Overview of recent supercomputers

Aad J. van der Steen
EuroBen Foundation,
SURFSara
The Netherlands
ajsteen1@euroben.nl
www.euroben.nl

Abstract

In this report we give an overview of high-performance computers which are currently available or will become available within a short time frame from vendors; no attempt is made to list all machines that are still in the development phase. The machines are described according to their macro-architectural class. The information about the machines is kept as compact as possible. No attempt is made to quote price information as this is often even more elusive than the performance of a system. In addition, some general information about high-performance computer architectures and the various processors and communication networks employed in these systems is given in order to better appreciate the systems information given in this report.

This document reflects the technical momentary state of the supercomputer arena as accurately as possible. However, the author cannot take any responsibility for errors or mistakes in this document. We encourage anyone who has comments or remarks on the contents to inform us, so we can improve this report.

Contents

1 Introduction and account

This is the 23nd edition of a report in which we attempt to give an overview of high-performance computer systems that are commercially available or are expected to become available within a short time frame (typically a few months to half a year). We choose the expression "attempt" deliberately because the market of high-performance machines is highly volatile: the rate at which systems are introduced — and disappear again — is high and therefore the information may be only approximately valid. Nevertheless, we think that such an overview is useful for those who want to obtain a general idea about the various means by which these systems strive at high-performance, especially when it is updated on a regular basis.

We will try to be as up-to-date and compact as possible and on these grounds we think there is a place for this report. Generally speaking, integrated parallel systems have for the most part given way to clustered systems. Indeed, it becomes hard to distinguish between them these days as many features that set integrated systems apart from clusters appear in the latter systems and characteristics present in clusters are now present in what used to be called integrated parallel sytems. It has become all the more clear that no one processor type is best for all possible types of computation. So, a trend is emerging of diversifying processor types within a single system. A sign of this is the appearance of FPGAs, high-end graphical cards and other computational accelerators in systems with standard processors. We may expect that this trend will continue in the coming years making the high-performance computer landscape more diverse, interesting, and harder to exploit efficiently.

The majority of systems, however, still look like minor variations on the same theme: clusters of Symmetric Multi-Processing (SMP) nodes are connected by a fast network. Culler *et.al.* [7] consider this as a natural architectural evolution. However, it may also be argued that economic pressure has steered the systems in this direction, thus letting the High Performance Computing world take advantage of the mainstream technology developed for the mass market.

The supercomputer market is very dynamic and this is especially true for the cluster world that has emerged at a tremendous rate in the last 10 years. The number of vendors that sell pre-configured clusters has boomed accordingly and therefore we do *not* include such configurations in this report: the speed with which cluster companies and systems appear and disappear makes this almost impossible. We will, however, comment on cluster characteristics and their position relative to other supercomputers in section 2.7.
For the tightly-coupled or "integrated" parallel systems though, we can by updating this report at least follow the main trends in popular and emerging architectures. The details of the systems to be reported do not allow the report to be shorter than in former years: in the order of 70+ pages.

For some years already we include sections about important system components in order to better appreciate the systems described later: processors in section 2.8, networks in section 2.10, and computational accelerators in section 2.9. This information is sometimes rather sketchy because of scant vendor information but nevertheless on average it gives a fair idea of the components' properties.

The rule for including systems is as follows: they should be either available commercially at the time of appearance of this report, or within 6 months thereafter. This excludes some interesting systems at various places in the world (all with measured performances in the range of Pflop/s), the Tianhe-2 system (with a measured performance of around 33 Pflop/s), and some accelerator-enhanced clusters with speeds of several Pflop/s. The Tianhe-2 is a one-of-a-kind system, while the clusters mentioned are of various makings with standard components and as such not systems that are sold as ready-made products.
The rule that systems should be available within a time-span of 6 months is to avoid confusion by describing systems that are announced much too early just for marketing reasons and that will not be available to general users within a reasonable time. We also have to refrain from including all generations of a system that are still in use. Therefore, for instance, we do not include the earlier IBM SP or the Cray X{1,2} series

anymore although some of these systems might still be in use somewhere. To add to the information given in this report, we quote the Web addresses of the vendors because the information found there may be more recent than what can be provided here. On the other hand, such pages should be read with care because it will not always be clear what the status is of the products described there.
Some vendors offer systems that are identical in all respects except in the clock cycle of the nodes. In these cases we always only mention the models with the fastest clock as it will be always possible to get the slower systems and we presume that the reader is primarily interested in the highest possible speeds that can be reached with these systems.

