# Scaling Towards Exaflops with Heterogeneous Architectures

## When hardware meets software

Arne Hensel


Summeracademy in Leysin, August 2016

**Abstract**

In the past 10 years high performance computing became the concern of a wide user base, not just a small high-end community. The promises made by Moore's Law of doubling the transistor density every 18months were interpreted as a never ending story of increasing computing power. Although this principle is still going strong, companies were facing serious difficulties with slowly increasing performance although the transistor density increased. Also high power consumption, too high temperature on the chip, slow memory access, problems with parallelism and inefficiency forced to think straight forward. Thus a new principle of processor design was introduced called heterogeneous architecture. Heterogeneous systems combine several application specific designed processing units onto one chip or connect these units. Thus they are able to extract the assets from every single core to form a powerful alliance of processing units for specific codes. With this new idea in mind the exascale computing is not as far as it seemed with homogeneous systems.

# Contents

Figure 1: Relatively slow cupcake production.

# 1 Introduction

## 1.1 Scientific Objective

No matter what a computer is going to do, whether it is a difficult calculation or processing input from the keyboard, a program has to be executed by the computers processing unit. This program can consist of different algorithms, which tell parts of the processing unit what to do, e.g. perform a floating point operation or write something to the cache. Most algorithms are like baking recipies, tailored for a single person/processor:

- First, do A,
- Then do B,
- Continue with C,
- And finally complete by doing D.

Let us intruduce a bakery with a single baker working there and producing cupcakes (Figure 1). To simplify the analogy let the baker produce one cupcake at a clockcycle. Furthermore we want to be able to not just produce one cupcake at a time, but many cupcakes. Let us assume we need $10^{18}$ cupcakes for our next party, and it is only one clockcycle left. Figure 2 illustrates the situation: We utilize an army of bakers to produce the cupcakes on the exascale. If we compare the computer's floating point operations per second to the bakery's production of cupcakes, we have to ask our self how we can speed up computations to the exa-flop scale?
The bakery-system is set up by using only one type of chef: a baker. Let it thus be called a homogeneous system. This has to be defined a little bit more specific for computer systems.

## 1.2 Homogeneous Systems

A computer can be classified as homogeneous if it's system meets the following requirements:[Bla96]

- The hardware of each processor guarantees the same storage representation and the same results for operations on floating point numbers.

