GEN-Z & EMERGING SYSTEM REQUIREMENTS

DATA WILL CHANGE THE ARCHITECTURE & OPERATION OF FUTURE SYSTEMS

EXECUTIVE SUMMARY

The ultimate objective of modern information technology (IT) systems is to transform data into information with actionable business insights. Businesses demand security and data protection throughout this entire process. Insights must be virtually immediate, derived from data of unpredictable volume, from hosts in many different locations, and from an increasing number of diverse sources. These requirements, especially size and the need for extremely low latency access, are changing the way systems handle data and the supporting architecture.

Gen-Z is a new communication / interconnect standard proposed to meet these emerging requirements. The operation of Gen-Z is both scalable and reliable and uses a simple, familiar scheme that does not require massive software complexity. Gen-Z appears to fit as the communication mechanism for the data domain. This paper explores the attributes of systems required to work at this scale and how Gen-Z is a candidate for their fulfillment.

A DATA-CENTRIC FUTURE

A data-centric future means that data will be the focal point of just about everything. Similar claims have been associated with the dawn of the information age (late 1970s / early 1980s) and the economic transition away from value derived from industry toward value derived from information. This change ushered in the development of technology enabling profound shifts in business practice. Considering the explosion of data (according to IDC, 44ZB by 2020 and 180ZB by 2025) combined with tens of billions of IoT data sources, it is clear that a new information age has already begun.

Computing has always focused on increasing the speed, accuracy, and capability of mathematical calculations performed manually for millennia. Initially, information (knowledge) was derived from carefully inspecting well-supervised data entered manually; once captured, an expert extracted value and insight from this data. As automation matured, less manual attention was required, which allowed the elimination of many operations. Thus began the quest for more capability and the desire to perform more work on common datasets. Processing quickly became the precious resource. However, it simultaneously created an impediment as it overshadowed the very object

of its attention: the data. In the 1990s, challenges with processing enormous volumes of data began to appear, and massively parallel processing (MPP) database systems (data warehouses) emerged. These systems were the first to overcome problems associated with extremely large scale data by "sharing nothing" (common data was not shared among processing agents). MPP increased system complexity and began the slow, inevitable shift from a processing-centric world to a data-centric world.

The internet and the requirement to complete fast search over large, ever-changing datasets were an important step toward the future. In the last few years as business practices have evolved, the collection of more data and the need to analyze it are commonplace. Additionally, the nature of the data itself has changed into what is now termed "big data", which usually refers to an unstructured dataset. Today, solutions are built on Hadoop or similar frameworks and continue to evolve at an astonishing rate.

The next step in system advancement is emerging data-centric systems. Such systems are primarily connected by the data or at least the domain that underpins it. "Big data" will expand to "gigantic data" and include a mixture of static, relatively constant data and a flood of real-time, streaming data. This new reality requires the supporting system architecture to enable the same objective but with expanded scale and scope.

- The objective of the emerging data-centric system architecture is to transform data into information with actionable insights, but doing so with data of unpredictable volume, from hosts in many different locations, and from an increasing number of diverse sources.

SOFTWARE IS A REVOLUTION

Software has been in a revolution since the beginning of the computer industry. History records many notable architectural shifts that have defined much of what exists today. At a high level, significant accomplishments include (but are not limited to):

- Sequential code execution
- Parallel computing
- Time domain multitasking
- Multithreading
- Multiprocessing optimized multithreading
- Massively parallel processing (MPP)
- Machine virtualization
- Multi-machine processing (search)
- Hyperscale cloud computing
- Cloud infrastructure (OpenStack)
- Big data frameworks
- Machine learning
- Explosion of data & data sources