The systems described are listed alphabetically. In the header of each system description the machine type is provided. There is referred to the architectural class for as far this is relevant. We omit price information which in most cases is next to useless. If available, we will give some information about performances of systems based on user experiences instead of only giving theoretical peak performances. Here we have adhered to the following policy: We try to quote *best measured performances*, if available, thus providing a more realistic upper bound than the theoretical peak performance. We hardly have to say that the speed range of supercomputers is enormous, so the best measured performance will not always reflect the performance of the reader's favorite application. In fact, when the HPC Linpack test is used to measure the speed it is fair to say that for the average user the application performance will be significantly lower. When we give performance information, it is not always possible to quote all sources and in any case if this information seems (or is) biased, this is entirely the responsibility of the author of this report. He is quite willing to be corrected or to receive additional information from anyone who is in the position to do so.

Although for the average user the appearance of new systems in the last years tended to become rapidly more and more alike, it is still useful to dwell a little on the architectural classes that underlie this appearance. It gives some insight in the various ways that high-performance is achieved and a feeling why machines perform as they do. This is done in section 2. It will be referred to repeatedly in section **??** in the description of the machines.
Up till the 10^{th} issue we included a section (4) on systems that disappeared from the market. We reduced that section in the printed and PostScript/PDF versions because it tends to take an unreasonable part of the total text. Still, because this information is of interest to some readers and it gives insight in the field of the historical development of supercomputing over the last 20 years, this information will still be available in full at URL `http://www.euroben.nl/web13/gone.html`. In section 5 we present some systems that are under development and have a fair chance to appear on the market. Because of the addition of the section on processors, networks, and accelerators that introduces many technical terms and abbrevations, also a glossary is included.

The overview given in this report concentrates on the computational capabilities of the systems discussed. To do full justice to all assets of present days high-performance computers one should list their I/O performance and their connectivity possibilities as well. However, the possible permutations of configurations even for one model of a certain system often are so large that they would multiply the volume of this report. So, not all features of the systems discussed will be present. We also omit systems that may be characterised as "high-performance" in the fields of database management, real-time computing, or visualisation, scientific and technical computing being our primary interest. Still we think (and certainly hope) that the impressions obtained from the entries of the individual machines may be useful to many. Furthermore, we have set a threshold of about 20 Tflop/s for systems to appear in this report as, at least with regard to theoretical peak performance, single CPU cores often exceed 5 Gflop/s although their actual performance may be an other matter entirely.

Although most terms will be familiar to many readers, we still think it is worthwhile to give some of the definitions in section 2 because some authors tend to give them a meaning that may slightly differ from the idea the reader already has acquired.

Lastly, we point out that there is also a web version. The URL is:
`http://www.euroben.nl/web13/overview.php`.

From the 16^{th} issue on we *attempt* to keep the web version up-to-date by refreshing the contents more frequently than once a year. So, the printed version may lag a little behind the web version over the year.

2 Architecture of high-performance computers

Before going on to the descriptions of the machines themselves, it is useful to consider some mechanisms that are or have been used to increase the performance. The hardware structure or *architecture* determines to a large extent what the possibilities and impossibilities are in speeding up a computer system beyond the performance of a single CPU core. Another important factor that is considered in combination with the hardware is the capability of compilers to generate efficient code to be executed on the given hardware platform. In many cases it is hard to distinguish between hardware and software influences and one has to be careful in the interpretation of results when ascribing certain effects to hardware or software peculiarities or both. In this chapter we will give most emphasis on the hardware architecture. For a description of machines that can be considered to be classified as "high-performance" one is referred to [7, 34].

2.1 The main architectural classes

Since many years the taxonomy of Flynn [11] has proven to be useful for the classification of high-performance computers. This classification is based on the way of manipulating of instruction and data streams and comprises four main architectural classes. We will first briefly sketch these classes and afterwards fill in some details when each of the classes is described separately.