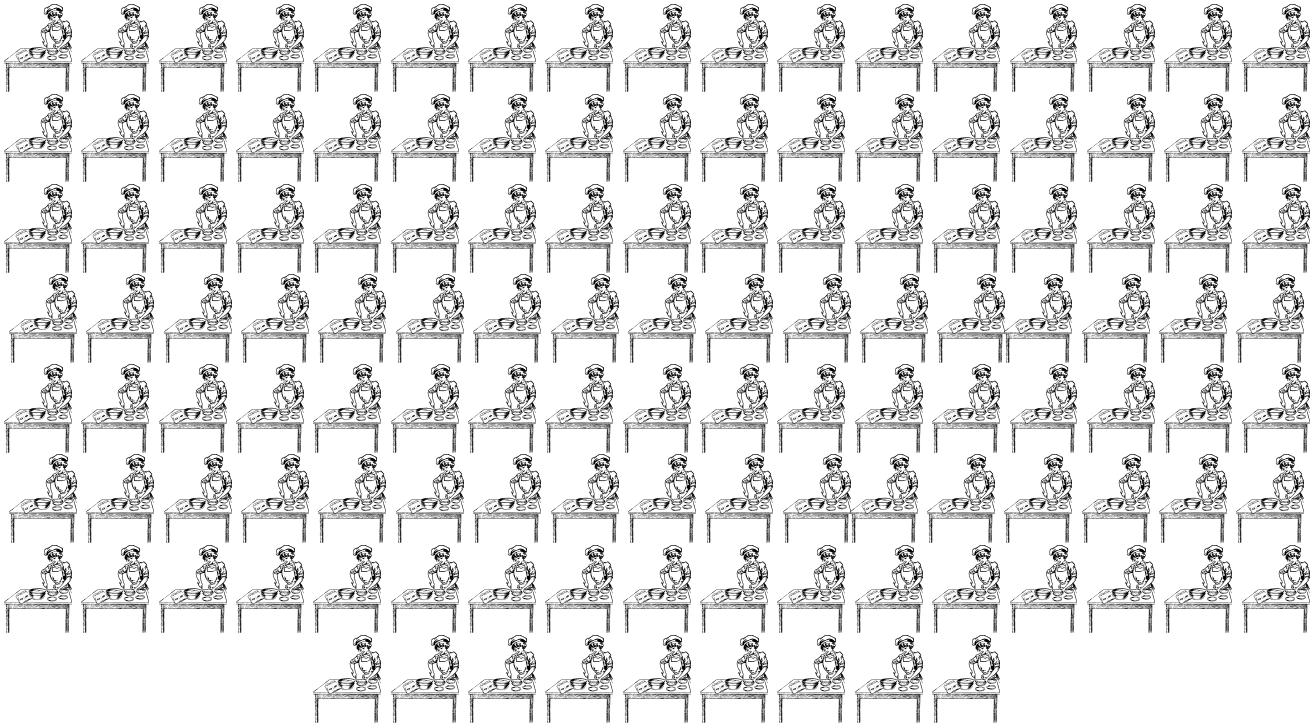

Figure 2: Incredibly fast production of exa-cupcakes

- If a floating point number is communicated between processors, the communication layer guarantees the exact transmittal of the floating point value.
- The software (operating system, compiler, compiler options) on each processor also guarantees the same storage representation and the same results for operations on floating point numbers.

Satisfying these requirements the following approaches have been made:

**Specialized processors** For a given application the processing unit can be designed to increase performance. As an example CPUs are designed for serial tasks while GPUs were invented for massive parallelism and data processing, e.g. for audio and video processing. In general this system includes one chip with one core.

**Multicore-processors** By creating a processing units with many cores we can speed up certain tasks. This principle is used withinn a GPU. Also the Intel Larrabee as a CPU with 32 cores was made for processing graphics with a CPU. In general this system includes one chip with multiple cores.

**Multiprocessors** As a combination of Multicore-processors, multiprocessors use several chips with each utilizing multiple cores. With the right connection between these chips a powerful processing network can be created.

**Cache size** While the clockrate of the processing elements and the transistor density increased very fast, the memory was involved in a development from small to large fast caches.

- 45nm POWER7 with 32MB cache
- 32nm POWER7+ with 80MB cache

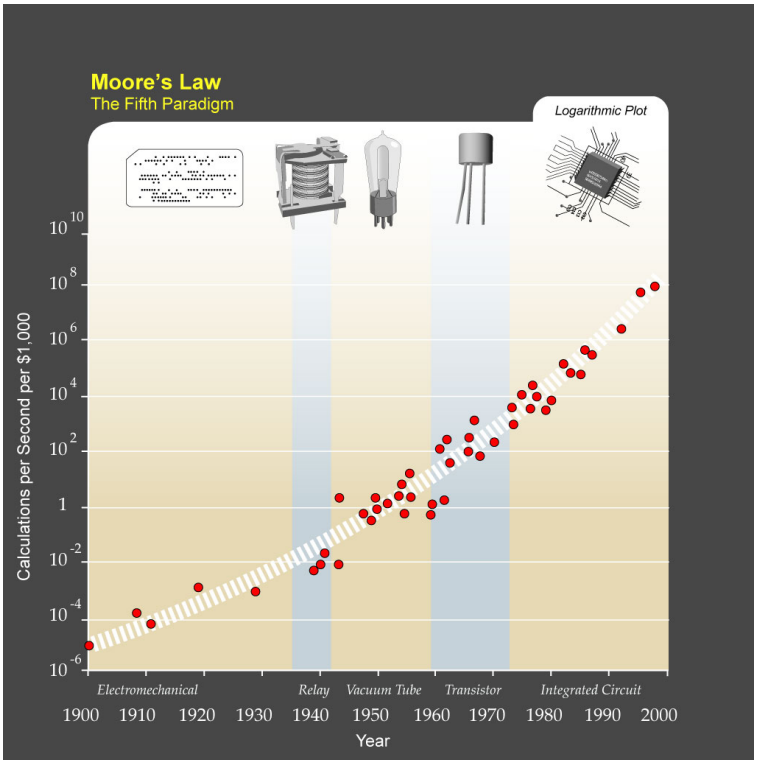


Figure 3: Moore's Law (Image credit: Von Courtesy of Ray Kurzweil and Kurzweil Technologies, Inc. - en:Image:PPTMooresLawai.jpg, CC BY 1.0, https://commons.wikimedia.org/w/index.php?curid=1273707)

- 22nm POWER8 with 96MB cache

Providing a fast memory with short access times is essential, otherwise a fast processor becomes completely useless when it cannot write the output to the cache.