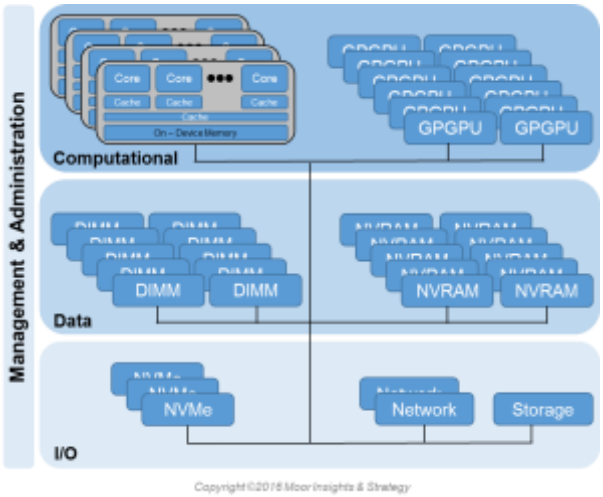
The next predicted step is the shift to containers and highly orchestrated micro-services, some of which will have an extremely short life and be unpredictable about where they are born and live.

- Future applications will most likely evolve to be intertwined collections of containerized services, requiring very straightforward and fast data access and intercommunications.

System Domains & the Scaling Challenge

A typical system has four functional domains (three primary and one support) as shown in Figure 1. A data-centric system will change this architecture significantly to include not only the discrete domains but also their interaction with other systems, specifically how data is accessed and exchanged.

Figure 1: System Domains



Source: Moor Insights & Strategy

Because of its focus on processing, the **computational domain** is well known due to Moore's Law (Appendix I. Moore's Law). This domain contains the elements that perform operations on data as part of information extraction. These elements include processors (CPUs) and accelerators (GPGPUs) and can include other devices (ASICs, FPGAs) considered accelerators that may perform specialized functions. Undoubtedly, multiple agents sharing data will become more prevalent in the future. The industry is familiar with difficulties associated with data sharing, coherency, and interconnects like the Scalable Coherent Interface or SGI's NUMALink (Appendix II. Caching Operation Coherency, III. The Work of von Neumann & Amdahl, IV. Coherent Bus Limitations).

System memory is the “seat” of the **data domain** and is the primary vehicle that holds active data while it undergoes transformation. It also has enjoyed the fruits of Moore’s Law. Today the system memory controller resides inside the CPU, making the memory bus (the ‘front side” bus for x86) little more than the interface to the DRAM array. Technology’s impact has been profound in the memory system as it blurs with both storage and computational elements. The introduction of on-device memory moves some of the memory closer to the CPU, and the addition of storage class memory (SCM) moves some of storage functions into memory. Both on-device memory and SCM are included in Figure 1 and require the support of split-transaction operations (Appendix V. Memory & Split or Deferred Operations).

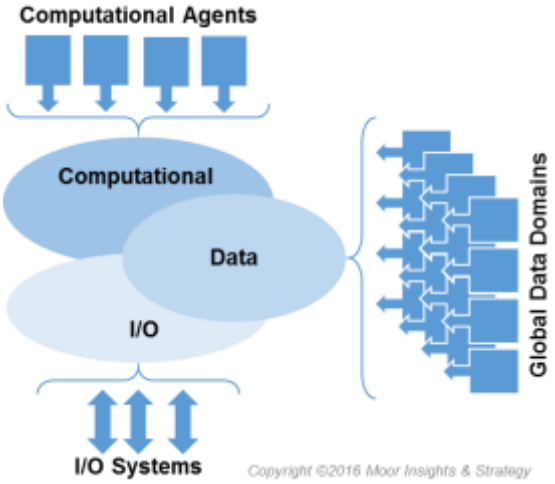
Scaling the data domain itself is one of the greatest challenges of a data-centric system. This domain is not only responsible for data but must also provide a mechanism that controls access to shared resources (Appendix VI. Semaphores & Lock Operations). This domain must further ensure rights, privilege, and credentials (R/P/C) are maintained and protected. Including this overhead in the computational domain not only adds extraordinary complexity but a real-time performance impact. R/P/C must be validated and remediated at the point and time of incursion, not stored for deferred action. Including this capability in the data domain seems to make the most sense.