- **SISD** machines: These are the conventional systems that use one CPU core for the execution of a program and hence can accommodate one instruction stream that is executed serially. Nowadays about all large servers have more than one multi-core CPU but each of these can execute instruction streams that are unrelated. Therefore, such systems still should be regarded as (a couple of) SISD machines acting on different data spaces. Examples of SISD machines are for instance workstations as offered by many vendors. The definition of SISD machines is given here for completeness' sake. We will not discuss this type of machines in this report.
- **SIMD** machines: Such systems often have a large number of processing units, that all may execute the same instruction on different data in lock-step. So, a single instruction manipulates many data items in parallel. Examples of SIMD machines in this class were the CPP Gamma II and the Quadrics Apemille which are not marketed anymore since several years. Nevertheless, the concept is still interesting and has recurred recently in co-processors in HPC systems, be it in a somewhat restricted form, like in GPUs.

  Another type of SIMD computation is vectorprocessing. Until a few years ago this used to be done in stand-alone vectorprocessing systems as for instance produced by Cray and NEC. Presently, the concept is mostly implemented in the vector units of common chips. Examples are the SSE and AVX units in AMD and Intel chips.
- **MISD** machines: Theoretically in these types of machines multiple instructions should act on a single stream of data. As yet no practical machine in this class has been constructed nor are such systems easy to conceive. We will disregard them in the following discussions.
- **MIMD** machines: These machines execute several instruction streams in parallel on different data. The difference with the multi-processor SIMD machines mentioned above lies in the fact that the instructions and data are related because they represent different parts of the same task to be executed. So, MIMD systems may run many sub-tasks in parallel in order to shorten the time-to-solution of the main task. There is a large variety of MIMD systems and especially in this class the Flynn taxonomy proves to be not fully adequate for the classification of systems. Systems that behave very differently like a 4-processor NEC SX-9 vector system and a 100,000-processor IBM BlueGene/Q fall both in this class. In the following we will make another important distinction between classes of systems and treat them accordingly.

- **Shared-memory systems**: Shared-memory systems have multiple CPUs all of which share the same address space. This means that the knowledge of where data is stored is of no concern to the user as there is only one memory accessed by all CPUs on an equal basis. Shared memory systems can be both SIMD or MIMD. We will sometimes use the abbreviations SM-SIMD and SM-MIMD for the two subclasses.
- **Distributed-memory systems**: In this case each CPU has its own associated memory. The CPUs are connected by some network and may exchange data between their respective memories when required by that network. In contrast to shared-memory machines the user must be aware of the location of the data in the local memories and will have to move or distribute these data explicitly when needed. Again, distributed-memory systems may be either SIMD or MIMD. The first class of SIMD systems mentioned which operate in lock step, all have distributed memories associated to the processors. As we will see, distributed-memory MIMD systems exhibit a large variety in the topology of their interconnection network. The details of this topology are largely hidden from the user which is quite helpful with respect to portability of applications but that at the same time may have an impact on the performance. For the distributed-memory systems we will sometimes use DM-SIMD and DM-MIMD to indicate the two subclasses.

As already alluded to, although the difference between shared and distributed-memory machines seems clear cut, this is not always entirely the case from user's point of view. For instance, the late Kendall Square Research systems employed the idea of "virtual shared-memory" on a hardware level. Virtual shared-memory can also be simulated at the programming level: A specification of High Performance Fortran (HPF) was published in 1993 [19] which by means of compiler directives distributes the data over the available processors. Therefore, the system on which HPF is implemented in this case will look like a shared-memory machine to the user. Other vendors of Massively Parallel Processing systems (sometimes called MPP systems), like SGI, also support proprietary virtual shared-memory programming models due to the fact that these physically distributed memory systems are able to address the whole collective address space. So, for the user such systems have one *global address space* spanning all of the memory in the system. We will say a little more about the structure of such systems in section 2.6. In addition, packages like TreadMarks ([2]) provide a "distributed shared-memory" environment for networks of workstations. A good overview of such systems is given at [9]. Since 2006 Intel markets its "Cluster OpenMP" (based on TreadMarks) as a commercial product. It allows the use of the shared-memory OpenMP parallel model [29] to be used on distributed-memory clusters. Since a few years also companies like ScaleMP and 3Leaf provide products to aggregate physical distributed memory into virtual shared memory. In case of ScaleMP with a small amount of assisting hardware.
Lastly, so-called Partitioned Global Address Space (PGAS) languages like Co-Array Fortran (CAF) and Unified Parallel C (UPC) are gaining in popularity due to the recently emerging multi-core processors. With proper implementation this allows a global view of the data and one has language facilities that make it possible to specify processing of data associated with a (set of) processor(s) without the need for explicitly moving the data around.

