

# Scaling the Capacity of Memory Systems; Evolution and Key Approaches

**DOI:**
10.1145/3357526.3357555

**Document Version**
Accepted author manuscript

# Scaling The Capacity of Memory Systems; Evolution and Key Approaches

Kyriakos Paraskevas
School of Computer Science
University of Manchester
United Kingdom
kiriakos.paraskevas@manchester.ac.uk

Andrew Attwood
School of Computer Science
University of Manchester
United Kingdom
andrew.attwood@manchester.ac.uk

Mikel Luján
School of Computer Science
University of Manchester
United Kingdom
mikel.lujan@manchester.ac.uk

John Goodacre
School of Computer Science
University of Manchester
United Kingdom
john.goodacre@manchester.ac.uk

## ABSTRACT

The demand on memory capacity from applications has always challenged the available technologies. It is therefore important to understand that this demand and the consequential limitations in various aspects led to the appearance of new memory technologies and system designs. Fundamentally, not a single solution has managed to fully solve this memory capacity challenge. As argued in this survey paper, limitations by physical laws make the effort of expanding local off-chip memory impossible without adopting new approaches. The concept of Non Unified Memory Access (NUMA) architecture provides more system memory by using pools of processors, each with their own memories, to workaround the physical constraints on a single processor, but the additional system complexities and costs led to various scalability issues that deter any further system expansion using this method.

Computer clusters were the first configurations to eventually provide a Distributed Shared Memory (DSM) system at a linear cost while also being more scalable than the traditional cache coherent NUMA systems, however this was achieved by using additional software mechanisms that introduce significant latency when accessing the increased memory capacity. As we describe, since the initial software DSM systems, a lot of effort has been invested to create simpler and higher performance solutions including: software libraries, language extensions, high performance interconnects and abstractions via system hypervisors, where each approach allows a more efficient way of memory resource allocation and usage between different nodes in a machine cluster.

Despite such efforts, the fundamental problems of maintaining cache coherence across a scaled system with thousands of nodes is not something that any of the current approaches are capable of efficiently providing, and therefore the requirement of delivering a scalable memory capacity still poses a real challenge for system architects. New design concepts and technologies, such as 3D stacked RAM and the Unimem architecture, are promising and can offer a substantial increase in performance and memory capacity, but together there is no generally accepted solution to provide efficient Distributed Shared Memory.

## CCS CONCEPTS

• **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Interconnection architectures**;

## KEYWORDS

DSM systems, NUMA evolution, Memory expansion, System Virtualization, Unimem, Chiplets

**ACM Reference Format:**
Kyriakos Paraskevas, Andrew Attwood, Mikel Luján, and John Goodacre. 2019. Scaling The Capacity of Memory Systems;
Evolution and Key Approaches. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '19), September 30-October 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3357526.3357555

## 1 INTRODUCTION

Ensuring fast memory access to the required capacity is crucial for defining the performance profile of a computing system. As already predicted by the memory wall concept in 1994, [75] there is a continued disparity between CPU and main memory speed increase in favour of the processor resulting in performance losses due to processor data starvation. Caches fill the gap between the slower main memory and the large data capacity demands of the applications [37], but due to the relative limited size of caches, memory accesses are still very frequent with applications often still requiring to access a byte of memory for every operation computed [54]. Moreover, it is also essential to keep main memory as physically close to the processor as possible in order to minimize latency and maintain signal integrity.

In addition, applications attempt to ensure data is stored entirely in main memory in order to avoid accessing the slower storage memory that would impact application performance significantly. Providing adequate and fast memory to store the majority if not all the information of data-intensive application is a difficult task for

*MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA*

any system. Besides, supplying the application with more memory poses a continuous challenge, due to the trend of the ever growing applications needs for memory. The problem of providing more memory by expanding the memory capacity has lots of aspects, and this paper describes how various attempts and approaches have been taken to relieve this problem, how technologies have evolved over time and why none of current approaches appear capable of solving it. To the best of our knowledge, there is no similar up-to-date survey in literature that covers the broad aspects of this issue to this extent, and provide an insight into the future vision and trends.

The issues in creating a processing device capable of supporting a large memory capacity range from the physics of resistance and capacitance through to hardware design rules, including several factors such as the maximum die size, number of routable pins, timing issues and constraints as well as the mechanical placement of the memory interfaces on the silicon die, all making it more difficult to further expand main memory capacity supported by a single processing device. Due to current semiconductor manufacturing process, there are limits in chip dimensions that in return, limit the number of supportable memory interfaces as well as the connectivity to the off-chip memory devices. Maintaining timing to achieve the high frequencies of modern memory devices is also a limiting factor when attempting to place multiple memory DIMMs next to the processing device. Current technology has also pushed the limits in increasing the memory density inside a DRAM chip due to physical restrictions [41]. However, newly introduced memory technologies such as non-volatile memories are promising as a capacity replacement but still fall short in terms of performance compared to conventional DRAM [74], [10]. On the other hand, new interfaces such as High Bandwidth Memory (HBM) [36] [58] and Hybrid Memory Cube (HMC) [60] [66] offer potential solutions to some of these problems described, but are still not widely available nor provide the overall capacity offered by a DSM system.

NUMA Architectures have attempted to address the need of increased memory capacity by allowing access to memory interfaces across multiple sockets of processing devices however with different performance characteristics. Although these multi-socket NUMA systems offer additional memory capacity than a single socket system, this is at the cost of non-uniform and additional latency overheads requiring power hungry protocols to maintain cache coherence and consistency of memory access. Such systems also require kernel [43] and application awareness to ensure scalability further adding to the cost of hardware and software complexity. However, such systems continue to struggle to scale up beyond a limited number of sockets [50] due to maintaining coherence and associated inter-processor communication.

The struggles in scaling NUMA and emergence of computer clustering led to the creation of other approaches and architectures, such as DSM across clusters, that albeit cheaper, offer a unified address space in which all participating nodes can share a global view of the available memory. Such a DSM system is Unimem, that encapsulates the mechanism of inter-node communication while providing a paradigm for accessing the global data. Another contribution is the description of the Unimem architecture and the subsequent ongoing hardware implementation inside the context of the EuroExa project, that will eventually contribute towards the mitigation of the memory capacity issue described in this paper.

## 2 CONSTRAINTS ON SINGLE SOCKET MEMORY CAPACITY

Increasing the total memory capacity supported by a single socket processor is a profound solution to the memory capacity problem, but in reality, it is limited by various factors. These include the placement of memory devices, the dimensions of the processor chip and the methods available to connect these two. Even with technologies that start to address aspects of these issues, the fundamental challenge related to the size of the capacitor cell required to hold the memory state is limiting any further increase in memory density and hence, capacity.

### 2.1 Die size limitations

Semiconductor design requires the high performance I/O such as memory interfaces to be placed near the edges of a processor die, however there are only four available edges on any conventional 2D die. It would be very convenient for the chip designer to have available any size and therefore the desired area in order to fit the required number of interfaces and logical components around the edges of the die. However, as discussed below, in addition to the obvious cost implications, there are various reasons on why this cannot happen. Chip manufacturers use a procedure called microlithography to fabricate a silicon die. A silicon wafer is a thin-sliced semiconductor material, which acts as a base out of which dies are produced. A chip manufacturer desires as many rectangular dies on a round wafer as possible, in order to maximize the yield, which is associated to the usable and flawless surface area of the wafer. A stepper machine first makes a number of passes around the wafer, successively projecting the image of each die layer through a photomask ,or reticle, and then, after several chemical procedures, each die layer is fabricated. Larger dimension designs require larger masks and size, and therefore it is more likely to have errors in the resulting die due to fabrication imperfections or impurities in the wafer, and as such resulting in a smaller yield. In addition, for a given manufacturing technology node, the reticle has a maximum size even if the designer can afford the limited yielding of a large die. Thus, because of these restrictions on size, there is a limit on the amount of edge space available for I/O logic which in turn limits the number of interfaces that are used to access memory. Current state-of-the-art processors are able to support around eight from a single large die. [53] [15].

### 2.2 Packaging design limitations

Even when a large die implements many memory interfaces, further issues arise due to the complexity of routing and driving these interfaces on the Printed Circuit Board (PCB) on which and memory devices reside. A key factor that affects this ability is the number of pins that the packaged processor device can support and be integrated using current PCB technologies. Despite the increasing performance and I/O capabilities of a die, this also increases the power requirement. This creates a tension between the number of pins required to deliver power and the number of pins that can be used for memory. Reducing the size and pitch of pins can provide

more pins, however each pin is less able to deliver the required power, further increasing this tension. There is also the issue of voltage drop in wires, also known as IR drop [65], which can reduce the actual core voltage and hence signal integrity between the chip and memory devices. This, in turn, limits the physical distance that can be supported between the processor's memory interface and the memory device itself, which practically provides a depth limit of eight rows of memory devices from the processor.

## 2.3 Interface and chip placement and routing

As mentioned earlier, the length of the signal path from the processor to the memory chip will affect performance and timing. The shorter the path, the less the latency, and hence the longer the path, the higher the frequency, but the more likelyhood of skew and longer latency, factors that together limit the amount of memory that can be placed on an interface. When adding more memory channels, careful signal routing is required to maintain a uniform distance while also avoiding noise and congestion between signals. Another scaling problem comes from the parallel bus of a DDRx systems. As the DRAM clock rates increased, the signal integrity can become further degraded due to any noise. The fact that the electrical contact to the DIMMs is maintained by physical pressure from the DIMM slot contacts and is not soldered further contributes to the issue. However in some situations in which more DIMMs are added per channel, this problem further increases and often the clock rate must be lowered in order to maintain signal integrity.