**Interconnect bandwidth** When it comes down to fast connection between processing elements in a system, the connection between the chips is a bottleneck and can confine the performance. By replacing a conventional PCIe with NVLink makes the connection up to 12x faster. In essence, a homogeneous system may contain multiple cores, but is still only using one type of core.

## 1.3 Limits Of Homogeneous Systems

For over four decades, Moore's Law (Figure 3) suggested a contiguous accretion of computing performance. Indeed the transistor density doubled every 18months. However this did not only result in a tremendous improvement of performance, but also in some serious problems. One of these technical issues is the so called power-wall (Figure 4). This describes the phenomenon of enormous heat produced by the cores leading to an immense power density on the chip with the result of destruction of the whole system. The powerwall is also an indicator how important energy efficient computing is in future. When we reach out to the limits of one chip one might ask if we could optimize our system by using several cores for the same task or perform the same task on multiple cores simultaneously. This idea of parallelism is very useful when it comes to low energy consumption as a goal for your processing unit because the powerdensity is related to the cube of the clockrate:

$$P_d \propto f_{clock}^3$$

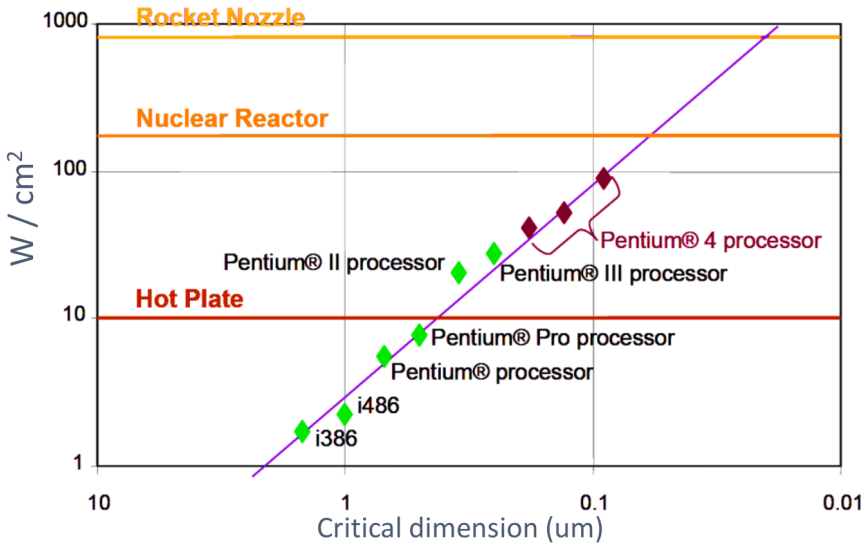


Figure 4: Power Wall (Image credit: G. Taylor, "Energy Efficient Circuit Design and the Future of Power Delivery")

As an example say we are looking at two different systems with one or two cores. We will be able to get the most performance out of the two core system while setting a fixed amount of energy:

- Single-core: 100 % power, 100 % frequency, 100 % performance
- Dual-core: 100 % power, 85 % frequency, 90 % performance each

Nevertheless parallelism is not the solution when you want to achieve the maximum performance. There is another interesting proportionality, called Amdahl's Law which links the speedup to the level of parallelism:

$$S = \frac{T_0}{\frac{T_p}{l} + T_S} = \frac{1}{\frac{T_p}{T_0 \cdot l} + \frac{T_s}{T_0}} = \frac{1}{\frac{\beta}{l} + (1 - \beta)}$$

Where $l$ is the level of parallelism and $\beta$ refers to the fraction of the computationtime influenced by the parallelism ($T_p/T_0$) (Figure 5). It is obvious when looking at Figure 6 that parallelism is only able to provide some speed up, but is far from beeing the answer to reach the exa-scale. Even a 100% parallel portion could not result in the needed speed up.

Other, compared to the described problems, minor issues are physical problems with transistor density e.g. wire-delays due to more transistors and the intrinsic problem of the bus-system of scalar processors: the Von-Neumann-Bottleneck.