Distributed processing takes the DM-MIMD concept one step further: instead of many integrated processors in one or several boxes, workstations, mainframes, etc., are connected by 1 or 10 GB/s Ethernet, or other, faster networks and set to work concurrently on tasks in the same program. Conceptually, this is not different from DM-MIMD computing, but the communication between processors can be much slower. Packages that initially were made to realise distributed computing like PVM (standing for Parallel Virtual Machine) [12], and MPI (Message Passing Interface, [24, 25]) have become *de facto* standards for the "message passing" programming model. MPI and PVM have become so widely accepted that they have been adopted by all vendors of distributed-memory MIMD systems and even on shared-memory MIMD systems for compatibility reasons. In addition, there is a tendency to cluster shared-memory systems by a fast communication network to obtain systems with a very high computational power. E.g., the NEC SX-9 has this structure. So, within the clustered nodes a shared-memory programming style can be used while between clusters message-passing should be used. It must be said that PVM is not used very much anymore and the development has stopped. MPI has now more or less become the *de facto* standard.
For SM-MIMD systems we mention OpenMP [29, 5, 6], that can be used to parallelise Fortran and C(++)

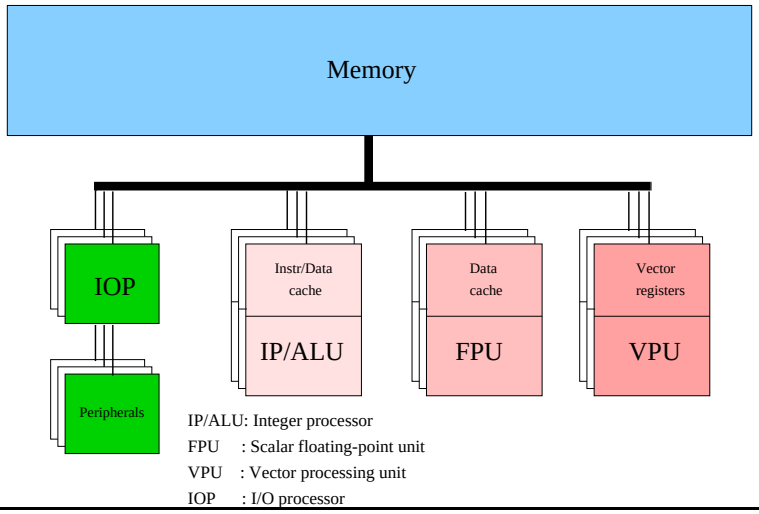


Figure 2.1: *Blockdiagram of a vector processor.*

programs by inserting comment directives (Fortran 77/90/95) or pragmas (C/C++) into the code. OpenMP has quickly been adopted by the all major vendors and has become a well established standard for shared memory systems.

Note, however, that for both MPI-3 and OpenMP 3, the latest standards, many systems/compilers only implement a part of these standards. One has therefore to inquire carefully whether a particular system has the full functionality of these standards available. The standard vendor documentation will almost never be clear on this point.

2.2 Shared-memory SIMD machines

This subclass of machines is practically equivalent to the single-processor vectorprocessors, although other interesting machines in this subclass have existed (viz. VLIW machines [33]) and may emerge again in the near future. In the block diagram in Figure 2.1 we depict a generic model of a vector architecture. The single-processor vector machine will have only one of the vectorprocessors shown here and the system may even have its scalar floating-point capability shared with the vector processor (as was the case in some Cray systems). It may be noted that the VPU does not show a cache. Vectorprocessors may have a cache but in many cases the vector unit cannot take advantage of it and execution speed may in some cases even be unfavourably affected because of frequent cache overflow. Of late, however, this tendency is reversed because of the increasing gap in speed between the memory and the processors: the Cray X2 had a cache and NEC's SX-9 vector system has a facility that is somewhat like a cache.