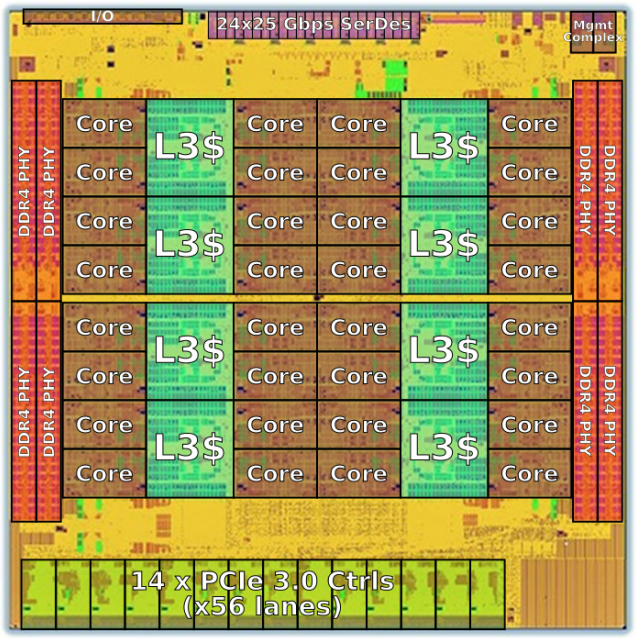


**Figure 1: Die layout of the Vulcan chip used within the Cavium ThunderX2 processor device demonstrating the limitations regards to the relative size and placement of I/O and memory interfaces on each edge of a large multicore processor die [4]**

This trade off of capacity for performance leads to further escalation of the capacity problem. Given all the above, the addition of more and more DRAM chips or interfaces to increase the memory capacity is not an option.

## 2.4 Limitations in increasing the capacity of a DRAM chip

In the memory chip level, extensive research and work has been conducted throughout the years in order to increase the memory cell density of a DRAM chip, and consequently, its capacity. Currently, we have almost reached a point where physical constraints do not allow any further increase of DIMM density on a single die, a trend clearly visible in Figure 3. In a DRAM chip which can include multiple dies, each bit of memory data is stored within a small capacitor and the presence or absence of the electric charge of the capacitor defines the memory state.

For successful DRAM cell operation, the capacitor in the DRAM cell should meet two requirements, sufficient capacitance (~10 fF/cell) and ultralow leakage current (J g < 10-7 A/cm$^2$ given an operating voltage) in order to limit the frequency by which a cell must be refreshed. The cell capacitance is expressed by

$$C=\epsilon_0 k \frac{A}{t_{phys}}$$

where C, $\epsilon 0$, k, A, and $t_{phys}$ are the capacitance, vacuum permittivity, dielectric constant, effective capacitor area, and the physical thickness of the dielectric layer, respectively. Scaling of the DRAM cell has continuously reduced the area allocated to the capacitor in the cell, such that a 3D structured capacitor is used to obtain the necessary capacitance in the limited area [42]. The aspect ratio of the capacitor has sharply increased and will reach ~100 shortly because of the aggressive scaling of DRAM. However, further increase in the aspect ratio is impossible because of structural vulnerability

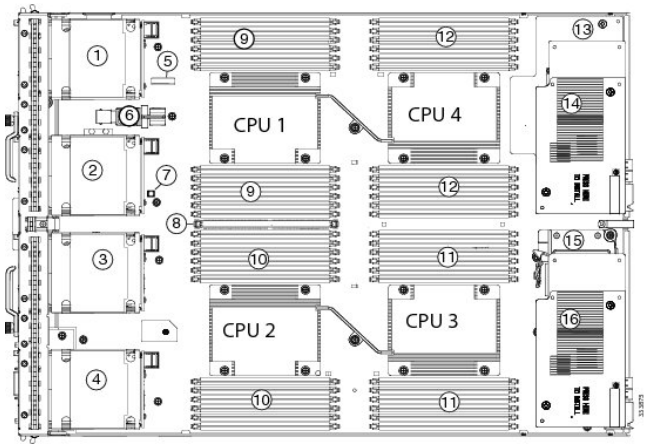


**Figure 2: The board layout of a Cisco blade server demonstrating the placing of DRAM memory devices (9-12) matching the edges that contain the memory interfaces inside the processors, further constrained by the physical dimensions of the blade server [19]**

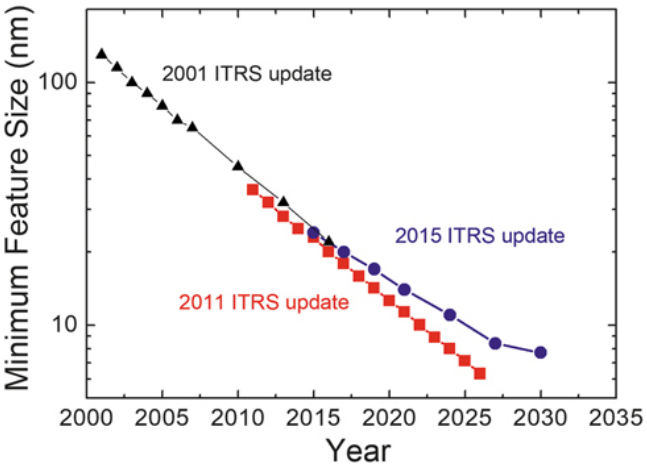


**Figure 3: Graph showing the reality in reducing the feature size (nm) (source [42])**

of the capacitor, therefore, a higher-k material has to be used as the dielectric. Industrial solutions provide new higher-k materials to address this problem, but as the size of the lithography process decreases, the task of developing higher-k materials becomes more and more challenging.

In addition, due to leakage of the electrical charge, the capacitor must be periodically refreshed, otherwise any data will be lost. By shrinking the size of the capacitor, the memory density of the DRAM can be increased. At the time being, DRAM manufacturers claim to have hit the limit in how many electrons are required to hold a charge within an acceptable refresh period.[42]. The complication being that DRAM data cannot be accessed during a refresh, resulting in performance loss.

### 2.5 Bypassing the limitations

In conclusion, the effort of increasing memory capacity associated with a single socket device in order to reach the capacity required by software has fought against physical limits that now more than ever are forming a red line against various laws of physics. For these reasons, interconnects and protocols have been created to allow access to additional memory owned by another socket within the same board or device within a network. This remote memory, although not as fast as memory directly attached to the processor, has promised to be faster than a traditional local storage device and therefore can act as an suitable extension to meet the application memory capacity requirements.

The following sections discuss the various technologies and approaches that attempt to provide to the application access to large capacities of memory.

## 3 NON UNIFORM MEMORY ACCESS

### 3.1 Emergence of NUMA

NUMA architecture implementations try to overcome memory capacity limitations by providing non uniform access to memory, increasing the capacity by connecting multiple processors and their associated memory interfaces to appear as a single processor. Each processor has directly attached memory, however any processor can access any memory attached to any processor in the system. NUMA does not change the memory to processor ratio in a system but instead provides a linear increase in memory capacity as the number of sockets increases. To improve NUMA memory access performance [23], several improvements in NUMA designs have been introduced, as well as increases in single node memory capacity. Due to the existent physical limitations that halt any further substantial increase on single socket memory capacity, NUMA can be a substitute in achieving memory expansion.

Historically, One of the first successful commercial NUMA machines was the Honeywell Bull XP S-100, a medium scaled NUMA multi-node system that was released in the late 1980s and supported up to 64 users and the by then incredible amount of 16MB of RAM across multiple memory banks.

### 3.2 Multi-node systems

A multi-socket NUMA system consists of processor chips in multiple sockets, each with globally accessible local memory (as shown in Figure 4). Accesses to remote memory of another socket was made possible through processor-to-processor interconnects and associated protocols, but their overhead induced higher access time than that of local memory. The transition from single node multi-socket to multi node multi-socket NUMA systems where each multi-socket node has additional links, required additional protocols in order to continue to provide a unified memory view.

Performance wise, these additional layers increased access latency due to the node-to-node communication overhead. This led to a shift of the memory challenges from the independent memory management into providing efficient software-hardware protocols that unify local and remote memory and allow any socket to access this unified memory address space. While minimizing latency is important, achieving higher performance on a Distributed Shared Memory NUMA system versus a typical Symmetrical Multi Processor (SMP) system with shared memory relies on using efficient interconnects and protocols, an efficient network topology ( ex. fat trees - hypercubes), as well as having a good fraction of data references satisfied by local memory. Also, locality is an important performance factor that can be greatly aided by the operating system when it can allocate memory for processes on the same node as the processors they are running on. From the developer's aspect, the distinction of the performance gap between local and remote memory accesses should also be known and therefore handled appropriately.

### 3.3 System scalability effort and the memory coherence problem

In addition to the overhead induced by the introduced interconnects and protocols to access remote memory, more issues arose as soon as the number of nodes increases in a NUMA system. In order to provide the same programming model, the fundamental aspect of cache coherence is guaranteed through the serialization of memory writes [5], but as the processor count increases, so does the scalability limitations due to this serialization effort, which after a point, it hits a limit [6]. Multi-socket cache coherent NUMA

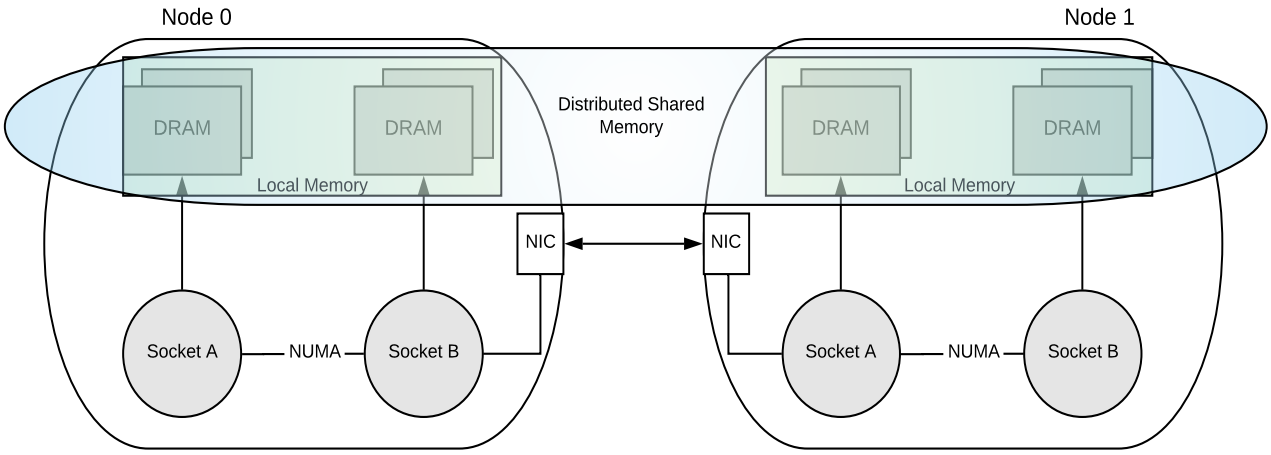


**Figure 4: An example of two dual socket NUMA machine each having local memories interconnected to provide a global Distributed Shared Memory system**

(cc-NUMA) systems after this point eventually struggled to provide increased performance while presenting a cache coherent memory view by means of write serialization, thus limiting any further scalability. On a typical cc-NUMA, (although typically still abbreviated to just NUMA) system communication between cache controllers provide a consistent memory image when more than one cache stores the same memory location. Fundamentally, as the system scales, it clearly becomes a communication-latency problem. As more processing nodes are inserted, the latency associated with the interconnect increases with the number of cores, which also further increase the per-core memory requirement for large systems and hence total capacity requirements. Also, it is becoming even harder to provide a system unified memory due to software overheads and network constraints, for which expensive custom hardware and software solutions are required.

