff Hammond, Krzysztof Czurlyo, and Simon David Hammond. memkind: An extensible heap memory manager for heterogeneous memory platforms and mixed memory policies. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2015.
5. Amir Charif, Gabriel Busnot, Rania Mameesh, Tanguy Sassolas, and Nicolas Ventroux. Fast Virtual Prototyping for Embedded Computing Systems Design and Exploration. In *Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '19, pages 1–8. Association for Computing Machinery, January 2019.
6. C. Feichtinger, S. Donath, H. Köstler, J. Götz, and U. Rüde. Walberla: Hpc software design for computational engineering simulations. *Journal of Computational Science*, 2(2):105–112, 2011.
7. Per Hammarlund, Alberto J Martinez, Atiq A Bajwa, David L Hill, Erik Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, et al. Haswell: The fourth-generation intel core processor. *IEEE micro*, 34(2):6–20, 2014.
8. Johannes Hofmann, Dietmar Fey, Jan Eitzinger, Georg Hager, and Gerhard Wellein. Analysis of intel2019s haswell microarchitecture using the ecm model and microbenchmarks. In *Architecture of Computing Systems–ARCS 2016: 29th International Conference, Nuremberg, Germany, April 4–7, 2016, Proceedings 29*, pages 210–222. Springer, 2016.
9. Fatma Jebali, Oumaima Matoussi, Arief Wicaksana, Amir Charif, and Lilia Zaourar. Decoupling processor and memory hierarchy simulators for efficient design space exploration. In *System Engineering for constrained embedded systems*, pages 47–52. 2022.
10. Chetana N Keltcher, Kevin J McGrath, Ardsher Ahmed, and Pat Conway. The amd opteron processor for multiprocessor servers. *IEEE Micro*, 23(2):66–76, 2003.
11. Ruben Laso, Francisco F Rivera, and José Carlos Cabaleiro. Influence of architectural features of the snc-4 mode of the intel xeon phi knl on matrix multiplication. In *Computational Science–ICCS 2019: 19th International Conference, Faro, Portugal, June 12–14, 2019, Proceedings, Part V 19*, pages 483–490. Springer, 2019.
12. James Laudon and Daniel Lenoski. The sgi origin: A ccnuma highly scalable server. *ACM SIGARCH Computer Architecture News*, 25(2):241–251, 1997.
13. Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann,

Srikant Bharadwaj, et al. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152*, 2020.

14. Oumaima Matoussi. Noc performance model for efficient network latency estimation. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 994–999. IEEE, 2021.
15. JD McCalpin. Memory bandwidth and machine balance in high performance computers, 1995.
16. Daniel Molka, Daniel Hackenberg, Robert Schöne, and Wolfgang E Nagel. Cache coherence protocol and memory performance of the intel haswell-ep architecture. In *2015 44th International Conference on Parallel Processing*, pages 739–748. IEEE, 2015.
17. S Park, H Kim, KS Kim, J So, J Ahn, WJ Lee, D Kim, YJ Kim, J Seok, JG Lee, et al. Scaling of memory performance and capacity with cxl memory expander. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–27. IEEE Computer Society, 2022.
18. Mitsuhisa Sato, Yutaka Ishikawa, Hirofumi Tomita, Yuetsu Kodama, Tetsuya Odajima, Miwako Tsuji, Hisashi Yashiro, Masaki Aoki, Naoyuki Shida, Ikuo Miyoshi, et al. Co-design for A64FX manycore processor and" fugaku". In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.
19. Ronak Singhal. Inside intel next generation nehalem microarchitecture. In *Hot Chips*, volume 20, page 15, 2008.
20. Avinash Sodani. Knights landing (knl): 2nd generation intel® xeon phi processor. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–24. IEEE, 2015.
21. Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. Knights landing: Second-generation intel xeon phi product. *Ieee micro*, 36(2):34–46, 2016.
22. Nigel Stephens, Stuart Biles, Matthias Boettcher, Jacob Eapen, Mbou Eyole, Giacomo Gabrielli, Matt Horsnell, Grigorios Magklis, Alejandro Martinez, Nathanael Premillieu, et al. The ARM scalable vector extension. *IEEE micro*, 37(2):26–39, 2017.
23. Radhika Thekkath, Amit Pal Singh, Jaswinder Pal Singh, Susan John, and John Hennessy. An evaluation of a commercial cc-numa architecture-the convex exemplar spp1200. In *Proceedings 11th International Parallel Processing Symposium*, pages 8–17. IEEE, 1997.
24. Sean Williams, Latchesar Ionkov, and Michael Lang. Numa distance for heterogeneous memory. In *Proceedings of the Workshop on Memory Centric Programming for HPC*, pages 30–34, 2017.
25. S. Cameron Woo, M. Ohara, E. Torrie, J. Pal Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. *SIGARCH Comput. Archit. News*, 23(2):24–36, may 1995.
26. Lingfeng Xiang, Xingsheng Zhao, Jia Rao, Song Jiang, and Hong Jiang. Characterizing the performance of intel optane persistent memory: A close look at its on-dimm buffering. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 488–505, 2022.
27. Lilia Zaourar, Mohamed Benazouz, Ayoub Mouhagir, Fatma Jebali, Tanguy Sassolas, Jean-Christophe Weill, Carlos Falquez, Nam Ho, Dirk Pleiter, Antoni Portero, et al. Multilevel simulation-based co-design of next generation hpc microprocessors. In *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 18–29. IEEE, 2021.