The data domain is exposed to the greatest expansion and must scale from a node, to multiple nodes, and then to racks of nodes. This domain also provides the missing element of the rack scale architecture and the decomposition of the CPU / memory structure. Today, memory is “welded” to each CPU / server complex. It is unlikely this strong bonding will be entirely dismissed (especially with on-device memory), but it must be modified to enable more plausible composable solutions. Further, the memory and associated technology must be allowed to evolve independently of the CPU.

The **I/O domain** has a specific role and has been historically hardware-defined but is becoming more software-defined. This domain will continue to be necessary for complex I/O and system expansion development. Much of the networking industry relies on its existence, and it is essential to the industry’s long-term evolution and success.

Providing data to the computational domain and the I/O domain is part of the system challenge. Equally important is making data appear as accessible and as ubiquitous as possible. As performance demands increase, there will be no time for complex decisions, and hardware must facilitate simplicity. Ideally, all data will appear as local, even though it may come from another data domain or even a remote node (Appendix VII. Work in the Wrong Domain). Figure 2 shows the resulting data-centric system.

FIGURE 2: DATA-CENTRIC SYSTEM



Source: Moor Insights & Strategy

- The introduction of on-device memory and SCM requires a structural change to the system memory architecture and the support of split-transaction operations.
- The data domain must ensure rights, privilege, and credentials (R/P/C) are maintained and protected.
- Memory and associated technology must evolve independently of the CPU.
- Data access complexity requires simplification by hardware.

THE DATA DOMAIN & GEN-Z, A CANDIDATE INTERCONNECT

Gen-Z is a communication and interconnect candidate for the data domain. It is designed to provide a scalable and reliable communication mechanism that meets the needs described in this paper. It is easy to support a separate cache coherent mechanism in the computational domain for processing and accelerators, but for the data domain, Gen-Z seems a good choice. It uses a memory-semantic (read and write) protocol that enables multiple device types to communicate efficiently using a straightforward and familiar scheme, without introducing software complexity. Its split-transaction protocol supports the disaggregated system memory brought about by high-speed on-chip memory and storage class memory.

Gen-Z specifies a set of mandatory memory-semantic operations (read and write) and a set of controlling operations (including locks and buffer movement). It also includes a set of application-specific semantic actions that add a high-level abstraction, which helps hide underlying component-specific microarchitectures.

Security and data assurance features are embedded in the interconnect, enabling R/P/C enforcement and support. Gen-Z secures them at the link level and is not bound to any encryption / decryption scheme. They are not considered link attributes and can be added more appropriately at a higher level. Assurance features include a set of detection and operational parameters that not only ensure an operation completes as expected but can determine and detect if an unexpected device appears in an existing path. If the time to complete an operation is altered, it is detected. Other tamper prevention and detection schemes also exist.

Gen-Z separates the core architecture and its associated functionality from the physical layer. This separation allows them to evolve independently and capitalize on technology advancements that will enable scaling to multiple racks. Potential physical layer advancements include silicon photonics and improvements in copper cabling.

Many different cache coherency schemes exist, and each creates a substantial challenge. Gen-Z tackles this problem with a set of lightweight primitives that shield the interconnect and architecture from this burden. Because Gen-Z operations are split-transactions, the supplier of the data can easily resolve (and must ensure) local coherency is established before any response. Unless explicitly specified, ownership of the data defaults where the data resides. In its base operations, any plan to cache the data in a remote cache must assume the original copy is the coherent copy.

Gen-Z supports options allowing different coherency schemes but in application specific overlays. Its architecture includes and mandates specified functionality that ensures basic interoperability (read-write memory semantic communications). Even as higher level specific functions appear, the system must operate correctly to ensure security and consistency in these basic operations. Additional features must be considered optional. These options can provide more efficient operation for participating parties.

Gen-Z is designed as a solution for scaling / interconnecting and appears to be a good candidate for the data domain. However, its connection to the other domains is not without challenge and can take three forms.

- Computational agents (CPUs, *etc*.) can be developed and include specific Gen-Z interfaces that interconnect to the system. This option is most likely the highest value option and offers the greatest promise.
- Bridging solutions can be designed that interconnect specific domains to Gen-Z domains. Candidates include all existing and emerging CPU interconnects such as CCIX, OpenCAPI, AMBA or CoreLink, and Intel's UPI.