Research studies [49] dictate that by tracking the shared data owners, the overall traffic overhead with each miss is not significantly larger with a larger core count. On the other hand the article does not provide an efficient solution on a larger scale beyond 4 thousand cores. After this core count, invalidations are becoming more expensive, and efficient tracking of the page or cache owners require even more hardware resources.

Industrial solutions such as NUMAscale with the NUMAConnect architecture [67] allow direct access to memory locations from any processor within the system, resulting in a homogenous cc-NUMA system, which is crucial to a program that exploits parallel processing, such as a High Performance Computing (HPC) applications. Instead, in a commodity cluster, communication typically takes place with explicit message passing between the processes on different nodes.

Historically, SGI (silicon graphics incorporated) [2] was one of the first companies to push NUMA the furthest of anybody with its Scalable Shared Memory Multiprocessor (SSMP) [46] architecture and largely scaled NUMA systems deployments using multi-sockets [69]. SSMP introduced and described by SGI as "the industry's first 64-bit cc-NUMA architecture and supported up to 1024 processors." SGI effectively created a DSM system while maintaining the simplicity of the SMP programming model and latency. However it was necessary for them to create a hierarchical partition scheme (hypercubes), to allow their multi-socket NUMA system to scale even further. This effectively alleviated the problem of write serialization however conceptually turned the system into a distributed clustered system. An example of such a system was the NASA Columbia supercomputer back in 2000s, that includes over 40 SGI Altix 4700 racks for a total of 4608 cores and 9 Terabytes of memory.[13] However, it became clear that clustering was a more efficient solution that could scale higher than a cc-NUMA system.

## 4 INTRODUCTION TO DISTRIBUTED SHARED MEMORY SYSTEMS

By definition, DSM systems consists of a collection of nodes with one or more processors and private memory that are connected by a high speed interconnection network. The structure of an individual node is similar to a stand alone computer, except the nodes are usually smaller in terms of peripherals. At the highest level, a DSM creates a global address space that can be accessed by all nodes. A range of the global address space can be inserted into the existing Local address space of each node, or each node can receive a fraction of a separate global address space that is then subsequently partitioned and shared across all system nodes. For the system to be scalable, various prerequisites which will be discussed later are required. In addition there is a distinction between how the global address space is presented to an application and how the system implements the node-to-node communication.

It is also useful to understand the distinction between system scale-up and scale-out, as these two forms of scaling act as the basis between the memory capacity that can be obtained within a single node (scale-up) and the number of nodes that are required to provide

the desired system capacity-capability (scale-out). Over time, many solutions have contributed in both these aspects, targeting either in increased performance through various innovations such as interconnects or to reduce system complexity such as software based DSM implementations.

The term scale-up, or vertical scaling is referring to the ability of an application running on a single node to use all the increasing resources attached to that node. By scaling up a system, memory capacity or processor count can be increased but the system node count does not. In cc-NUMA, coherence is provided to maintain the application programming model, but as the system scales, it reaches a point where coherence overheads become so large that it negates any performance gains by adding more resources, such as processors, meaning that further scale-up is not possible.

Scale-out, or horizontal scaling is effectively the addition of system nodes typically with similar capabilities, by attaching all nodes to a common network. The aim is to create a uniform and capable system where applications that adopt a distributed computing model can take advantage of the additional number of nodes. However, application developers still long fought for the existence of the shared memory software model which offer an efficient way of passing data between programs, even on these network clustered machines and marked the beginning of existence of DSM.

Therefore, the evolution of DSM needed to concentrate on two aspects; how applications access the global address space typically through libraries and language extensions and how the node interconnect creates the global address space.

## 4.1 Evolution of Interconnects

Initial DSM systems used the existing interconnect protocol stacks between nodes, such as TCP/IP, to provide communication between the nodes of a DSM. By definition these stacks add a significant overhead to the remote memory access and became a prime candidate for initial innovation. An obvious improvement was to replace the TCP with a lightweight protocol such as UDP or support the protocol with hardware offload such as Remote Direct Memory Access (RDMA) where blocks of memory can be moved directly between nodes without processor involvement. This trajectory has continued with newer innovative interconnect architectures that offer significant performance gains by further removing the network overhead of existing protocols and processor involvement.

RapidIO [24] was a hardware communication stack which provided coherence between nodes in clusters, however it was not well adopted due to its proprietary nature. Newer non-proprietary interconnection protocols provide the semantics for easier adaptation. For example, Gen-Z [22] is a scalable, universal system interconnect that uses a memory-semantic (Load/Store) protocol and enables multiple components of different types to efficiently communicate by instantiating a bridging device which provides direct byte addressable communication with the remote node compared to the RDMAs block structure while, at the same time, natively providing addressability into the global address space. Practically, this allows to any component (processors, accelerators, network cards) that can access this bridging device to talk to any other component as if it were communicating with its own local memory using dedicated memory access commands, and thus it is called as "memory-semantic protocol" by Hewlett Packard Enterprise. Memory-semantic communication moves data between memories located on different components with minimal overhead and latency by using load-store memory semantics everywhere. The drawback is that it does not support cache coherence, thus limiting performance in shared memory systems in which software (and subsequently additional overhead) is required to maintain coherence around every transaction across the application. Although promising, lack of cache coherence support will make scalable applications development difficult beyond the benefit of providing byte access to large storage devices.

Another interconnect, the Cache Coherent Interconnect for Accelerators (CCIX) [21] is a scale-up interconnect architecture developed by the CCIX consortium and offers memory consistent communication between heterogenous elements such as system accelerators and processors. This is accomplished by providing a cache-coherent protocol over the standard PCIe physical interface which may allow it to be more successful than RapidIO. Although high-performance, this heterogenous NUMA architecture suffers from the very same memory coherence scalability issues that write serialization induces, introducing similar bottlenecks as in a homogeneous scaled system, and therefore this cannot be considered a solution to the memory scalability issue.

## 4.2 Advancement of language extensions and libraries

From software perspective, in order to abstract the interconnect protocols and provide a DSM programming model, several language extensions as well as software libraries have been implemented. Within the HPC market various Partitioned Global Address Space (PGAS) programming models have been developed that assume a parallel programming model with a logically partitioned global memory address space across nodes. PGAS languages try to combine the advantages of a distributed memory system programming model such as MPI [72], that can benefit from hardware accelerated interconnect, with explicit data referencing semantics of a shared memory system. Examples of such implementations include, BXI by Atos [26] and ConnectX by Mellanox [73], that combine proprietary interconnects and hardware primitives that can be mapped directly to communication libraries.

PGAS can be provided to the programmer either as a language, language extension or through libraries, such as such as Chapel [16], Unified Parallel C (UPC) [29], and Co-array Fortran [57]. They often allow a weak memory consistency to counter the restrictions of a strict sequential consistency which as the system scales, becomes prohibitively expensive performance wise. Unfortunately, these approaches are provided by multiple vendors, each often using proprietary hardware solutions, and therefore have a limited adoption. Efforts have been made to tackle this proprietary nature constrain, with attempts to standardize RDMA access to lower latency, or adding additional layers of abstraction such as Portals 4 [12] in order to standardize underlying hardware and provide low latency from either the language semantics or software libraries.

Several software DSM implementations rely on kernel modifications and libraries to provide a portable solution of a DSM system

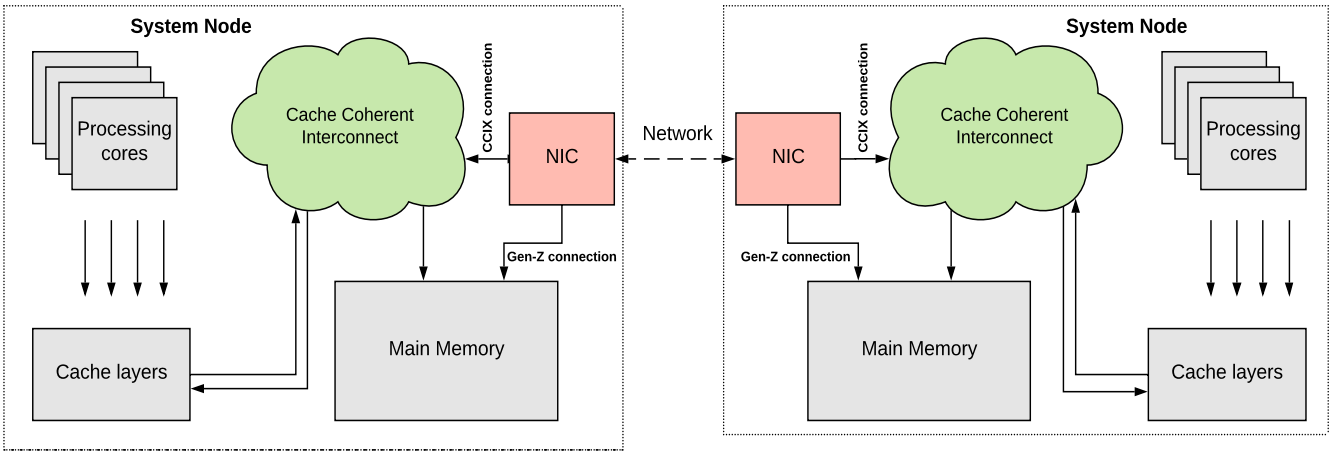


**Figure 5: Demonstrating the distinction between CCIX and Gen-Z path to main memory**

over network by merging mainly commodity solutions into a unified system. Treadmarks [40] supported DSM over UDP/IP, where Grappa supported both TCP and RDMA configurations [56] and provided a user-level library and runtime to the user, although these solutions came with a significant software overhead and memory latency. Treadmarks tried to reduce the amount of necessary communication to maintain memory consistency but was successful only to a point. FaRM [27] utilized RDMA for better performance by removing TCP/IP network stack, but throughput is still limited to the RDMAs lowerr effective bandwidth compared to the cross sectional bandwidth of the network between multiple nodes and the inefficiencies of setting up an RDMA transfer for small amounts of data. In the latter case, a more traditional approach of data copying should be adopted.