Homogeneous systems are easy to program, but the homogenenous approach will fail at some point.... Adding another good point to the challanges of homogeneous exascale-computing:

> "Also important is the likely degradation of reliability due to the multiplicative factors of MTBF[1] and scale of components." [Zim03]

> "While reliability is a major concern for Exascale, another key-challenge is to minimize energy consumption, both for economic and environmental reasons." (Estimated threshold ≈ 20MW) [HR15]

[1] Mean Time Between Failures

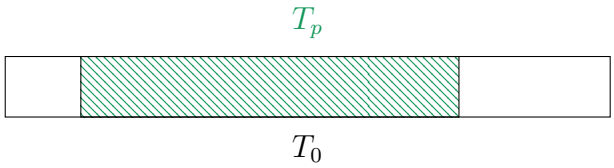


Figure 5: Fraction of the computationtime influenced by the parallelism as used in Amdahl's Law

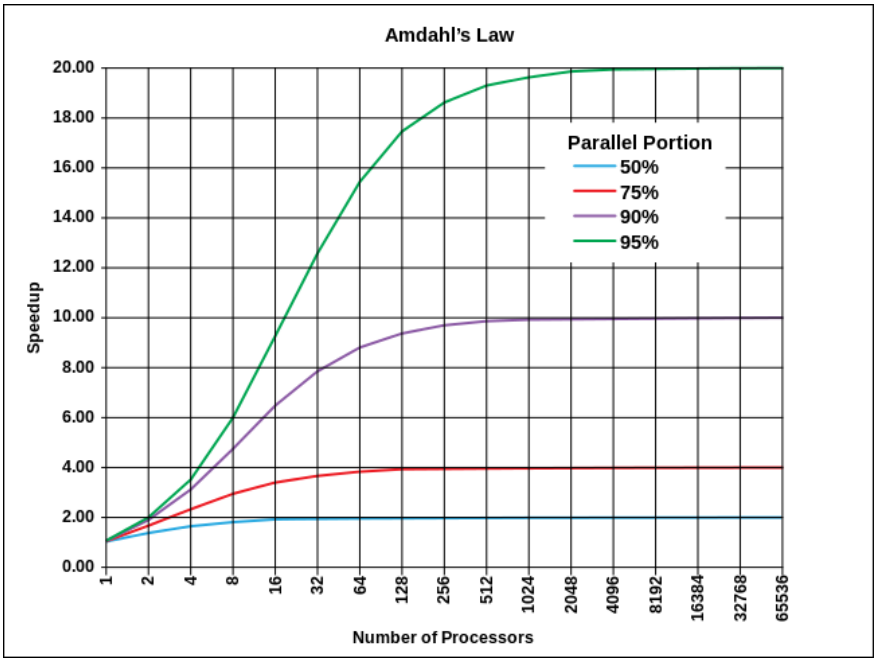


Figure 6: Amdahl's Law (Image credit: By Daniels220 at English Wikipedia - Own work based on: File:AmdahlsLaw.png, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6678551

# 2 Heterogeneous Systems

## 2.1 Motivation For Heterogeneous Computing (HC)

As pointed out homogeneous systems have some critical weeknesses throughout their construction. The CPU is straight forward for latency-critical applications because of it's fast serial procession. In contrast the GPUs massive parallelism using multiple cores guarantees good performance for throughput-critical applications.

Although this sounds like we have just a good device if we combine these to processing units we have to keep in mind that the serial allocation of tasks and work division between these two processing units will cause either the CPU to stay idle while the GPU works, or to let the GPU stay idle while the CPU works. This results in a immense waste of energy. What is needed at this point would be a work-division at runtime in means of allocating tasks or furthermore splitting tasks and parts of algorithms between the two processing units to achieve the best performance with low energy consumption.

For our example using a GPU and a CPU, the GPU could process long lists and the CPU could work on short lists instead at the same time. So called heterogeneous systems try to do exactly this. A practical hint that this idea works out well is given by the Green500, a list of environmental friendly super computers. The top 15 systems in the Green500 are heterogeneous systems, according to [VM15].

## 2.2 Mixed-Systems

The term heterogeneous needs some further explanation and definition. Let a system be called heterogeneous if it is not homogeneous. To go into some more detail, a system can be classified as heterogeneous if it contains different types of cores working together as one processing unit, mostly designed for a specific application. A task is then split upon the different cores of the processing unit.