- PCIe-based solutions are conceivable but will not provide the ability to optimally forward data to the computational domain. It can create a separate shared data domain that would address the need to provide a vastly scaled data system.

Gen-Z is designed to resolve the data domain challenges of data-centric systems.

- The amount and new types of data require larger memory capacity.
- Memory capacity, access, scale, and latency are key performance requirements.
- The data domain and all types of memory allow independent innovation and evolution without impact to the other domains.
- Data access avoids the need to move data around.
- Data security and R/P/C capability is embedded in the interconnect.

Gen-Z is designed to provide the capability needed to integrate SCM both inside and outside of the box. The best results will occur when it is incorporated into the compute silicon, but good solutions can be provided by bridging to a high-speed, coherent bus.

AN OPEN INDUSTRY STANDARD

Gen-Z is an open industry standard. IT has historically been governed by standards that reflect something the industry finds valuable. Many successful industry standards are "open", avoiding lock-in controlled by a particular vendor or group. Although there is no formal definition of an open industry standard, Gen-Z shares characteristics that maximize success, including:

- Available to the general public
- Intended to promote widespread adoption and distribution
- Based on a simple and straightforward licensing agreement
- Developed and maintained through a collaborative process
- Includes some form of validation or adherence process

A colloquial term, "standard-washing", means releasing something to a standard body or as a standard after it is operational. Sometimes, it is an effort to speed or streamline development. In such cases, one should question the incumbents' advanced knowledge or experience and whether it presents a long-term advantage. Gen-Z is attempting to be a genuinely open standard and avoid these problems.

Gen-Z has a foundation responsible for its stewardship. Genzconsortium.org describes who they are, how they are selected, for how long, and how decisions are made.

A Peek into the Future

In 2025, technologists will look back at 2017 and measure predictions against their present realities. They will realize that the explosion of data and diverse sources was understated. The internet of things (IoT) and the number of machine-to-machine transactions were vastly underestimated. The real data includes entire businesses built from new and unique forms of information derived from data obtained and correlated from other systems and devices. The physical boundaries of many systems are undetectable, known only by the services they provide. A simple and unburdened data domain exists and uses Gen-Z as the standard interconnect for its simple memory semantic programming model. Applications evolved into an interlocked set of service-based containers that appear and disappear as needed in just the right place and at just the right time. These containers are also physically unbounded and enabled by strict APIs. These APIs support the necessary micro-services that abstract them from the physical capabilities underpinning them on specific platforms. Data is equally available to all requestors with the rights and privileges controlled by a key system embedded into the fabric of the interconnect itself. Underutilization is almost extinct, and excess capacity is tightly managed, delivering energy utilization well below the 20 pico-joule goal set by the Department of Energy. Big data has morphed into a Lambda like system where decisions are made in real-time considering static knowledge derived from information at rest and its companion streaming data. Lifecycle management of data is fully automated using machine analytics to ensure data is at the right place at the right time. Data migration is only required when there is a substantial gain.

Call to Action

Moor Insights & Strategy (MI&S) believes this paper reflects many of the requirements and trends necessary for emerging systems. We think the introduction of on-device and storage class memory will change both the memory and storage system. Further, we believe that Gen-Z has the potential to address many of the challenges facing the industry as an open industry standard. For companies considering the impact of data on future systems, MI&S suggests adding Gen-Z as an architectural consideration and participating in its development. For further information about Gen-Z Consortium including membership details, see genzconsortium.org.

APPENDIX

I. MOORE'S LAW

From 1965 to 1975, Gordon Moore developed the principal that stated the economics associated with manufacturing electronic components on silicon would double every two years. History has proven this to be accurate. Moore's Law has had a two-fold impact: a doubling in density (the profound economic value on cost) and a doubling in device speed (the equally profound performance contribution).

