9 Critical Analysis of the Modular Supercomputing Architecture

Estela Suarez(1), Norbert Eicker(1,2), Thomas Moschny(3), Thomas Lippert(1,4)

(1) Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Leo Brandt Straße, 52428 Jülich, Germany

(2) Fakultät für Mathematik und Naturwissenschaften, Bergische Universität Wuppertal, Gaußstraße 20, 42119 Wuppertal, Germany

(3) ParTec AG, Possartstraße 20, 81679 Munich, Germany

(4) Goethe-Universität Frankfurt, Frankfurt Institute for Advanced Studies (FIAS). Ruth-Moufang-Straße 1, 60438 Frankfurt am Main, Germany

e.suarez@fz-juelich.de

9.1 Introduction

The modular supercomputing architecture (MSA) is a novel approach to implement heterogeneous supercomputing[125]. MSA's major differentiation to other types of approaches is that it defines a new intermediate level in the computer architecture hierarchy, which is located between the node- and the system levels. In MSA, subsets of nodes are grouped into special "computational modules" according to their common characteristics and algorithmic features of the corresponding subtasks.

For example, CPU-only nodes are put together into a cluster module, GPU accelerated nodes into a booster module, or quantum devices constitute a quantum module. A Modular Supercomputer is born when these modules, each of which is a high performance computer in its own right, are interconnected via a high-speed network, and are integrated by a common software stack that allows the dynamical allocation of resources from and between all modules.

This meta-architecture allows to dynamically reserve and allocate hardware resources and enables users to select the most suitable mix of resources at each time, respecting the characteristics and requirements of their code portions.

In this chapter, the MSA concept is explained in more detail to dispel some frequent misconceptions. For better understanding, MSA is contrasted to the conventional,

[125] E. Suarez, N. Eicker, Th. Lippert, "Modular Supercomputing Architecture: from idea to production", Chapter 9 in Contemporary High Performance Computing: from Petascale toward Exascale, Volume 3, pp 223-251, Ed. Jeffrey S. Vetter, CRC Press. (2019) [ISBN 9781138487079]

approach of tightly integrating all possible kinds of compute and memory elements within each node, and then replicating this entity several thousand times to build up a "monolithic" HPC system. We argue that the two architectural lines are not mutually exclusive, but that their combination by "integrating" a tightly integrated module into MSA can be beneficial to end users and operators.

9.2 Partitions vs. modules

Very diverse application profiles of HPC users, various kinds of processor types, and pressure on budgets for both procurement and operational costs have made heterogeneity of computers the rule rather than an exception (e.g. [126,127,128]). HPC providers deploy systems that combine different kinds of CPUs and accelerators (in general GPUs), organized in various node configurations. Frequently, supercomputers have multiple compute partitions, with different amounts of memory per node, with or without accelerators, even with different numbers or generations of GPUs.

Often the two fundamental questions are raised: when is a heterogeneous computer considered to be an MSA system? What is the difference between heterogeneous computing and modular supercomputing? The answer to these questions lies more in the manner the system can be operated rather than on its specific hardware configuration. It is the software stack that "modularizes" a heterogeneous supercomputer.

As a principle, MSA strives for a homogeneous internal configuration within each hardware module and achieves global heterogeneity by interconnecting the different modules enabling dynamical allocation of compute resources from several modules from a given program or workflow. One reason for this approach is that combining too many different computing resources within a single node makes it very difficult to share them efficiently between users with different requirements for those resources. In addition, many programs use only one variant of processors on such a "fat node" in a given part of code. All of this results in many elements in the supercomputer being idle and potentially continuing to consume power. Such underutilization can be avoided by MSA.