Although vectorprocessors have existed that loaded their operands directly from memory and stored the results again immediately in memory (CDC Cyber 205, ETA-10), present-day vectorprocessors use vector registers. This does not impair the speed of operations while providing much more flexibility in gathering operands and manipulation with intermediate results.

Because of the generic nature of Figure 2.1 no details of the interconnection between the VPU and the memory are shown. Still, these details are very important for the effective speed of a vector operation: when the bandwidth between memory and the VPU is too small it is not possible to take full advantage of the VPU because it has to wait for operands and/or has to wait before it can store results. When the ratio of arithmetic to load/store operations is not high enough to compensate for such situations, severe performance losses may be incurred. The influence of the number of load/store paths for the dyadic vector operation $c = a + b$ (a, b, and c vectors) is depicted in Figure 2.2. Because of the high costs of implementing these data paths between memory and the VPU, often compromises are sought and the full required bandwidth (i.e., two load operations and one store operation at the *same* time). Only Cray Inc. in its former Y-MP,

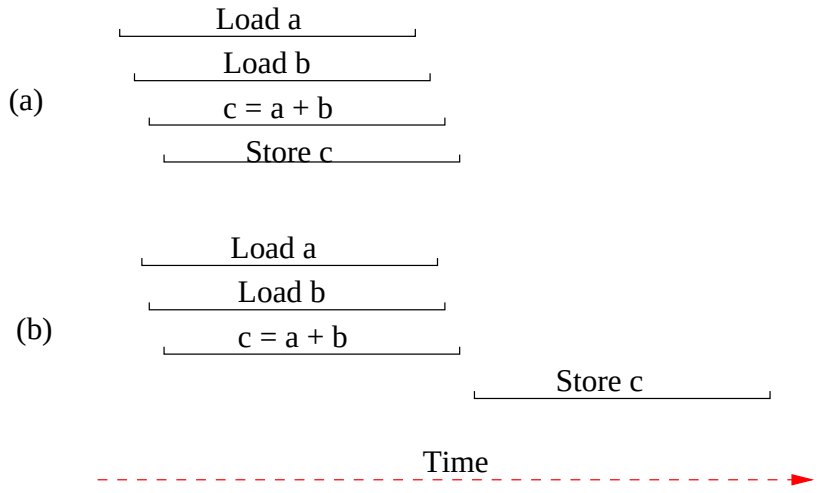


Figure 2.2: *Schematic diagram of a vector addition. Case (a) when two load- and one store pipe are available; case (b) when two load/store pipes are available.*

C-series, and T-series employed this very high bandwidth. Vendors now rather rely on additional caches and other tricks to hide the lack of bandwidth.

The VPUs are shown as a single block in Figure 2.1. Yet, there is a considerable diversity in the structure of VPUs. Every VPU consists of a number of vector functional units, or "pipes" that fulfill one or several functions in the VPU. Every VPU will have pipes that are designated to perform memory access functions, thus assuring the timely delivery of operands to the arithmetic pipes and of storing the results in memory again. Usually there will be several arithmetic functional units for integer/logical arithmetic, for floating-point addition, for multiplication and sometimes a combination of both, a so-called compound operation. Division is performed by an iterative procedure, table look-up, or a combination of both using the add and multiply pipe. In addition, there will almost always be a mask pipe to enable operation on a selected subset of elements in a vector of operands. Lastly, such sets of vector pipes can be replicated within one VPU (2 up to 16-fold replication occurs). Ideally, this will increase the performance per VPU by the same factor provided the bandwidth to memory is adequate.

Lastly, it must be remarked that vectorprocessors as described here are hardly considered a viable economic option anymore and only NEC still produces such stand-alone vectorprocessing systems. Instead, in a small way, vectorprocessing is now integrated into the common CPU cores. Like in vectorprocessors the principal idea of vector processing is that the same operation is being performed on multiple elements of arrays at the same time. For instance, when one wants to add the elements of two arrays $c = a + b$ several elements of these arrays can be handled at the same time. A vector instruction causes that the elements are loaded into the vector registers and are all added simultaneously. This has not only the effect of speeding up the computation it also leads to less instructions to be decoded as one instruction generates multiple results. Depending on the type of CPU 2–8 elemental results can be generated per clock cycle (a clock cycle being the fundamental time unit within the CPU). In Figure 2.3 the addition of 8 elements of arrays a and b is depicted.