II. CACHING OPERATION COHERENCY

A cache is a small (usually) memory capable of operating at or near the maximum rate of the computational agent. Cache operation is based on the proven principle that once data is involved in an operation, it is extremely likely to be used again. Caching specifically "write-back" caching provides a vital performance gain and introduces the need for data coherency.

As systems evolved, caching techniques were created to improve performance by minimizing the impact of slower memory on processing agents. Caching introduced the need for data coherency, a process that ensures valid data is always used by agents sharing it. Note that while coherency techniques work well, performance can degrade as the number of agents participating in this coherency increases. Moreover, there are "fringe" cases where it is difficult to ensure proper operation. If something goes wrong, or a transaction does not complete, data corruption can result. Unfortunately, both the overhead and the number of potential fringe cases rise exponentially with the number of agents. Caching also revealed the condition known as non-uniform memory access (NUMA). Simply stated it means that not all memory elements perform at the same speed (latency and bandwidth), and the system must accommodate their difference.

Write-back caching and data sharing require several mechanisms that ensure the correct data is always used by the system.

- Data can exist without issue in multiple caches and be used by multiple agents as long as it has not been modified. In this case, the data in the cache remains consistent with the original copy in memory.
- Data in a cache can be modified locally for performance and not updated to memory until it is necessary or convenient.
- All caches must be interrogated to determine the potential presence and state of candidate data each time it is fetched. This creates the need for cache

coherency, operations that ensure no out-of-date data is used. It is especially difficult where many computational agents are sharing a common dataset.

- If an agent needs to modify the data, it must notify all other agents that any copy of the data they share is about to become stale and should be invalidated in their local cache. This can be indicated by a special operation informing all agents of the intent to modify this data, which is now "owned" by the modifying agent.
- If another agent wishes to use the "owned" (and potentially modified) data, it can ask the owning agent to update the modified data (if any) to memory where all can share it.

III. The Work of von Neumann & Amdahl

The work of many pioneers has set the stage for this time and their principles must be applied. Consider the work of John von Neumann, the basis of most modern systems.

Von Neumann Model: Physicist John von Neumann in 1945

- Both programs and data reside in the same memory with a single linear address space.
- All access methods (address, data, and control) are common and used by all involved elements.

The von Neumann Model works perfectly, but as systems scale, not all memory will always appear in a single von Neumann space (for example, multiple systems that perform the same function in parallel on different parts of a dataset). Multiple interconnected spaces will exist, and a system may access data in another's data domain. This requires both local and remote availability.

Eugene Amdahl also contributed vast knowledge particularly in the area of improving program execution using parallelization.

Amdahl's Law: Eugene Amdahl in 1967

- Programs consist of two parts.
    - An indivisible, single thread that cannot be improved by applying execution units
    - Parallelizable threads whose execution time can be improved by dividing its work among multiple execution units and operating them in parallel

Key factors impacting parallelization

- Time required to dispatch work to individual, parallel execution units and start them
- Time required for parallel units to complete the work
- Time required to coalesce individual results and incorporate into the overall result

Both depend on the number of parallelizable execution units and the time needed to deal with them. Limits of improvement are reached when the combined dispatch and coalesce time exceeds any gain from further parallelization

Amdahl's work focused on scaling and parallelism. He obsessed with combining computational agents to achieve maximum performance and the need to feed data to these units. He became emphatic that anything not essential to directly solving a problem should be avoided and set the groundwork for two driving principles.

- **Work should be performed as close as possible (in time) to the location of data**. Work should be dispatched to the agent that can reach data fastest.
- **Data at rest should remain at rest**. Anything that does not directly contribute to results like moving data should be avoided.

IV. Coherent Bus Limitations

There are limits to the effectiveness of sharing. Many would argue that it begins to taper off quickly at a small number of agents (32) and even turns negative if it is pushed far enough and depending on the amount of actual sharing.