The first MSA system deployed in the DEEP project was a cluster-booster platform where the cluster was composed of general-purpose (Intel Xeon) CPUs on an InfiniBand network, and the booster consisted of many-core accelerators (host-less

[126] http://www-hpc.cea.fr/en/complexe/tgcc-JoliotCurie.htm

[127] https://docs.nersc.gov/systems/cori/

[128] https://www.bsc.es/marenostrum/marenostrum/mn3

Intel Xeon Phi) on an Extoll network[129]. However, this maximal separation (disaggregation) of CPUs and accelerators is one of many potential hardware realizations but it is not the defining criterion of the MSA. As a matter of fact, in most recent modular supercomputers (e.g. JUWELS[130] and MELUXINA[131]) the booster is a GPU-accelerated platform where management-CPUs are used to orchestrate the GPUs. Here, the booster node itself obviously is a heterogeneous system, but the computational power, to the largest extent, is delivered by the GPUs, while the host-CPUs clearly play a secondary role in so far as they mainly support the GPUs to fulfil their task.

It is indeed possible to choose a different interconnect technology for each module, as was the case in the first DEEP prototype, but this is not a criterion for defining modularity. Avoiding gateways and network bridges between modules, as of course expected and experienced on physical systems, leads to better performances. For this reason, the latest MSA systems use a homogenous interconnect and integrate modules in a common fabric.

Therefore, from the hardware point of view, a supercomputer with two or more distinctive partitions can be considered as a modular supercomputer. The decisive criterion for modularity is whether users can, at the same time, reserve resources on multiple modules and can run their applications across them in a distributed manner, performing communication and data transfers between these modules at runtime. What is more, modularity allows dynamically changing the size of the partitions on the modules according to the needs of the codes at runtime.

Modularity as operational and usage mode requires a software stack and programming environment that supports its requirements. The scheduler and resource manager must be aware of the hardware partitions and their features, and provide an interface enabling users to define the mix of resources to be employed in each partition. In the ideal case, dynamic allocation of the diverse resources is supported, so that each compute element is assigned to the job, when the execution of the application phase that needs it, starts, and only then. Outside these phases, these computing resources are available for other jobs. For example, for applications organized as job chains, different time windows can be set up for reserving the individual partitions. These features as well as multi-tenant use of partitions are important topics of research for the effective realization of modularity.

---

[129] N. Eicker, Th. Lippert, Th. Moschny, and E. Suarez, "T*he DEEP Project - An alternative approach to heterogeneous cluster-computing in the many-core era*, Concurrency and computation: Practice and Experience, Vol. 28, p. 2394–-2411 (2016), doi = 10.1002/cpe.3562.

[130] https://apps.fz-juelich.de/jsc/hps/juwels/configuration.html

[131] https://luxprovide.lu/technical-structure/

Modularity must also be enabled in the programming environment and the runtime system. Sections of the application's code have to be programmed and compiled to run on the hardware of the modules where they shall be executed. The various executables must be enabled to communicate with each other (e.g., via MPI or some other communication interface). This requires changes at the lower layers of the programming models that interface to the different kinds of compute (and possibly interconnecting) hardware. All these features were developed and implemented in the ParaStation Modulo[132,133] software suite in the course of the European DEEP projects. Furthermore, profiling and performance analysis tools running on MSA systems must be capable of collecting hardware counters across partitions and understand the correlation between them for modularity-enabled applications.

All these software components together have a common goal: enable each part of an application to utilize the best suitable selection of resources. This goal, aiming at globally maximizing the use of a heterogeneous set of closely interconnected supercomputers, is what characterizes a Modular Supercomputer.

9.3 Data movement

Dividing computing resources into different modules as strived for in MSA could raise concerns about performance degradation in communication and data transfers between computing elements that are separated from each other. We have already argued in Section 9.2 that such segregation is not necessary in a strict sense when one computes in a "modular" manner. Nevertheless, we would like to adduce some arguments addressing concerns about data-movement. Such concerns are often brought forward to favour monolithic supercomputers that integrate many different kinds of compute resources within each node, colloquially called "fat" node.