Note that there is a difference in the processing of the vector elements in vectorprocessors and within the vector units of the common CPU cores. While the former produces a result every clock cycle, the latter yields several results in a single clock cycle.

Various additional units had to be added to CPUs to accomodate vector processing capabilities and corresponding vector instructions had to be added to the compilers to make use of them. Originally these vector units and instructions were meant to speed up graphical processes for multimedia applications but it was recognised soon that it could benefit computational work. For the x86 line of processors the vector units has for instance been the SSE2, SSE3, and SSE4 units. In the most recent Intel and AMD CPUs AVX units provide 8 32-bit precision floating-point results or 4 64-bit precision results, while in the IBM BlueGene/Q

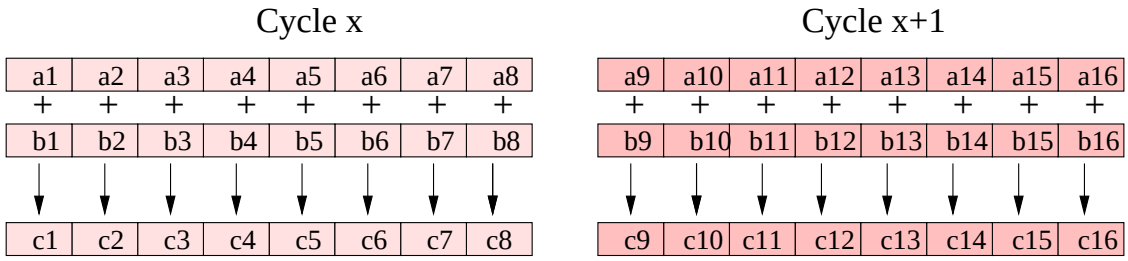


Figure 2.3: *Diagram of a vector addition.*

(see section 3.1.8) the AltiVec unit can produce 4 64-bit results simultaneously.

Presently, the vector instructions supported by the compilers are not yet as powerful as those of the late stand-alone vectorprocessors used to be. For instance, chaining vector operations, i.e., feeding the result from one vector operation directly to another without the need to store the intermediate result is still limited. The capabilities of the vector units are, however, constantly improving and one can expect them to become on par with the vectorising compilers for the stand-alone systems from the former Cray and Fujitsu systems (see 4) and the NEC compiler within a few years.

In a fair amount of cases the compiler can detect whether a part of a program can be vectorised. But in many cases the compiler has not enough information to decide whether it is safe to vectorise a particular part of the code. When it is known that this is the case one can give a hint to the compiler that it is safe to vectorise the code by means of so-called directives for Fortran and, equivalently, pragmas for C an C++ codes. Also libraries may hold vectorised versions of intrinsic functions or (numerical) algorithms. The vectorisation technique only works within a shared-memory environment as the compiler only can address memory that is in the local memory space.

2.3 Distributed-memory SIMD machines

Machines of the DM-SIMD type are sometimes also known as *processor-array* machines [15]. Because the processors of these machines operate in lock-step, i.e., all processors execute the same instruction at the same time (but on different data items), no synchronisation between processors is required. This greatly simplifies the design of such systems. A *control processor* issues the instructions that are to be executed by the processors in the processor array. Presently, no commercially available machines of the processor-array type are marketed. However, because of the shrinking size of devices on a chip it may be worthwhile to locate a simple processor with its network components on a single chip thus making processor-array systems economically viable again. In fact, common Graphical Processing Units (GPUs) share many characteristics with processor array systems. This is the reason we still discuss this type of system.

DM-SIMD machines use a front-end processor to which they are connected by a data path to the control processor. Operations that cannot be executed by the processor array or by the control processor are offloaded to the front-end system. For instance, I/O may be through the front-end system, by the processor array machine itself or by both. Figure 2.4 shows a generic model of a DM-SIMD machine of which actual models will deviate to some degree. Figure 2.4 might suggest that all processors in such systems are connected in a 2-D grid and indeed, the interconnection topology of this type of machines always includes the 2-D grid. As opposing ends of each grid line are also always connected the topology is rather that of a torus. This is not the only interconnection scheme: They might also be connected in 3-D, diagonally, or in more complex structures.