ArgoDSM [39] is another new, highly-scalable, software DSM system for HPC and Big Data applications, meant to be run on top of a RDMA-capable network layer that can provide a cache coherent global address space in a distributed system without dedicated hardware support but with efficient synchronization, page locking and a new software based coherence protocol. Benchmarks show that Argo scales more than UPC after a certain number of nodes, and generally relieves the problem of centralized coherence approach, but without dedicated hardware support, such as fast interconnects, access latency remains high.

Lastly, efforts exist to create a specification for a standardized API for parallel programming in PGAS through OpenSHMEM [17]. Along with the specification, OpenSHMEM provides a reference of a portable API implementation that allows it to be deployed in various environments and hardware interconnect accelerators.

## 4.3 Virtualization of the memory system

Despite the efforts to standardize languages and library access to DSM, the most portable standard in terms of application accessing memory is the processor architecture itself, the Instruction Set Architecture (ISA).

In this aspect, efforts there have been made to utilize the hardware virtualization of the processor to virtualize the Global memory address space and implement the DSM within the hypervisor and hence provide the global memory as an integrated part of the virtualized processor local address space. This removes the need for any library and language extension to be used by the application to access global memory, thus reducing the overall overhead. Over the years, the pure software and language implementations of DSM systems focused on reducing memory update cost for increased system scalability whereas virtualization [61] was utilized to share the available resources for increased flexibility. In the first examples where the DSM moved into the hypervisor it simply presented remote memory pages in the virtualized local address space [18], however more recent examples have been able to provide a more relaxed memory consistency models in order to reduce the network load while also using advanced AI/ML algorithms that provide real-time optimization mechanisms, such as page and thread migration based on workload and memory patterns as an attempt to improve locality and offer optimal performance in common situations.

Examples of hypervisors that provide DSM to ther guests include industrial products, such as vSMP [68] by ScaleMP that combines clusters of commodity x86 servers in order to create a distributed virtual symmetric multiprocessing system. By doing so, the virtual SMP system can have better scaling capabilities to proprietary SMP systems, while maintaining the cluster's low cost. Another product example is Hyperkernel, the core of the TidalScale [55] tool, and as the name suggests, is a hypervisor that creates a distributed kernel to manipulate system nodes, but is transparent to the applications that run on it. TidalScale currently supporting only intel Xeon processors. AMD Epyc processors, ARM server processors, or IBM Power processors are yet to be supported. Although the application has been simplified due to access to a processor native memory system, the total complexity and depth of the stack used to implement the DSM has increased. Even with the addition of further intelligent innovations offered through the flexibility that

the hypervisor provides to limit the costs of remote access, there are still many usage scenarios, such as when a large number of threads access the same page, or when a single task is accessing lots of remote pages that lead the application suffering from the full latency cost of the stack. Nevertheless, system virtualization, with further novel additions, can be a potent contributor towards the relief of the memory capacity problem.

## 5 EMERGING TECHNOLOGIES

On the hardware level, innovations and new architectural approaches of the memory subsystem led to significant breakthroughs, resulting in new technologies that offer higher memory capacity. Several memory dies are now stacked inside a single die, Non Volatile Memories (NVM) are pushing the density limits far beyond DRAM could ever reach, memory interfaces with serialized bus allow more memory devices to be attached to a single processor, and novel approaches, such as Unimem that aims to lift the barriers of a DSM system at scale, all these innovations would eventually allow higher capacities of memory available to a system.

### 5.1 Multi-chip modules

One of the ways to overcome the size induced yields issues on a single die is by partitioning the large die into smaller dies which also in consequence increases the total edge area. This partitioning simplifies power supply requirements because of shorter interconnect lengths, and offers the opportunity for greater miniaturization and reliability at system level due to decreased number of interconnects between system level components thus leading to lower cost and simplification of by putting several devices, including memory devices, into a single package.

Historically, research on the Multi-chip modules (MCMs) dates back to the 80s. [14]. Currently, the evolution of chip manufacturing process allowed the creation of devices with more dies integrated into a single package both in 2D and 3D plane, providing higher yielding devices capable of supporting a larger number of memory interfaces, as demonstrated in Figure 6.

### 5.2 Non Volatile Memory technologies

As discussed earlier, single DRAM chips have almost reach their density limits. However, since NVMs do not use capacitors, they are able to scale further and provide additional capacity compared to DRAM. Current generation of volatile memories cannot replace DRAM due to performance and durability concerns as already stated in [10], but since they offer higher capacity and lower power, NV Memory is a very interesting trajectory for increasing the capacity of memory that is close to the processor. This emerging technology could solve may of the big issues introduced earlier, such as the capacity per $mm^2$ while also capable of solving the pin out problem, due to the capabilities of the processors to control flash through the high speed serial I/O already present.

Research has shown that NV Main Memory (NVMM) improves application performance significantly over flash-based SSDs and HDDs, without application changes [76]. Table 1 summarizes the characteristics of NVM technologies, compared to DRAM [51]. Spin Transfer Torque RAM [8] has worse write performance compared to DRAM, as well as lower density compared to other NVMs. Resistive-RAM [71] has the potential to be integrated into a CPU die, since it can be fabricated in a standard CMOS logic process, and enable massive memory parallelism, high performance and power efficiency, as shown in [35]. The same study also makes clear that several challenges that span across multiple system abstraction layers need to be overcome for this concept to be realized, More specifically, in addition to the immature ability of the ReRAM to be used as main memory instead of flash memory (in terms of latency and reliability), a large chip area is to be occupied by ReRAM logic, directly pointing to the need for designing much larger cores, a task that is already difficult as we already described in Section 2.1.

Phase Change Memory (PCM) [63] offers higher memory density but also higher write latency and low endurance, however, as this is a significant research area, newer technologies such as Domain Wall Memory [70], which is the less mature of the non volatile Memory technologies due to its physical nature, can offer higher density but non uniform access latency. Other promising technologies include CeRam, nanotube or Skyrmion [52] based devices.

Currently, processors do not uniformly have native provision for the persistence in main memory which is a trait of NVM. Assuming that future generations of processors will overcome this by provide support for NVMM persistence, and the technologies will mature, there is a true potential to increase the total memory capacity of even a single node.

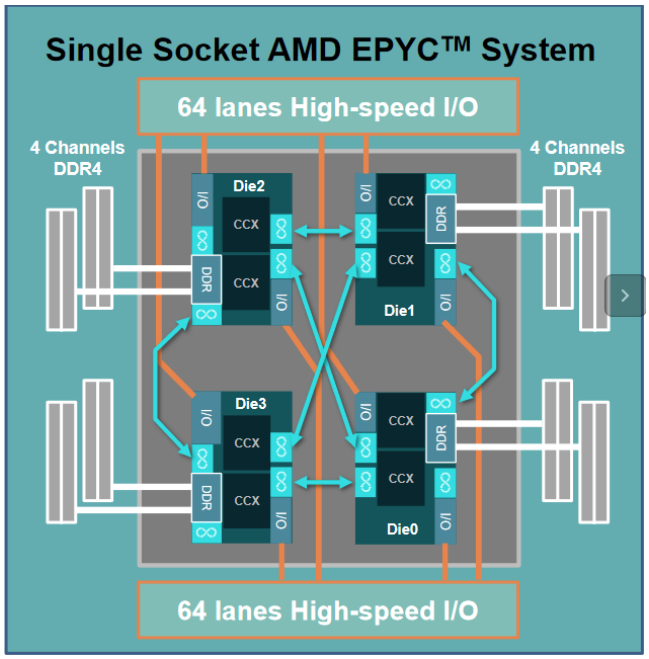


**Figure 6: The layout of an EPYC package containing 4 chips, supporting a total of eight DDR4 channels**

| | DRAM | PCM | Other NVMs |
|---|---|---|---|
| Density ($F^2$) | 6-12 | 4-16 | 4-60 |
| Read Latency | 10-50 ns | 48-70 ns | 10-100 ns |
| Write Bandwidth | 1 GB/s per die | 100 MB/s per die | 140 MB/s - 1 GB/s per die |
| Endurance (cycles) | $>10^{16}$ | $10^9$ | $10^{12}$ - $10^{15}$ |
| Byte Addressable | Yes | Yes | Yes |
| Volatile | Yes | No | No |

**Table 1: Comparison of Memory technologies properties [76]. We can see the larger density supported by NVMs over DRAM. Other NVMs include the Memristor, STTM, FeRAM, and MRAM..**

### 5.3 Memory Interfaces

As discussed in section 2, the delivery of memory comes through a parallel bus, which limits the off-chip capacity that can be supported. To address this problem, in-package substrates which can support many smaller pins between processors and memory can be used to alleviate the pin count problem. HBM leverages the benefits of smaller pins, and in turn by being integrated within a MCM, together they significantly reduce the issues associated with interfacing a processor with memory, however the amount of memory that can be placed inside the MCM is still small compared to the amount of memory that can be placed off-chip.