Let us first state that in most situations of parallel data processing data movements between nodes cannot be avoided. Only so-called embarrassingly parallel problems can work entirely without significant inter-node data movement. For the rest, simple to sophisticated strategies are used to minimize the surface-to-volume ratio, particularly for regular problems. There are data-centric concepts as well to avoid data movement – at the expense of more computational operations or increased memory consumption. All these strategies must be and indeed are applied within system modules in the MSA. Therefore, in the following, we focus on the particular case of inter-module communication only.

[132] ParaStation Modulo. https://par-tec.com/software/

[133] S. Pickartz, Virtualization as an enabler for dynamic resource allocation in HPC, Dissertation, RWTH AachenUniversity, Aachen, 2019. https://doi.org/10.18154/RWTH-2019-02208 .

When switching between different accelerator types, the impact of data movement on performance depends on the volume and frequency of data exchange. For a given application, these factors are correlated with the computational size of the code sections involved in the communication:

i. **Small kernels**: a typical example is often given by the innermost loop of an application, where a small but computationally intensive calculation is repeated at high frequency for a given number of iterations. This kind of computation requires very small latencies and directly profits from intra-node acceleration. Such type of computations are in fact the traditional target of CPU-GPU systems, where the main program is executed on the host CPU and the small – in the sense that they fit on the GPU memory – but computationally intensive kernels are offloaded to the GPU.

ii. **Large code parts**: in complex applications, and especially those that simulate multi-scale or multi-physics phenomena, code partitioning is done at a much coarser level. Different larger portions of the code are responsible for computing specific parts of the overall problem. They most of the time communicate internally within the respective code part, exchanging information with the rest of the code parts relatively infrequently and mainly to share intermediate results and to update parameters. As the different code parts might also have very different structures and requirements, they might profit from different types of hardware. This is where inter-module communication in MSA is required, which happens between larger code elements such as (library) functions. Between such a coarse-grained code partitions, data movement (off module) involves a rather small amount of data to be exchanged compared to module-resident (on-module) data movement.

Therefore, intra-node heterogeneity applies to on-node and on-module computation of smaller code elements (case (i)), while MSA operates off-module on bigger code-structures of an application or workflow, i.e., large code elements (case (ii)).

Increasing the compute-power of a single node by including multiple (heterogeneous) accelerators can be very helpful to speed-up the execution of small code kernels. However, this makes the supercomputer more imbalanced, and therefore less efficient as to running system-wide problems scalably. A very strong pressure is set on the system network, which cannot increase the bandwidth between nodes at the same rate as the increasing computational power inside the nodes does. In consequence, data movement off the node must be avoided, or the advantage gained by the kernel speed-up may be nullified.

Furthermore, data movements between different accelerators inside a highly heterogeneous node are not necessarily cheap either. They would be if all accelerators

could access the same (high-bandwidth) main memory in the node. However, if the main memory is standard DDR-RAM it will always be faster to stay within one single accelerator's (HBM) memory. The situation is even worse when PCIe is involved in linking the memories of the various accelerators, as is the case today. The communication performance between accelerators is then only marginally different from off-node communication. All current monolithic heterogeneous HPC systems connect their computing elements via PCIe, which requires expensive cross-element transfers and leads to a similar impact on data movement as the inter-module communication in MSA does.

The strongest caveat one often hears as to separation of resources in MSA is the occurence of increased latency for inter-module communication. This effect certainly is most acute when the data have to pass network gateways, i.e., when the modules utilize different interconnect technologies and are connected via a network bridge. However, in case the same or a fully compatible network technology is used across the entire MSA and gateways do not need to be involved, the inter-module communication capabilities are indeed comparable in capability and latency to the inter-node communication as given within an HPC module.