To use our analogy of the bakery again, we can illustrate the heterogeneous system with different bakers working in the bakery. One is producing the dough, one is pugging the dough, another one forms the cupcakes and the last one puts them into the oven. Every single baker has it's own task where he performs best. Only together they are able to run the bakery. This is how heterogeneous computing works: The hardware is designed for a certain application and to process a specific task, dividing the sub-tasks onto different parts of the processing unit.

There are different models for heterogeneous computing. From their outer appearance we differ between fused or integrated systems and discrete systems.

**Fused/integrated systems** These systems only use one die containing all needd processing units. If we combine a CPU and a GPU on one chip together with an APU[2], we then would call this an integrated/fused system.

**Discrete systems** In comparison to the fused systems we can clearly distinguish discrete systems with their multiple die architecture. Assuming a discrete CPU-GPU system, a CPU-GPU connection via PCIe[3] bus would be necessary to make this a discrete heterogeneous system.

[2] accelerated processing unit

[3] Peripheral Component Interconnect Express

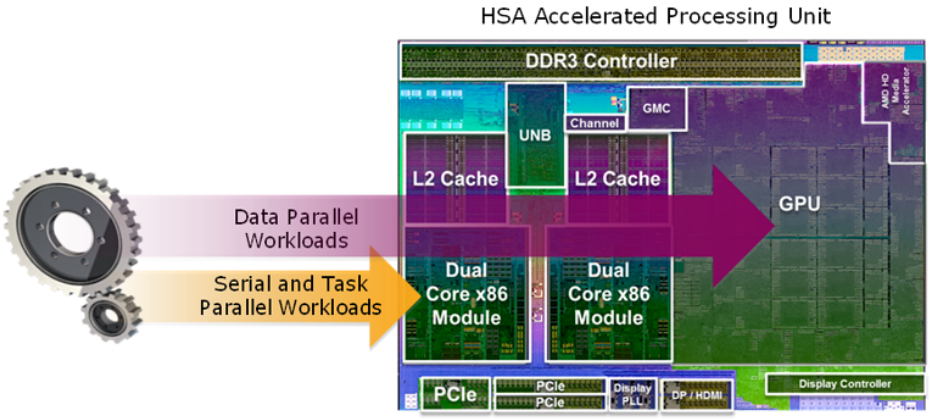


Figure 7: Heterogeneous system architecture accelerated processing unit (HSA APU). (Image credit: AMD, http://amd- dev.wpengine.netdna- cdn.com/wordpress/media/2012/10/HSAAcceleratedProcessingUnit.png

## 2.3 The Components

Commonly known as processing units we want to examplify some heterogeneous architectures by combining a CPU and a GPU. CPU-GPU-combination use the assets of a scalar *and* a vector processor for small data sets/certain algorithms *and* large data sets/numerical computation. A possible approach was made by AMD (Figure 7). Two dual core processors combined with a GPU to a fused system is able to handle a variety of intensive computations. Other attempts were carried out by Nvidia and Intel. The Nvidia Tesla combined CPU and GPU in an interesting way, referred to as GPGPU. The CPU was used as a host while the GPUs should perform the computation.

Vice versa the Intel Larrabee is a graphics processing unit unsing only CPUs. A massive parallelization of cores and multiple CPUs were used to process data obtained from audio and video data-files.

Both, the Intel Larrabee and the Nvidia Tesla are some kind of heterogeneous system, but they do not use the strengths of different core types as well as reasonable work-division for an accelerated computation. Instead they tried to get the most performance out of one of the components. Nevertheless this was an important step for CPU and GPU development.

## 2.4 Challenges Of Heterogeneous Computing

Promising extraordinary performance and environmental friendly computing, heterogeneous systems seem to outdate the homogeneous system at some point in the future. Beside all this great things there is of course a price to pay when working with heterogeneous systems [VM15].

As an intrinsic characteristic heterogeneous systems use at least two different types of cores. This also includes that the used processing units have a different architecture (at least on the microarchitecture level), application specific performance and thus different programming models are available or at some point required.

Considering the heterogeneous system as one processing unit, other features have to be taken into account. The system can be discrete or fused. The present cores may need different energy supply, they may have different computation power as well and cannot communicate at the same bandwidth. Another important point is that, this is the concept of a heterogeneous system, every processing units has it's strengths and weeknesses, e.g. the GPU has a reduced performance when doing computations with double-precision. This is important when constructing a host which has to set up the work-division between the kernels. It does not make sense to create an equal work-division or even might be impossible.