An alternative to the parallel memory bus is to use a high speed serial bus. This approach addresses many issues, including the number of pins needed to be supported by the processor device. The narrower bus is also easier to route and can travel further across the PCB without suffering from timing skew. Together, this effectively allows more memory devices to be attached to a single processor, however it requires the memory device to include active control of the DRAM. The HMC includes a logic layer that redirects requests between off-chip serial interfaces and die components, while also supporting in-memory operations. HMC is a good example of this approach which in addition when multiple high speed interfaces are provided to the HMC device, a chain network of memory devices and processors can be created [66], thus effectively creating a DSM. In both the HMC and HBM technology approaches, the capacity to the associated memory devices is limited by the form factor, however the stacking of memory dies is allowing these to approach the capacity of traditional DRAM DIMMs. Performance-wise, simulations have shown that HMC and HBM can reduce end-to-end application execution time by 2x-3x over DDRx and LPDDR4 architectures [47].

### 5.4 Stacked memory technologies

The introduction of 3D-stacked DRAM inside the die however leads to the increase of the single memory device capacity by accommodating more dies inside a package, and therefore can offer more package memory density.

The DDR4 [25] standard introduced the ability for Through-Silicon Vias (TSV) stacking extensions that allowed manufacturers to create high capacity DIMMs. In addition, Registered DIMMs (RDIMMs) allow more stability in situations where more than one DIMM per channel is used. Samsung announced 256-Gbytes 3D stacked RDIMM based on the 10 nm fabricating process, 16-gigabit DDR4 DRAM that offer lower power consumption and improve performance by doubling its current maximum capacity [32].

The stacked DRAM dies are connected with TSVs that shorten the interconnection paths, reducing the channel latency and the energy consumption. An HMC implementation provides device addressable memory stacked DRAM dies on top of a silicon die while removing the obstruction of limited pin count by introducing differential serial links connected to the processor. A version of HMC named Multi-Channel DRAM (MCDRAM) was developed in partnership with Intel and Micron to be used in the Intel Xeon Phi processor codenamed Knights Landing. This introduces a new paradigm in which DDR4 and 3D-stacked RAM are used by the processor, each with different performance characteristics and modes, such as a cache-only, memory addressable, and hybrid mode, which is a mixture of both [9]. Lastly, it is worth mentioning what is argued in [62]; even when device chains of 3D-stacked DRAM may significantly increase the available memory bandwidth, only applications with high level of memory-level parallelism will benefit, while the rest will not.

### 5.5 Compute units, Chiplets and Unimem

At the DATE conference [20] in 2013, the concept of a Compute Unit was first discussed in which a computation engine, its memory and a coherence interconnect provides a locally coherent view of the unit's memory hierarchy to the outside world while also providing a path to remote memory (see Figure 8). This remote memory space by means of a global interconnect becomes a globally shareable address space, in addition to the local address space, in which remote

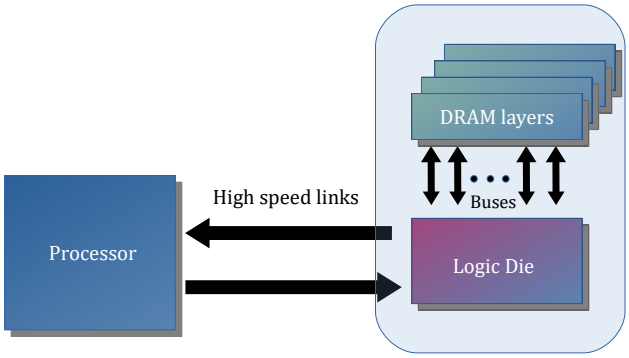


**Figure 7: An example of 3D stacked RAM.**

units can access locally owned memory coherently with any cached access from any computation engine within the local unit. Unlike traditional shared memory models, this model defines a single-sided coherence scheme in which only the owning unit can cache any globally shared region of local memory. This removes the complexities and costs associated with maintaining a coherence protocol between units, and hence lifting the scalability limiting serialization of coherent writes, while also removing any need for software to manage caches at the memory owning unit. Address translation facilities in the bridges between the local and remote address spaces also enable the remote address map to be defined by the configuration of the global interconnect as opposed to a globally agreed configuration between all the local unit's address spaces.

These aspects of a Compute Unit were defined to enable a DSM capability between multiple units with the global interconnect operating at the processor native address layer of the Compute Unit. This also removes various levels of the typical DSM communication stack in that an application or any hardware block executing in one unit is able to natively Load/Store a remote unit's memory. Protection and translation between each unit's configuration of its local address space is accomplished via the configuration of their local bridges to the global address space. Support for global atomic transactions by using monitors local to each of the local memories, thus providing the fundamental capabilities to support NUMA enabled operating systems. Clearly the requirement not to cache remotely accessed memory would significantly lower the performance of operations on remote memory, however, other than sharing within a single distributed virtual address space it was found that for inter-process communication, the local cache was typically cold as the processors move data between processes.

A further benefit of the Compute Unit and the flexibility in the arrangement for the global address space was also defined so that a silicon implementation of a Compute Unit could be reused between designs and be delivered as a silicon module called a Chiplet. The term Chiplet was first used to define a silicon module that has the properties of a Compute Unit, however more recently the term is also being used to mean any subsystem of a design hardened as a module in silicon. The link between the concept of a Compute Unit, its delivery as a Chiplet, and the scalability of a design was first investigated and prototyped in the EU funded FP7 project EuroServer [28].

Following the EuroServer project, a number of subsequent projects further refined various aspects of the Compute Unit based DSM architecture, in what has become known as the Unimem Memory System Architecture [64]. Currently, the H2020 project EuroExa [3]

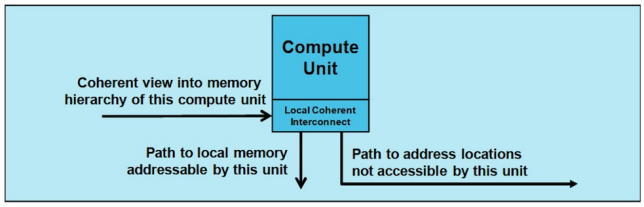


**Figure 8: The concept of compute unit (Source: ARM)**

is implementing a large scale prototype of a Unimem capable HPC platform that implements an Arm processing system as a Compute Unit Chiplet capable of processor-native Load/Store transactions through address translation bridges in which a global interconnect and routing topology provides an owner-coherent, DSM system.

## 6 OVERVIEW OF THE UNIMEM ARCHITECTURE

A further contribution towards the alleviation of the issue comes by providing a description of the Unimem memory scheme, and outlining the ongoing effort of a Unimem-enabled hardware implementation, and more importantly, describing an efficient hardware mechanism for the address translation procedure in order to support a global virtual address space.

### 6.1 An evolved compute architecture

From 1970 up to 90's the typical compute architecture was comprised of the CPU, the main memory and the storage memory, following the von-Neumann model. Later on, until 00's, newly introduced Multi-socket SMP systems provided a computing paradigm that offers more computing power at the expense of less main and storage memory per thread. In the decade that followed, the commercially available Multi-core Multi-socket NUMA systems delivered an increased computing power while at the same time mitigating the memory issue by providing more RAM and storage memory to threads. Nevertheless, the impotence of NUMA systems to scale beyond a certain point contributed to the appearance of non Von-Neumann architectures in which the computations are offloaded to accelerators, leading to shorter computation times, but at the same time, access to main or storage memory is bottlenecked, since all memory operations are propagated through the host itself.

In order to alleviate this issue, a control-data plane architecture is proposed in EuroEXA project, where each data plane owns its own main-storage memory. In the context of the project a prototype system that aims to take the HPC approach a step further is being implemented in which accelerators are given native access to both network and application memory, and in addition, storage memory is distributed with locality to each node in contrast with the traditional HPC model where all the network and accelerator data movement to the host memory is processed by the host itself, as shown clearly in Figure 9. The acceleration is centralised to the FPGA fabric of a single device and the storage is moved physically onto the network interface, which is also implemented on FPGA.

Unimem is an innovative scalable memory scheme firstly described in the EUROSERVER project as a new memory model across the server system, which breaks the traditional dual memory types available in today's systems: cached memory and device memory. Typically, in order to provide scalability to the cached memory type, a global coherent protocol is required, thus limiting system scalability. In return, this practically limits the ability to deliver fast, shared memory to the application, but at the same time is crucial for maintaining a consistent shared memory state across the system. It is known that applications in data-centres tend to partition their datasets across servers, presuming that these datasets will be placed near the processors and caches of an application task. In many cases, it is faster and more energy efficient to move tasks near the datasets

| ***Unimem compared to:*** | |
|---|---|
| *Cache coherent shared memory (eg. SMP, ccNUMA, SGI)* | Requires coherence protocol across all memories and nodes, thus limiting scalability. Unimem surpasses this limitation by delivering local memory coherence |
| *Software managed PGAS* | Software solutions required to create a model and API for communicating in address space. Instead, Unimem provides true hardware support for PGAS |
| *RDMA communication* | By using a dedicated DMA device engine to move data directly to remote memory, Unimem supports a common global address space with no correlation between source and destination addresses |
| *Communication devices* | No need for the application to allocate software managed buffers for data movement. Unimem natively supports direct puts-gets at hardware level |

**Table 2: Comparison of the Unimem approach with other communication schemes**

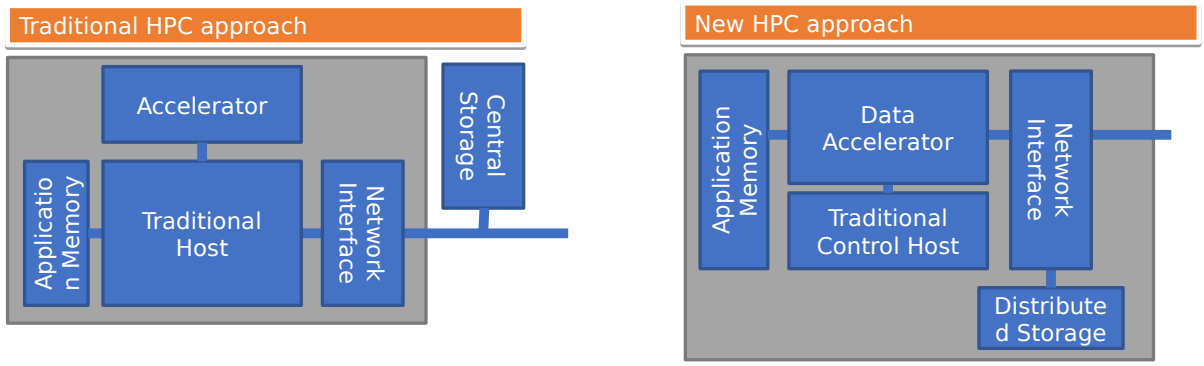


**Figure 9: Illustration of the node approach in a traditional HPC system vs the proposed approach by the EuroEXA project, where the Accelerators have direct access to application memory and network interface**

than moving the datasets near the processor, especially when the datasets are large, but this requires advanced AI algorithms and pattern analysis for page and thread migration, just as hypervisors mentioned above already provide.