But even on a homogeneous network it is obvious that the latency between a CPU on the cluster and a GPU on the booster, is slightly higher than if they were located inside the same node, where they save the hop over the interconnect. It is for this reason that it is not advisable to offload small kernels between modules in MSA. Therefore, as already stated, in contrast to offloading small kernels as done on node (see in case (i)), in MSA code-partitioning is carried out at a much coarser granularity (see case (ii)). Moreover, on these coarse structures, one can benefit from algorithmic methods in order to reduce data movement between the MSA modules. For example, when running larger code parts on the different modules in parallel, communication between the modules is required much less frequently than within the module, dramatically reducing the impact of the inter-module latency. Finally, to accelerate small compute kernels, MSA can resort to exactly the same strategy as one does on the monolithic fat node system (case (i)). MSA can thus take full advantage of the standard strategy for accelerating small computational cores, while providing a massive improvement in speed when accelerating large compute kernels.

In conclusion, the communication and data movement strategy of MSA relies on executing fine-grained communication within the modules, while only coarse-grained state-exchange information is transferred between modules. This allows both the individual application kernels within a module to be accelerated on the nodes, and the entire application workflow between modules to be boosted via a matching set of resources for each large section of code. In contrast, a monolithic system composed

of identical nodes each containing a diversity of compute and acceleration resources has no means to efficiently accommodate the coarse-grained granularity of case (ii), which leads to resource under-utilization.

9.4 Energy efficiency

Many strategies are applied today in HPC centres to optimize energy efficiency. They comprise the use of low-power computing elements and/or accelerators, shutting-down unused resources, holistic system monitoring, optimizing the site-infrastructure and system cooling (e.g., through direct liquid cooling), actively re-using waste heat, etc. All these approaches can profit from MSA in the same manner as known from any other heterogeneous architecture. What is more, MSA operates at a coarse-grained scale that naturally matches the sub-second timescales handled by monitoring and cooling systems. Heterogeneous System-on-Chip (SoC) approaches – which represent the smallest form of intra-node heterogeneity – are governed by much smaller spatial-scales and shorter time-scales (micro- to nanoseconds). Holistic monitoring starting out from this level would require a vertical integration of monitoring capabilities from very deep (SoC-level) up to very high (infrastructure-level). This ambition constitutes a complicated technological challenge and may not be feasible due to timescales involved differing by orders of magnitude[134].

On top of the general methodology to save energy as mentioned above, MSA can increase energy efficiency by applying three additional strategies:

1.) **Targeted hardware scale-out**: the dimensions of the individual MSA-modules are determined by the requirements of the user-portfolio running on the MSA system, as well as by the energy efficiency of its components. For instance, a cluster module, where applications in need for high single-thread performance run, is composed of relatively power-hungry general purpose CPUs and is therefore kept rather small. The booster, on the other hand, which runs highly-scalable applications (or parts thereof) achieves a very high compute performance using more energy-efficient accelerators. In MSA, only this part of the system is scaled-out to thousands or tens-of-thousands of nodes, if needed, in contrast to fat node systems where complex and expensive fat nodes need to be scaled out.

2.) **Tailor system to application needs**: by running each part of the user code on the kind of node that allows best performance, improved application efficiency

[134] An additional aspect is the fact that, due to the electrical capacities in the hardware, neither accurate power measurement nor adequate power and cooling management seem realistic on a time scale of less than a millisecond.

and performance is achieved. The speed-up gained by the individual applications translates into a shorter execution time, which typically leads to lower overall energy consumption.

3.) **Maximize use of resources**: MSA enables dynamic scheduling, reservation and allocation of resources and makes them available for the job only for the relevant time window, while the rest of the time they are free to be used by others. This enables more efficient resource sharing, and therefore achieves a higher utilization of the individual components, reducing idle time and unnecessary energy waste. In contrast, on a monolithic supercomputer with fat nodes, all resources of all utilized fat nodes are blocked during a job's runtime. Sharing of nodes is expected to be inefficient due to the impact of jitter effects induced by co-utilization[135] on such fat nodes.