Difficulties with data sharing and coherency are well known. The industry is familiar with interconnects like the Scalable Coherent Interface or SGI's NUMALink. However, external connections (links outside of the "box") are concerning. Coherency is complex and has been the source of several failures. Covering every rare combination or "fringe" case is difficult and is one reasons such interconnects have not generally flourished. If coherency is disrupted, for example, by a simple power failure in a remote system, the entire collection of all agents involved in the coherency scheme can fail. Even Intel has been reluctant to extend QPI (now UPI) beyond the confines of the motherboard largely because of this difficulty and the challenge of driving signaling to any distance.

V. Memory & Split or Deferred Operations

Beginning with processor cores, the first layer includes the caches (L1, L2, and L3) that are closest in time to the processing elements. Process evolution and technology

advancement now make it possible to have GBs of extremely fast on-device memory. System Random Access Memory is more than just DRAM and the functions of the front-side bus that exists today. The arrival of storage class memory adds non-volatile technology back to the memory map and significantly larger compared to DRAM or on-device memory but with higher latency.

Using the deferred response mechanism, in the case of a read a request for data is made to memory in a single terminated transaction. In the case of a write, the data is included in the request. The memory responds later with a memory requested transaction that in the case of a read, supplies the potentially out-of-order data to the requestor or in the case of a write, signals its completion. In the emerging world, system memory will need dual access from the transformation and I/O domain as they exist today and from the memory system itself and any other sharing mechanism.

VI. Semaphores & Lock Operations

An additional consideration is the need to programmatically control the use of globally available resources or software structures in the data domain. Semaphores or locks have been historically used to accomplish this control. A semaphore is a definable software flag used to indicate a state. It is often associated with the availability of a resource or software defined structure and operates as follows:

- If the semaphore is set, it means the region or resource is unavailable.
- If the semaphore is clear, it means the resource is available for use.
- Any agent that uses the resource must first set the semaphore and ensure it is set before proceeding.

The semaphore is usually set by a special operation that is an indivisible function. If the lock tests clear, it is set without the possibility of another agent intervening and setting or clearing it for their purpose.

VII. Work in the Wrong Domain

The application of agents like GPGPUs as accelerators is a perfect example of being forced to do something in the wrong domain because of the system architecture. Since, GPGPUs are usually PCIe plug-in cards and do not have direct random access to system memory, the system is forced to move large blocks of data from system memory through the I/O domain into local memory for processing. This argument favors an interconnect supporting the computational domain.

IMPORTANT INFORMATION ABOUT THIS PAPER

AUTHOR

Jimmy Pike, Chief Technical Officer at Moor Insights & Strategy

PUBLISHER

Patrick Moorhead, President & Principal Analyst at Moor Insights & Strategy

EDITOR

Scott McCutcheon, Director of Research at Moor Insights & Strategy

INQUIRIES

Contact us if you would like to discuss this report, and Moor Insights & Strategy will promptly respond.

CITATIONS

This note or paper can be cited by accredited press and analysts but must be cited in-context, displaying author's name, author's title and "Moor Insights & Strategy". Non-press and non-analysts must receive prior written permission by Moor Insights & Strategy for any citations.

LICENSING

DISCLOSURES

Gen-Z Consortium is a research client of Moor Insights & Strategy and commissioned this paper. Moor Insights & Strategy provides research, analysis, advising, and consulting to many high-tech companies mentioned in this paper. No employees at the firm hold any equity positions with any companies cited in this document.

DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors. Moor Insights & Strategy disclaims all warranties as to the accuracy, completeness, or adequacy of such information and shall have no liability for errors, omissions, or inadequacies in such information. This document consists of the opinions of Moor Insights & Strategy and should not be construed as statements of fact. The opinions expressed herein are subject to change without notice.

Moor Insights & Strategy provides forecasts and forward-looking statements as directional indicators and not as precise predictions of future events. While our forecasts and forward-looking statements represent our current judgment on what the future holds, they are subject to risks and uncertainties that could cause actual results to differ materially. You are cautioned not to place undue reliance on these forecasts and forward-looking statements, which reflect our opinions only as of the date of publication for this document. Please keep in mind that we are not obligating ourselves to revise or publicly release the results of any revision to these forecasts and forward-looking statements in light of new information or future events.