Instead, Unimem maintains consistency across the system by providing cache coherence within each compute node and its local DRAM, and treating remote memory as shared and uncached device memory, thus providing flexibility to the communication paradigm with small additional overhead, by utilizing the available ARM technology. This approach offers a scalable system with minimal performance loss due to maintaining cache coherency. A notable proposed scalable system architecture is described in [34] in which, NAND flash is used as main memory technology and DRAM as a cache for flash, allowing a substantial increase in main memory capacity with negligible performance loss. Although this system can provide a global virtual address space and a considerable amount of main memory per node, cache coherence is relied on software solutions and eventually performance may suffer the latency cost of the stack.

In Unimem, data movement acceleration is achieved by replacing the commodity TCP/IP with RDMA for large data sizes or direct CPU memory operations for smaller sizes, thus delivering a large, fast amount of shared memory to the application. Since this Unimem sharing path is directly from within the hardware memory system of the Compute Units, the ExaNeSt-EuroEXA consortium has already demonstrated that moving the traditional TCP/IP based I/O communication stack directly on top of the Unimem RDMA stack can significantly accelerate data movement, and has also implemented additional features (such as transfer QoS) [31] [45] [59] [44] [48]. The majority of data-centre applications are I/O [11] and memory-bound [33], and should therefore benefit from remote DRAM borrowing via Unimem. Ensuring backwards compatibility with the existing application models and software investments is important, however, Unimem's raw capability also opens up significant opportunity for further optimizations in future runtimes and application frameworks.

## 6.2 Features of Unimem

Unimem exposes a 128-bit global address space consisted of multiple addressable Compute Units that can be addressed natively by low latency, hardware level Load/Store transactions and primitives, without any additional CPU intervention other than issuing the transactions. In order for this local memory to be globally accessible,

| *Component* | Width | *Usage* |
|---|---|---|
| *Global Virtual Address* | 64 | Determine the memory page and offset inside the GVAS |
| *Location coordinates* | 16 | Used for interconnect routing |
| *User bits* | 48 | Used for a variety of functions, ex. to define the operation type |

**Table 3: Brief breakdown of the 128-bit Unimem global address space format**

the global address of each compute unit maps directly back to the unit's local address. In such a configuration:

- Any Processing Element (ex. CPUs) inside a Compute Node can access the full local memory map
- Only a single data owner can cache globally shared memory, thus providing data locality. This ensures optimal performance and memory coherence across the system
- Nodes read/write data coherently with the data-owner
- Native hardware level one-sided communication is provided for Load/Store or atomic operations.

A breakdown of the Unimem global address space is shown in Table 3. Since current ARMv8 [30] architecture supports a 48-bit Virtual Address Space, 16 out of the 64 Global Virtual Address bits are provisioned to support future generations of CPUs. The location coordinates field are used for interconnect routing and allow geographic routing, since location awareness is provided to each node, thus simplifying network logic. Lastly, 8 of the user bits are used to specify the kind of operation, out of a total of 256 individual operations, and 16 bits are used to specify the Protection Domain ID and used for security purposes, while the rest are also reserved for future use.

Additionally, applications can use RDMA to generate both local and remote transactions, in an library based API that also provides global address memory allocation and sharing. Furthermore, EuroExa adds support of native generation of remote transactions from the processor, hence applications can also access remote address space without using any intermediate software stack that. Table 2 compares briefly the Unimem to other communicating schemes, highliting any similarities-differences.

Virtualized Mailboxes and Packetizers-Depacketizers are implemented in FPGA in order to send short messages between Compute Units as well as to support Atomic and Collective Unimem operations (such as barriers and reductions) across nodes, a concept that been succesfully exposed already by ExaNode/ExaNeSt/EuroExa projects [64] [38]. One additional feature of this system compared to others is the implementation on FPGA of a large part of the Unimem hardware involved in the translation procedure, which, in conjuction with the flexibility of the user bits described earlier, provides the flexibility to broaden the range of operations accelerated in hardware and therefore, be able to adapt to specific software semantics. This approach eventually allows the implementation of special DSM functions in order for the application to natively access the shared memory.

## 6.3 Implementation of Unimem

Each EuroEXA Compute Unit shares a common local address space that includes ARM Cortex A series processors, 16GB of LPDDR4 DRAM, 154Gbps FPGA link and a 400 Gbps Interchip Link. Accessing the local address space in each node is achieved using native AXI transactions generated from the local CPU or FPGA accelerators. In current CPUs there is lack of native support for accessing a large global address space, since the larger address width surpasses the capabilities of current CPUs in terms of availible address bits. For example, current ARM processors support up to 40 bits of physical addressing that could be used to map into memories of remote nodes, practically limiting scalability. In order to provide such a capability for remote memory transactions in a large global address space, additional hardware beyond the CPU is required.

For this reason, each node contains hardware bridges implemented partially in ASIC and FPGA logic that provide the required functionality for any kind of remote memory operations between nodes, and that are also easily extendable with low cost to support future operations. The AXI locally generated transactions towards the To-Bridge have a number of user bits that are used for operation encoding, thus enabling a particular remote operation to be implemented in hardware. The To-Bridge is then responsible for translating local memory addresses into Unimem Global Virtual Addresses (Unimem GVA). These transactions reach the packetizer, which is implemented in FPGA and converts the 128-bit Unimem AXI operations into Exanet [7] packets, according to the transaction type specified in the AXI user bits. In addition, the ExaNet packet 80-bit address format has a 80-bit address address space includes:

- The Protection Domain ID (PDID) that identifies the system-level process group to which the data belong. At each node, there is at most one process per PDID;
- the Destination Coordinates ID (DCID) that identifies the node at which the data reside, thus allowing a flexible network topology;
- the Destination Memory Address that identifies a virtual or physical address of the process belonging to the group PDID and running at node DCID.

The ExaNet packets are then propagated into the network through the ExaNet Network IP[7], implemented on FPGA by one of the project partners, which is part of the remote interconnect as shown in Figure 10.

On the receiver side, the From-Bridge of the remote node essentially depacketizes and translates the exanet packets into a set of operations, where operations such as atomics may trigger an interrupt to the CPU and the corresponding interrupt handler perform the specified operation. This setup offers flexibility in operations, allowing collective or atomic operations to be conducted through the appropriate CPU interrupt handler, or RDMA engines to be used as accelerators for faster large remote reads and writes, where in the common case of load and store instructions, the depacketizer in the From-Bridge will issue native AXI read or write operations directly to its local memory without invoking the local CPU, then send the data or the notification back into the request node through mailboxes.

Likewise, Figure 10 displays a high-level detail of implementation that shows the distinction of the local interconnect and remote

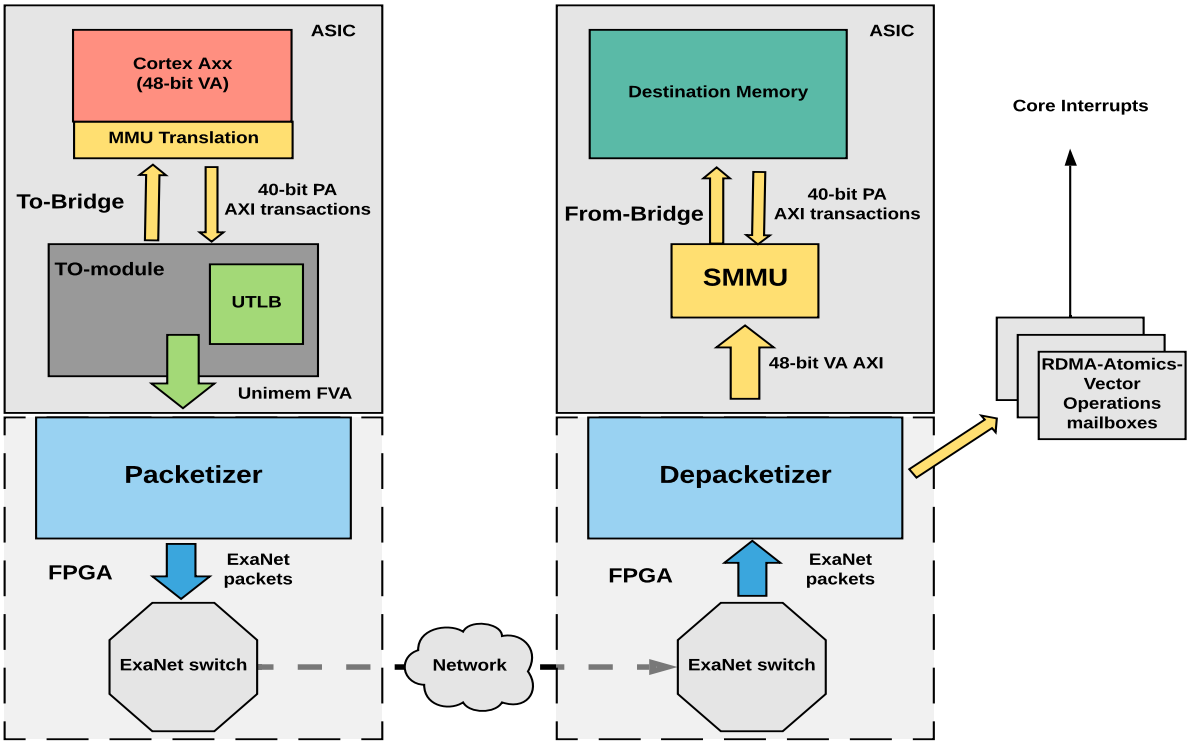


**Figure 10: Distinction of the various EuroEXA address formats (both for Virtual and Physical addresses), while a memory operation traverses through the EuroEXA resources. On the initiator node , the ARM Cortex A series generates a 48 bits Virtual Address (VA), which is fed to the Memory Management Unit (MMU). The Physical Address (PA) generated from the MMU is 40 bits. Then, the Unimem-To module is responsible for translating the Local Physical Addresses into Unimem Full Virtual Addresses (UFVA). Finally, the reverse translation process is handled in the destination node and the resources are accessed through the System Memory Management Unit (SMMU)**

interconnect in a compute unit, and the "To" and "From" Bridges, implemented both in ASIC and FPGA, which are essentially responsible for mapping the local addressed transaction into a remote addressed transaction, and also map the local addressed transaction into a remote addressed transaction respectively.