As far as system scaling is concerned, one might argue against point (1) that in a booster built as a GPU-accelerated system, the necessary amount of (power hungry) host-CPUs also grows with system size. This issue is, however, readily avoided by choosing a suitable, low-power CPU for the booster, as the CPU only needs to manage the GPUs and not to perform relevant application computation. It is expected that the market will offer GPU designs with integrated orchestrator CPU cores in the near future. This would make GPUs much more independent and allow building a true GPU-only booster.

Building "lean" booster nodes employing low-power management-CPUs (or host-less GPUs) also addresses point (3), as it minimizes (eventually even down to zero) the energy consumption of host CPUs, which are among the very few resources in an MSA system that are prone to be idle, since they are less intensively used for application computation.

Here it is worth mentioning that maximum resource utilization (3) is an important advantage of MSA compared to monolithic systems based on highly-heterogeneous (fat) nodes. An increased intra-node heterogeneity leads to underutilization of resources, since for a given job either CPUs or GPUs, but very rarely different accelerators, are simultaneously in use. The unused node-elements run idle and continue to consume power. Given a broad portfolio of applications, this problem cannot simply be overcome by choosing the best-suited accelerator mix for the heterogeneous node, as this will always introduce a fixed ratio between CPUs and accelerators. This ratio will support only a few applications optimally while others have their sweet-spot at higher or lower ratio. MSA, on the other hand, is fully flexible and

[135] F. Petrini, D. J. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8192 Processors of ASCI Q," in ACM Supercomputing, 2003.

dynamic in the assignment of resources even during program execution, which is its most characteristic new feature of MSA.

In order to compensate these limitations of fat nodes, some chip-designers propose the idea of so-called “dark silicon”. It leverages the concept of integrating an amount of computational resources that deliberately would exceed the chip’s actual power envelope, while selectively switching some resources on and off when possible. In principle, this strategy can be equally applied to heterogeneous chip designs by powering off unused accelerators units. However, it is questionable if steering the power is possible at such extremely small time-scales (see case (i) in Section 9.3) required by the tight integration of accelerators within a chip. More importantly, even if the power for the processing elements is switched on only during operation, the investment made for the switched-off elements is lost for this idle time. Taking into account that during the lifetime of an HPC system, the hardware investment is about two thirds of the total cost of ownership, the energy adjustment as just described can only partially compensate for the underutilization. We argue that maximizing resource utilization by MSA is a fundamentally better approach to increase energy efficiency and reduce cost, and increases the total scientific throughput of HPC systems.

Beyond that, the central assumption behind the dark-silicon strategy is that the cost of transistors’ silicon is negligible when compared to the power-consumption of running them. Reaching the end of Moore’s law by now and observing the worrying situation of the silicon industry since 2020 lets us have serious doubts on this underlying assumption of the dark-silicon strategy.

The challenge of connecting the additional transistors should not be neglected either.

Highly integrated systems are widely used in the mobile and embedded markets, where space and power constraints play a crucial role. Need for high energy efficiency together with moderate prices of mass-market components have been arguments for applying similar strategies in HPC. However, mobile and embedded markets are completely different from the HPC market. In mobile devices, a small number of heterogeneous elements (thin cores, fat cores, GPU, memory, flash, modem, AI,…) are interconnected via standardized interfaces and integrated on an SoC. Until now, HPC has not yet settled on a standard interface for the hardware elements, which limits the possible combinations of elements, and the bandwidth demand in between the elements is significantly higher than on the mobile devices. The main motivation for a SoC in mobile devices is the level of integration and low production costs, rather than bandwidth and latency as in HPC. In HPC, high bandwidth and latency requirements lead to the use of highly sophisticated interposers. Considering the technological challenges and the economic constraints, which these intermediate layers are subject to, their feasibility has not really been proven to date. Therefore, the amount of dark

silicon elements is limited by both the technology of the interposer and the cost of the silicon.