When it comes to the construction of an heterogeneous system also objective-specific questions have to be clarified. The architecture has to be designed for either energy efficient computing, performance or fairness or a mixture of all these three concepts.

Last but not least the system should be made for a particular application. This is why the nature of the desired algorithms, the subdivision and allocation of work or the dependencies to previous tasks has to be taken into account beforehand.

It is not only the hardware-design of heterogeneous systems which is much more complicated than that of homogeneous systems, but also the programming. Heterogeneous systems create new burdens for every programmer who wants to fully exploit the ressources of a given system.

The desired code should have low complexity paired with high portability and of course supporting debugging and performance tools. A possible approach to this was published by D.M. Kunzmann [KK11]. Introducing the C++ based programming language Charm++, he offers the opportunity to write a flexible code for different heterogeneous systems.

> Charm++ is based on C++. The unit of concurrency is a **share object** with member functions and additional **entry methods**, which are asynchronously invoked member functions. This allows the programmer to arbitrary create asynchronous tasks (e.g. broadcast, reductions, near neighbour interactions, data parallelism) [KK11]

The language Charm++ comes with three key-concepts:

**Accelerated entry methods** Charm++ provides accelerated entry methods which *may or may not* be executed on accellerator. If there is a specific kernel available for this task the host may decide if this task is allocated to another kernel and how to balance the work-division dynamically at runtime.

**Accelerated blocks** A new kind of location for shared functions is intruduced with accelerated blocks. These are blocks of code that exist at the *global scope*. They may be called by accellerated entry methods as well as normal entries. They serve as a location to declare shared functions and constants with the purpose to let functions call these blocks from accelerators and host cores.

**SIMD instruction abstraction** This part of the language maps operations from the API[4] to the hardware's SIMD[5] extensions. If there is no such hardware or system-specific instruction, the API defaults to a C++ implementation of the operations.

In general heterogeneous systems require hardware-specific code. Charm++ comes with the key concept of hardware dependent code interpretation of "flexible" written Charm++-code.

## 2.5 State Of The Art

Heterogeneous system architecture is already more than a concept, it has been used for a variety of computers, including some of the most powerful machines today. The concept of heterogeneous architecture is examplified in the following by describing two supercomputers: *XD1* and *Titan*, both manufactured by Cray.

[4]Application Programming Interface
[5]Single Instruction, Multiple Data


Figure 8: The XD1 supercomputer. (Image credit: Cray 2005)

**XD1** The supercomputer XD1 [Sha06] was ranked # 161 of Top500 in November 2005. It is a flexible system with linkable racks (Figure 8). Thus it's computation power can be adjusted to your needs.

The hardware of the XD1 can meet your requirements at theoretical peak of 3.633 TFLOPS. The model started at 100.000$. One of it's most interesting features is the FPGA[6] coprocessor-model. This results in hardware reconfigurable heterogeneous HPC[7].

The board of the XD1 is illustrated in Figure 9. The serial tasks are process by two AMD Opteron 64-bit processors to deliver 58 GFLOPs per chassis. A specialized Linux runs the system and the RapidArray$^{TM}$ Interconnect with a speed of 96GB/s guarantees fast communication. The active management is responsible for the runtime observation of the computer's health. The heart of this system is the AAS[8] built from Six Xilinx Virtex-II Pro$^{TM}$ FPGAs[9].

> The application acceleration subsystem incorporates reconfigurable computing capabilities to deliver superlinear speedup of targeted applications.
> [...]
> Well suited to functions such as all-reduce operations, searching, sorting and signal processing, the application acceleration subsystem acts as a coprocessor to the AMD Opterons, handling the computationally intensive and highly repetitive algorithms that can be significantly accelerated through parallel execution.
> (Credit: Cray XD1 Datasheet)

**Titan** The recently launched supercomputer Titan was the first hybrid to perform over 10 petaFLOPS with a CPU-GPU combination. It's theoretical peak was calculated to be 27 petaFLOPS at a power consumption of 8.2 MW. Titan was also designed for six designated problems in computation (Figure 10).