It is also worth describing more thoroughly how the address translation process is carried out. The Unimem-To module is a configurable custom hardware that provides a Unimem address translation mechanism. The ARM-based processing elements generate AARCH64 virtual addresses, which are initially translated by the Memory Management Unit of the CPU. Then, the Physical Addresses are routed into the Unimem-To module via the ARM AXI4 interconnection protocol [1]. The module contains two translation caches that act as TLBs, depending on the Memory Type, either Unimem, or Page Borrowing, where, in each case, the appropriate translation cache is accessed. The output of the Unimem-To module is a 128 bits Unimem Global Virtual Address.

These remotely addressed transactions are then directly packetized into ExaNet packets rather than having to create an ExaNet packet by writing multiple times to the packetizer AXI transactions which would lead to additional latency. This approach allows remote accesses to be implemented with native Load and Store instructions issued from the local CPU. We also argue that by utilizing the configurable SMMU and the protection features it provides, we can implement functions in hardware that do not invoke the OS (ex. through interrupts) to access protected pages or other resources. Instead, these functions can easily get access to these resources directly by simply meeting the SMMU security requirements already specified for these resources.

In terms of software, provisions to the OS include the support the larger 128-bit Unimem Address translation scheme, as well as to provide as much local memory quantity as possible where needed, in order to avoid the remote access penalty, and also provide self-location awareness. In addition, since the local virtual address translation into remote addresses is controlled by the remote To-Bridge, a kernel managed lookup table will be implemented to provide the Memory Management Unit (MMU) with the required additional attributes in order to extend the local virtual address into a Unimem global virtual address.

## 7 CONCLUSIONS - VISION OF THE FUTURE

Due to the ever lasting conflict between the available memory capacity and the application needs, and the base technologies that deliver memory capacity, the implementation constraints of DRAM and processors appear to be reaching the limits of current technologies. Consequently, other approaches have been introduced, such as the NUMA architecture which maintains a system in which memory is accessible by any processor, although with subsequent limits in scalability. Creating a distributed memory view across clusters of machines, even though at a lower cost, suffered greatly from interconnect technology inefficiencies and the abstractions required to transit from the software layer to the various interconnects. Emerging technologies and memory designs look promising in increasing the memory capacity, however, in conjunction with a scalable cluster technology at the hardware level, might be the catalytic agent for providing a system with fast, and ample memory capacity.

The described Unimem system enables the low cost future implementation of DSM functions in hardware, by exploiting the existing memory management capabilities of commercial processors while providing a global addressing scheme that, as already discussed, allows a scalable coherent view of memory while at the same time being customizable for supporting additional functionality in the future. This functionality is to be evaluated throughout the software stack involved, and the performance increase in various software DSM environments and applications will be measured, but for this to happen, additional kernel drivers for hardware management are to be developed as well as modifications to extend the API of software DSM environments to exploit the custom underlying hardware. Along with the increase in examples of silicon module based semiconductor design, it is expected that the Compute Unit concept will be able to provide a reuse model for such modules, and that the hardware transaction level interconnect of the Unimem system will finally close the designed questions of DSM across a cluster and once again deliver hardware native support for shared memory between nodes in scalable and low cost cluster of computers.

## REFERENCES


[1] [n. d.]. AMBA AXI specs. http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf. Accessed: 2019-05-31.

[2] [n. d.]. Silicon Graphics – SGI – Computing History. http://www.computinghistory.org.uk/det/8312/Silicon-Graphics-SGI

[3] 2017. Co-designed Innovation and System for Resilient Exascale Computing in Europe: From Applications to Silicon. http://cordis.europa.eu/project/rcn/210095{_}en.html

[4] 2019. The Vulcan Microarchitecture. https://en.wikichip.org/wiki/cavium/microarchitectures/vulcan

[5] Sarita V Adve and Kourosh Gharachorloo. 1996. Shared memory consistency models: A tutorial. *computer* 29, 12 (1996), 66–76.

[6] Gene M Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 483–485.

[7] Roberto Ammendola, Andrea Biagioni, Fabrizio Capuani, Paolo Cretaro, Giulia De Bonis, Francesca Lo Cicero, Alessandro Lonardo, Michele Martinelli, Pier Stanislao Paolucci, Elena Pastorelli, et al. 2018. Large Scale Low Power Computing System-Status of Network Design in ExaNeSt and EuroExa Projects. *arXiv preprint arXiv:1804.03893* (2018).

[8] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, et al. 2013. Spin-transfer torque magnetic random access memory (STT-MRAM). *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 9, 2 (2013), 13.

[9] Ryo Asai. 2016. MCDRAM as High-Bandwidth Memory (HBM) in Knights Landing processors: developers guide. *Colfax International* (2016).

[10] Amro Awad, Simon Hammond, Clay Hughes, Arun Rodrigues, Scott Hemmert, and Robert Hoekstra. 2017. Performance analysis for using non-volatile memory DIMMs: opportunities and challenges. In *Proceedings of the International Symposium on Memory Systems*. ACM, 411–420.

[11] Manu Awasthi, Tameesh Suri, Zvika Guz, Anahita Shayesteh, Mrinmoy Ghosh, and Vijay Balakrishnan. 2015. System-level characterization of datacenter applications. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, 27–38.

[12] Brian W Barrett, Ron Brightwell, Scott Hemmert, Kevin Pedretti, Kyle Wheeler, Keith Underwood, Rolf Riesen, Arthur B Maccabe, and Trammell Hudson. 2012. The Portals 4.0 network programming interface. *Sandia National Laboratories, November 2012, Technical Report SAND2012-10087* (2012).

[13] Rupak Biswas, Dochan Kwak, Cetin Kiris, and Scott Lawrence. 2006. Impact of the Columbia supercomputer on NASA space and exploration missions. In *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)*. IEEE, 8–pp.

[14] Richard H Bruce, William P Meuli, Jackson Ho, et al. 1989. Multi chip modules. In *26th ACM/IEEE Design Automation Conference*. IEEE, 389–393.

[15] Cavium. 2018. ThunderX2®CN99XXProduct Brief. (2018). https://www.marvell.com/documents/cmvd78bk8mesogdusz6t/

[16] Bradford L Chamberlain, David Callahan, and Hans P Zima. 2007. Parallel programmability and the chapel language. *The International Journal of High Performance Computing Applications* 21, 3 (2007), 291–312.

[17] Barbara Chapman, Tony Curtis, Swaroop Pophale, Stephen Poole, Jeff Kuehn, Chuck Koelbel, and Lauren Smith. 2010. Introducing OpenSHMEM: SHMEM for the PGAS community. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*. ACM, 2.

[18] Matthew Chapman and Gernot Heiser. 2009. vNUMA: A Virtual Shared-Memory Multiprocessor.. In *USENIX Annual Technical Conference*. 349–362.

[19] Cisco. 2016. *Cisco UCS B420 M3 High Performance Blade Server Installation and Service Note No Title*. Technical Report.

[20] Luke Collins. 2013. DATE: ARM proposes âĂŸunit of compute' as basis for energy-efficient systems. http://www.techdesignforums.com/blog/2013/03/22/date-arm-unit-of-compute-energy-efficient-systems/

[21] CCIX consortium. 2018. *An introduction to CCIX*. Technical Report. https://docs.wixstatic.com/ugd/0c1418_c6d7ec2210ae47f99f58042df0006c3d.pdf

[22] Gen-Z consortium. 2018. *Gen-Z overview*. Technical Report. https://genzconsortium.org/wp-content/uploads/2018/05/Gen-Z-Overview-V1.pdf

[23] Alan L Cox, Robert J Fowler, and Jack E Veenstra. 1990. *Interprocessor invocation on a numa multiprocessor*. Technical Report. ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE.

[24] System Architecture Manager Dan Bouvier. 2003. RapidIO: The Interconnect Architecturefor High Performance Embedded Systems. (2003).

[25] Inphi Corp. David Wang, Senior Principal Architect. 2013. *Why migrate to DDR4?* Technical Report. https://www.eetimes.com/document.asp?doc_id=1280577

[26] Saïd Derradji, Thibaut Palfer-Sollier, Jean-Pierre Panziera, Axel Poudes, and François Wellenreiter Atos. 2015. The BXI interconnect architecture. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 18–25.

[27] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast remote memory. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*. 401–414.

[28] Yves Durand, Paul M Carpenter, Stefano Adami, Angelos Bilas, Denis Dutoit, Alexis Farcy, Georgi Gaydadjiev, John Goodacre, Manolis Katevenis, Manolis Marazakis, et al. 2014. Euroserver: Energy efficient node for european microservers. In *2014 17th Euromicro Conference on Digital System Design*. IEEE, 206–213.

[29] Tarek El-Ghazawi and Lauren Smith. 2006. UPC: unified parallel C. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 27.

[30] Shaked Flur, Kathryn E Gray, Christopher Pulte, Susmit Sarkar, Ali Sezgin, Luc Maranget, Will Deacon, and Peter Sewell. 2016. Modelling the ARMv8 architecture, operationally: concurrency and ISA. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 608–621.

[31] Dimitris Giannopoulos, Nikos Chrysos, Evangelos Mageiropoulos, Giannis Vardas, Leandros Tzanakis, and Manolis Katevenis. 2018. Accurate congestion control for RDMA transfers. In *Proceedings of the Twelfth IEEE/ACM International Symposium on Networks-on-Chip*. IEEE Press, 3.

[32] Marshall Gunnell. 2018. Samsung Begins 7nm EUV Production, Unveils Next Generation NAND SSD & DRAM. https://www.storagereview.com/samsung_begins_7nm_euv_production_unveils_next_generation_nand_ssd_dram

[33] Alex Hutcheson and Vincent Natoli. 2011. Memory bound vs. compute bound: A quantitative study of cache and memory bandwidth in high performance applications. In *Technical report, Stone Ridge Technology*.