9.5 System integration

For more than a decade, standard accelerators have been integrated within fat nodes to achieve very high peak performance. The main disadvantages of this approach, i.e., underutilization of resources and shared network interfaces, have been discussed extensively above. Today, its strongest advantage as to closer integration with the CPU resources is still diminished by the lack of a technology, where CPU and accelerators have access to shared high-bandwidth memory. Heterogeneous chips (e.g., GPUs with integrated CPU cores and dynamical mutual assignment), which are under development, promise access to shared high-bandwidth memories. If such chips reach the market, they will benefit both monolithic and modular architectures that, for example, could build a cheaper and more energy-efficient booster by getting rid of management CPUs.

Interestingly, the MSA technology also enables the coupling of modules that are operated by GPUs from different manufacturers, for example. In this case, it is not so much about accelerating computations in cluster-booster mode, but rather about equipping the overall system with various accelerator technologies. This strategy makes it possible, on the one hand, to make the most suitable technology available to the user in the workflow and, on the other hand, to still make the entire system accessible to applications that have very large memory and computing requirements. Such type of HW requirements can currently only be delivered by MSA.

From a physical system integration perspective, building, maintaining, and operating MSA platforms are just as complex as monolithic systems: the single modules itself are similar to monolithic systems, they just use slim nodes in contrast to fat complex nodes. Interconnecting them is a problem that is solved by using the right system software, as proven by JUWELS. MSA-modules can also be adapted over time to meet new user requirements by substituting modules or adding new ones when enhanced technology emerges. In fact, MSA also opens up opportunities to integrate presumably disruptive technologies into HPC systems, such as neuromorphic devices or quantum computers. They are still in very early development stages, but have already demonstrated impressive performances for some specific applications.

The inclusion of neuromorphic or quantum modules in the MSA might facilitate their adoption by the wider user communities. For example, it has been demonstrated that quantum computers are extremely efficient to solve specific kinds of problems such as high-dimensional optimization scenarios. While it is very unlikely to see a large-scale HPC application executed fully on a quantum computer anytime soon, it seems

worthwhile to explore an application running e.g., on the cluster module of an MSA, which offloads an optimization problem as part of its code to be solved by a quantum module. These types of embedded optimization problems are ideal for MSA, as they consist of a well-isolated and large part of the code, with only small amounts of data being exchanged between the cluster and the quantum module – which is again ideal for a quantum computing system allowing for small data rates only. This coarse-grained quantum-hybrid strategy allows for the exploration of quantum computing especially for applied problems from science and industry already today, in particular when a quantum annealer like a D-Wave system is exploited.

9.6 Application scalability

Another frequently expressed misconception about MSA is the fear of hindering application-scalability by the need to spread the code components across vastly different module architectures until all available compute resources are occupied. However, for the analysis of the scalability of codes on the MSA, only the booster module should be considered. As with Amdahl's law, the maximum problem size and maximum scalability is always given by the highly scalable part of the code that, in MSA, runs on the scalable booster. In addition to that, decoupling the less-scalable code parts from the high-scaling ones and running them on the cluster improves the overall application scalability: the high-scaling part can scale unhindered on the booster, while the low-scaling part is speed-up through the high single-thread performance of the cluster module.

On the other hand, a justified criticism of MSA – or rather of the current software environment – is that it imposes a relatively high burden on application developers to prepare their codes for execution in a multi module mode. First of all, it is emphasized that such code-distribution is an opportunity in MSA not a general obligation. To give an example, highly scalable applications with an intrinsic monolithic structure (e.g., tightly coupled differential operations on regular lattice systems) should never be spanned across modules, but rather run entirely within the booster.