# 3 Conclusion

## 3.1 Summary And Outlook

From what was said before it becomes obvious that homogenous systems are outreached by heterogeneous systems. With it's application specific hardware design these non-homogeneous system can

[6] Field Programmable Gate Array
[7] high-performance-computing
[8] Application Accelleration System
[9] Field Programmable Gate Arrays

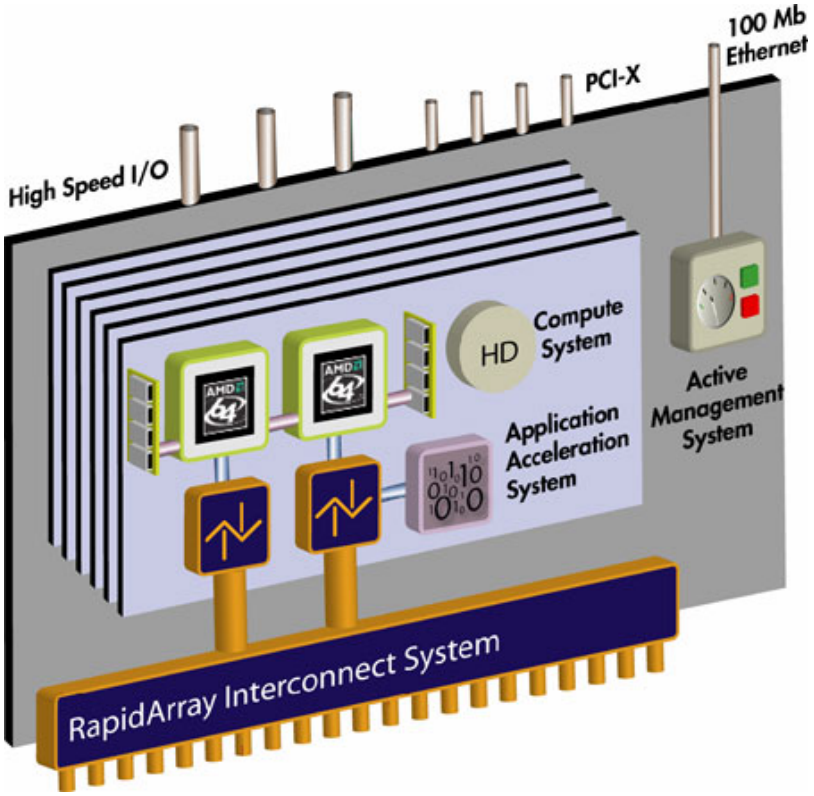


Figure 9: The XD1 supercomputer. (Image credit: Cray)