[34] Bruce Jacob. 2015. The 2 petaFLOP, 3 petabyte, 9 TB/s, 90 kw cabinet: a system architecture for exascale and big data. *IEEE Computer Architecture Letters* 15, 2 (2015), 125–128.

[35] Meenatchi Jagasivamani, Candace Walden, Devesh Singh, Luyi Kang, Shang Li, Mehdi Asnaashari, Sylvain Dubois, Bruce Jacob, and Donald Yeung. 2018.

Memory-systems challenges in realizing monolithic computers. In *Proceedings of the International Symposium on Memory Systems*. ACM, 98–104.

[36] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*. IEEE, 1–4.

[37] A Kagi, James R Goodman, and Doug Burger. 1996. Memory bandwidth limitations of future microprocessors. In *23rd Annual International Symposium on Computer Architecture (ISCA'96)*. IEEE, 78–78.

[38] Manolis Katevenis, Nikolaos Chrysos, Manolis Marazakis, Iakovos Mavroidis, Fabien Chaix, N Kallimanis, Javier Navaridas, John Goodacre, Piero Vicini, Andrea Biagioni, et al. 2016. The exanest project: Interconnects, storage, and packaging for exascale systems. In *2016 Euromicro Conference on Digital System Design (DSD)*. IEEE, 60–67.

[39] Stefanos Kaxiras, David Klaftenegger, Magnus Norgren, Alberto Ros, and Konstantinos Sagonas. 2015. Turning centralized coherence and distributed critical-section execution on their head: A new approach for scalable distributed shared memory. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 3–14.

[40] Peter J Keleher, Alan L Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. 1994. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems.. In *USENIX Winter*, Vol. 1994. 23–36.

[41] Seong Keun Kim, Sang Woon Lee, Jeong Hwan Han, Bora Lee, Seungwu Han, and Cheol Seong Hwang. 2010. Capacitors with an equivalent oxide thickness of< 0.5 nm for nanoscale electronic semiconductor memory. *Advanced Functional Materials* 20, 18 (2010), 2989–3003.

[42] Seong Keun Kim and Mihaela Popovici. 2018. Future of dynamic random-access memory as main memory. *MRS Bulletin* 43, 5 (2018), 334–339.

[43] Christoph Lameter et al. 2013. NUMA (Non-Uniform Memory Access): An Overview. *Acm Queue* 11, 7 (2013), 40.

[44] Joshua Lant, Andrew Attwood, Javier Navaridas, Mikel Lujan, and John Goodacre. 2019. Receive-Side Notification for Enhanced RDMA in FPGA Based Networks. In *International Conference on Architecture of Computing Systems*. Springer, 224–235.

[45] Joshua Lant, Caroline Concatto, Andrew Attwood, Jose A Pascual, Mike Ashworth, Javier Navaridas, Mikel Luján, and John Goodacre. [n. d.]. Enabling shared memory communication in networks of MPSoCs. *Concurrency and Computation: Practice and Experience* ([n. d.]), e4774.

[46] Daniel Lenoski. 1998. Multiprocessor design options and the Silicon Graphics S2MP architecture. *Computer physics communications* 110, 1-3 (1998), 59–68.

[47] Shang Li, Dhiraj Reddy, and Bruce Jacob. 2018. A performance & power comparison of modern high-speed DRAM architectures. In *Proceedings of the International Symposium on Memory Systems*. ACM, 341–353.

[48] Marios Asiminakis Nikos Chrysos Vassilis Papaeustathiou Pantelis Xirouchakis Michalis Gianoudis Nikolaos Dimou Antonis Psistakis Panagiotis Peristerakis Giorgos Kalokairinos Manolis Ploumidis, Nikolaos D. Kallimanis and Manolis Katevenis. 2019. Software and Hardware co-design for low-power HPC platforms.

[49] Milo Martin, Mark D Hill, and Daniel J Sorin. 2012. Why on-chip cache coherence is here to stay. (2012).

[50] Collin McCurdy and Jeffrey Vetter. 2010. Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms. In *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 87–96.

[51] Sparsh Mittal, Jeffrey S Vetter, and Dong Li. 2015. A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. *IEEE Transactions on Parallel and Distributed Systems* 26, 6 (2015), 1524–1537.

[52] Constance Moreau-Luchaire, C Moutafis, Nicolas Reyren, João Sampaio, CAF Vaz, N Van Horne, Karim Bouzehouane, K Garcia, C Deranlot, P Warnicke, et al. 2016. Additive interfacial chiral interaction in multilayers for stabilization of small individual skyrmions at room temperature. *Nature nanotechnology* 11, 5 (2016), 444.

[53] David Mulnix. 2017. *Intel® Xeon® Processor Scalable Family Technical Overview | Intel® Software*. Technical Report. https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview

[54] Richard Murphy, Arun Rodrigues, Peter Kogge, and Keith Underwood. 2005. The implications of working set analysis on supercomputing memory hierarchy design. In *Proceedings of the 19th annual international conference on Supercomputing*. ACM, 332–340.

[55] Ike Nassi. 2017. Scaling the Computer to the Problem: Application Programming with Unlimited Memory. *Computer* 50, 8 (2017), 46–51.

[56] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. 2015. Latency-tolerant software distributed shared memory. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*. 291–305.

[57] Robert W Numrich and John Reid. 1998. Co-Array Fortran for parallel programming. In *ACM Sigplan Fortran Forum*, Vol. 17. ACM, 1–31.

[58] Mike OâĂŹConnor. 2014. Highlights of the high-bandwidth memory (hbm) standard. In *Memory Forum Workshop*.

[59] Kyriakos Paraskevas, Nikolaos Chrysos, Vassilis Papaefstathiou, Pantelis Xirouchakis, Panagiotis Peristerakis, Michalis Giannioudis, and Manolis Katevenis. 2018. Virtualized Multi-Channel RDMAwith Software-Defined Scheduling. *Procedia Computer Science* 136 (2018), 82–90.

[60] J Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *2011 IEEE Hot chips 23 symposium (HCS)*. IEEE, 1–24.

[61] Gerald J Popek and Robert P Goldberg. 1974. Formal requirements for virtualizable third generation architectures. *Commun. ACM* 17, 7 (1974), 412–421.

[62] Milan Radulovic, Darko Zivanovic, Daniel Ruiz, Bronis R de Supinski, Sally A McKee, Petar Radojković, and Eduard Ayguadé. 2015. Another trip to the wall: How much will stacked dram benefit hpc?. In *Proceedings of the 2015 International Symposium on Memory Systems*. ACM, 31–36.

[63] Simone Raoux, Geoffrey W Burr, Matthew J Breitwisch, Charles T Rettner, Yi-Chou Chen, Robert M Shelby, Martin Salinga, Daniel Krebs, Shih-Hung Chen, Hsiang-Lan Lung, et al. 2008. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development* 52, 4/5 (2008), 465.

[64] Alvise Rigo, Christian Pinto, Kevin Pouget, Daniel Raho, Denis Dutoit, Pierre-Yves Martinez, Chris Doran, Luca Benini, Iakovos Mavroidis, Manolis Marazakis, et al. 2017. Paving the way towards a highly energy-efficient and highly integrated compute node for the Exascale revolution: the ExaNoDe approach. In *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE, 486–493.

[65] Josep Rius. 2013. IR-drop in on-chip power distribution networks of ICs with nonuniform power consumption. *IEEE transactions on very large scale integration (VLSI) systems* 21, 3 (2013), 512–522.

[66] Paul Rosenfeld, Elliott Cooper-Balis, Todd Farrell, Dave Resnick, and Bruce Jacob. 2012. Peering over the memory wall: Design space and performance analysis of the Hybrid Memory Cube. *Technical Report UMD-SCA-2012–10-01* (2012).

[67] Einar Rustad. 2007. A high level technical overview of the NumaConnect technology and products. (2007).

[68] ScaleMP. 2018. *The Versatile SMPâĎć (vSMP)Architecture*. Technical Report. https://www.scalemp.com/wp-content/uploads/2018/10/vSMP-Foundation-White-Paper{_}2012-08-1.pdf

[69] SGI. 2006. *Press release, , SGI Altix Again Crushes World Record for Memory Bandwidth*. Technical Report.

[70] Pankaj Sharma, Qi Zhang, Daniel Sando, Chi Hou Lei, Yunya Liu, Jiangyu Li, Valanoor Nagarajan, and Jan Seidel. 2017. Nonvolatile ferroelectric domain wall memory. *Science advances* 3, 6 (2017), e1700512.

[71] Shyh-Shyuan Sheu, Meng-Fan Chang, Ku-Feng Lin, Che-Wei Wu, Yu-Sheng Chen, Pi-Feng Chiu, Chia-Chen Kuo, Yih-Shan Yang, Pei-Chia Chiang, Wen-Pin Lin, et al. 2011. A 4Mb embedded SLC resistive-RAM macro with 7.2 ns read-write random-access time and 160ns MLC-access capability. In *2011 IEEE International Solid-State Circuits Conference*. IEEE, 200–202.

[72] Sayantan Sur, Matthew J Koop, and Dhabaleswar K Panda. 2006. High-performance and scalable MPI over InfiniBand with reduced memory usage: an in-depth performance analysis. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 105.

[73] Sayantan Sur, Matthew J Koop, Dhabaleswar K Panda, et al. 2007. Performance analysis and evaluation of Mellanox ConnectX InfiniBand architecture with multi-core platforms. In *15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007)*. IEEE, 125–134.

[74] Brian Van Essen, Roger Pearce, Sasha Ames, and Maya Gokhale. 2012. On the role of NVRAM in data-intensive architectures: an evaluation. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 703–714.

[75] Wm A Wulf and Sally A McKee. 1995. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.

[76] Yiying Zhang and Steven Swanson. 2015. A study of application performance with non-volatile main memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–10.

## ACKNOWLEDGMENTS

This work is supported by the European Commission under the Horizon 2020 Framework Programme for Research and Innovation through the EuroEXA project (grant agreement 754337)