Candidate MSA codes from multiphysics and multiscale applications to be coarse-grained assigned to modules must undergo a series of analyses and transformations: any such application has to be analyzed as to its internal structure and potential performance bottlenecks, code parts need to be identified and ported to the given module architectures using a suitable programming model (e.g., CUDA or OpenACC for GPUs), and scaling studies need to be performed with relevant and suitable use-cases to find their best modus operandi and the appropriate number of nodes on each module. All these steps are summarized in the best practices guide provided as Chapter 8 of this book. Many of the adaptations to optimize application performance

on specific modules (e.g., increase vectorisation, keeping data locality, proper organisation of data structures, etc.) are necessary on any modern heterogeneous compute platform, not only on MSA. The additional MSA-specific considerations are those related to the implementation of a coarse-grained code partition.

The additional effort of porting codes to MSA might scare application developers. While so far, only a few applications are enabled to run in multi-module mode, from a user and computing centre perspective MSA is even beneficial for single-module operations, as the different modules provide a variety of computing resources for a diverse application portfolio of an HPC centre, even if each code runs on only one type of node. Still, in order to improve user experience and to promote the modularization of HPC applications, the MSA software stack is in continuous development in order to make the MSA-specific and the more general code porting actions as comfortably as possible: this is the goal of the EU-funded DEEP-SEA project, which started in April 2021 and will run for three years[136]. It will continue the software development efforts of the DEEP project series, which already delivered an MSA-enabled runtime system (ParaStation Modulo), as well as a scheduler and a resource manager targeting heterogeneity at system level. Advanced features for a better support of compute and memory heterogeneity, enhanced malleability and interoperability features, co-scheduling aspects, and performance portability will be developed in DEEP-SEA.

9.7 Conclusion

The goals of MSA are to offer the best system configuration to a portfolio of applications with very different profiles and requirements, to assign the best suited hardware resources to each of them (and each of their code-parts), and to maximize system usage and energy efficiency by enabling an efficient sharing of compute resources overall. Most of the reservations for which MSA is often criticized and contrasted with other alternative heterogeneous computing approaches have their roots in simple misunderstandings about basic MSA principles.

The MSA is fundamentally different from other heterogeneous computing approaches, and in particular from highly integrated monolithic systems, in that system-level heterogeneity is achieved by combining a set of (rather) homogeneous computational modules, which allows coarse-grained partitioning of application code among these modules. Multi-module execution is foreseen mainly for applications with an intrinsic multi-physics or multi-scale nature. The associated large code parts run within the modules exchanging a limited amount of data between each other at relatively low frequency. Performance is therefore not impacted by the slightly increased inter-

[136] www.deep-projects.eu

module latency. Intra-node heterogeneity, on the other hand, is suitable for low-granularity operations, such as the execution of small but computationally intensive kernels. Here data is exchanged at a much higher rate and low latency is very crucial to achieve performance.

Because the operational levels of both approaches to heterogeneous computing (MSA and highly-integrated node designs) are so different, it suggests itself to combine them. Therefore, the MSA welcomes the inclusion of heterogeneous modules, and, in fact, current MSA systems do contain them. The combination of different modules with diverse node configurations, some homogenous, some heterogeneous, makes MSA extremely flexible and adaptable to any application portfolio. Further benefits include the possibility to scale out only the most energy-efficient modules of the system, keeping the power-hungry modules at a relatively low node count but still available for the user codes that require them, and the ability to include modules based on disruptive computing technologies such as quantum technologies.

9.8 Acknowledgements

The authors thank all the institutions and individuals involved in the DEEP series of projects and, in particular, all the members of the DEEP-EST team who have contributed to the development of the MSA architecture, its prototype hardware implementation, and its software environment.

This work has been partially funded by the European Union’s Seventh Framework (FP7/20017-2013) and Horizon 2020 Framework Programmes for Research and Innovation under grant agreements 287530 (DEEP), 610476 (DEEP-ER), 754304 (DEEP-EST), and 955606 (DEEP-SEA). The present publication reflects only the authors' views. The European Commission is not liable for any use that might be made of the information contained therein.