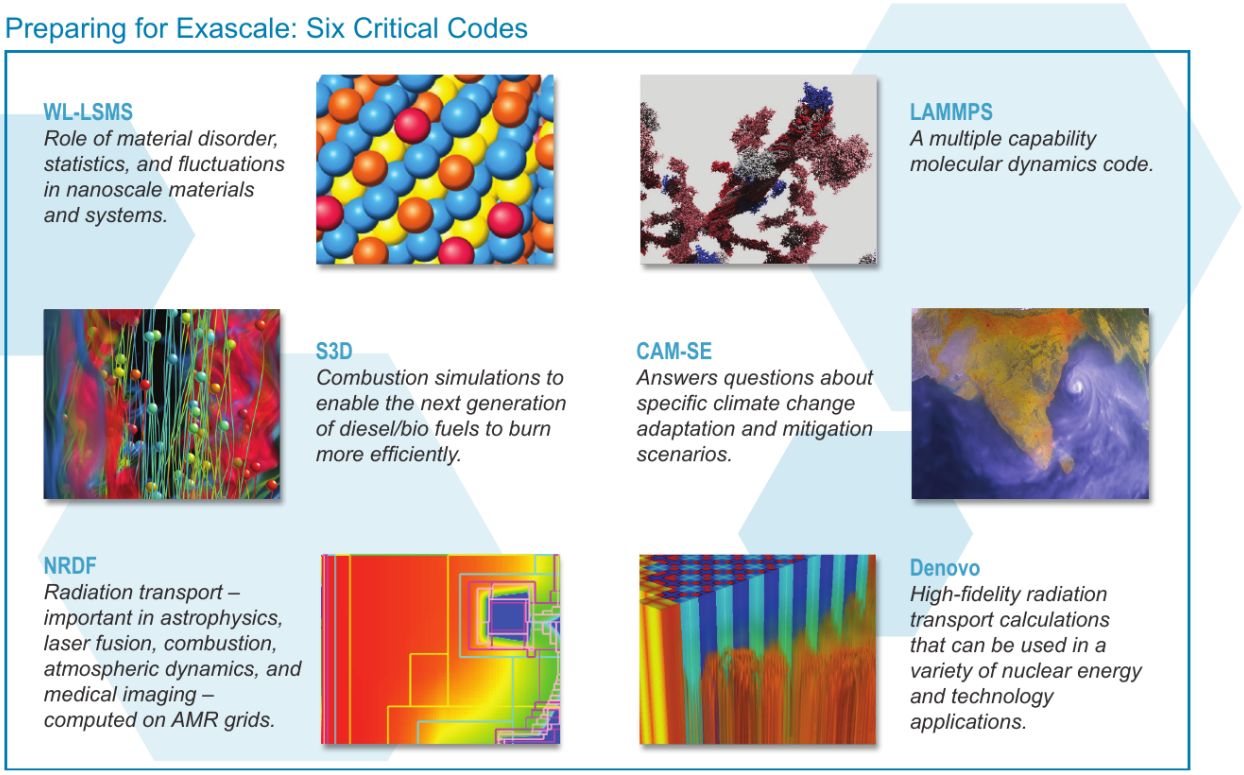


Figure 10: Titan's tasks. (Image credit: Oak Ridge Leadership Computung Facility)


Figure 11: The less positive trend of performance since 2014. (Image credit: Nicole Hemsoth)

deliver higher performance for the desired field. They can also be more energy efficient because of a better work-division between the kernels. To accompany all these positive aspects high pressure is transfered to the programmer who has to work out the right balance and write flexible code for the machines.

China planned the first exa-scale computer for 2020, when their Tianhe-3 shall start operating. Beside all these positive and promising concepts and achievements, the fact that the performance did not increase reasonably since 2014 (Figure 11) lets computer-experts be careful with two positive expectations for the future.

Nevertheless, HSA holds tremendous potential to open the era of exascale-supercomputing.

# References


[Bla96] L. S. Blackford, A. Cleary, J. Demmel, I. Dhillon, J. Dongarragif, S. Hammarlinggif, A. Petitet, H. Ren, K. Stanley, and Whaley R. C. "Practical Experience in the Dangers of Heterogeneous Computing". German. In*LAPACK Working Note 112*, vol. (1996). URL: http://www.netlib.org/utk/papers/practical-hetro/paper.html (cit. on p. 3).

[Chi14] M. Chiesi. "Heterogeneous Multi-core Architectures for High Performance Computing". PhD thesis. University of Bologna, 2014.

[HP12] J. L. Hennessy and D. A. Patterson. *Computer Architecture - A Quantitative Approach*. Ed. by T. Green. Elsevier Inc., 2012.

[HR15] T. Herault and Y. Robert. *Fault-Tolerance Techniques for High-Performance Computing*. Ed. by T. Herault and Y. Robert. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-20943-2 (cit. on p. 6).

[KK11] D. M. Kunzmann and L. V. Kale. "Programming Heterogeneous Systems". In*IEEE International Parallel and Distributed Processing Symposium*, vol. (2011) (cit. on p. 10).

[KM08] S. Kaxiras and M. Martonosi. *Computer Architecture techniques for Power-Efficiency*. Ed. by M. D. Hill. Morgan & Claypool, 2008. DOI: 10.2200/S00119ED1V01Y200805CAC004.

[PH14] D. A. Patterson and J. L. Hennessy. *Computer organization and Design*. Ed. by T. Green. Elsevier Inc., 2014.

[Rol09] Sascha Roloff. *Multicore-Architekturen*. German. University of Erlangen. Sept. 2009. URL: http://www2.in.tum.de/hp/file?fid=311.

[Sha06] A. Shan. "Heterogeneous Processing: a Strategy for Augmenting Moore's Law". In*Linux Journal*, vol. (2006) (cit. on p. 11).

[VM15] J. Vetter and S. Mittal. "A survey of CPU-GPU heterogeneous computing techniques". In*ACM Computing Surveys*, vol. (2015) (cit. on pp. 8, 9).

[Zim03] H.P. Zima, K. Joe, M. Sato, Y. Seo, and M. Shimasaki. *High Performance Computing: 4th International Symposium, ISHPC 2002, Kansai Science City, Japan, May 15-17, 2002. Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003. ISBN: 9783540478478. URL: https://books.google.de/books?id=3QVrCQAAQBAJ (cit. on p. 6).