It is possible to exclude processors in the array from executing an instruction on certain logical conditions, but this means that for the time of this instruction these processors are idle (a direct consequence of the SIMD-type operation) which immediately lowers the performance. Another factor that may adversely affect the speed occurs when data required by processor i resides in the memory of processor j — in fact, as this occurs for all processors at the same time, this effectively means that data will have to be permuted across

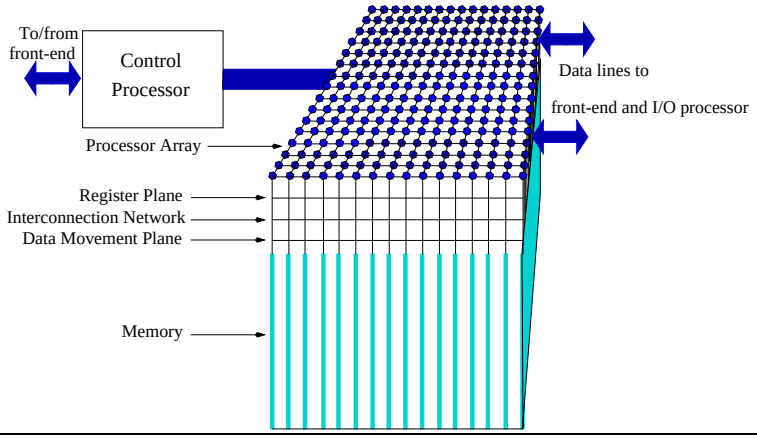


Figure 2.4: *A generic block diagram of a distributed-memory SIMD machine.*

the processors. To access the data in processor j, the data will have to be fetched by this processor and then send through the routing network to processor i. This may be fairly time consuming. For both reasons mentioned, DM-SIMD machines are rather specialised in their use when one wants to employ their full parallelism. Generally, they perform excellently on digital signal and image processing and on certain types of Monte Carlo simulations where virtually no data exchange between processors is required and exactly the same type of operations is done on massive data sets with a size that can be made to fit comfortable in these machines. They will also perform well on gene-matching type of applications.

The control processor as depicted in Figure 2.4 may be more or less intelligent. It issues the instruction sequence that will be executed by the processor array. In the worst case (that means a less autonomous control processor) when an instruction is not fit for execution on the processor array (e.g., a simple print instruction) it might be offloaded to the front-end processor which may be much slower than execution on the control processor. In case of a more autonomous control processor this can be avoided thus saving processing interrupts both on the front-end and the control processor. Most DM-SIMD systems had the possibility to handle I/O independently from the front-end processors. This is not only favourable because the communication between the front-end and back-end systems is avoided. A (specialised) I/O device for the processor-array system is generally much more efficient in providing the necessary data directly to the memory of the processor array. Especially for very data-intensive applications like radar and image processing such I/O systems are very important.

A feature that is peculiar to this type of machines is that the processors sometimes are of a very simple bit-serial type, i.e., the processors operate on the data items bit-wise, irrespective of their type. So, e.g., operations on integers are produced by software routines on these simple bit-serial processors which takes at least as many cycles as the operands are long. So, a 32-bit integer result will be produced two times faster than a 64-bit result. For floating-point operations a similar situation holds, be it that the number of cycles required is a multiple of that needed for an integer operation. As the number of processors in this type of systems is mostly large (1024 or larger, the Quadrics Apemille was a notable exception, however), the slower operation on floating-point numbers can be often compensated for by their number, while the cost per processor is quite low as compared to full floating-point processors. In some cases, however, floating-point co-processors were added to the processor-array. Their number was 8–16 times lower than that of the bit-serial processors because of the cost argument. An advantage of bit-serial processors is that they may operate on operands of any length. This is particularly advantageous for random number generation (which often boils down to logical manipulation of bits) and for signal processing because in both cases operands of only a few bits are abundant. Because, as mentioned, the execution time for bit-serial machines is proportional to the length of the operands, this may result in significant speedups.

Presently there are no DM-SIMD systems on the market but some types of computational accelerators (see 2.9) share many characteristics with DM-SIMD systems that have existed until shortly. We will briefly

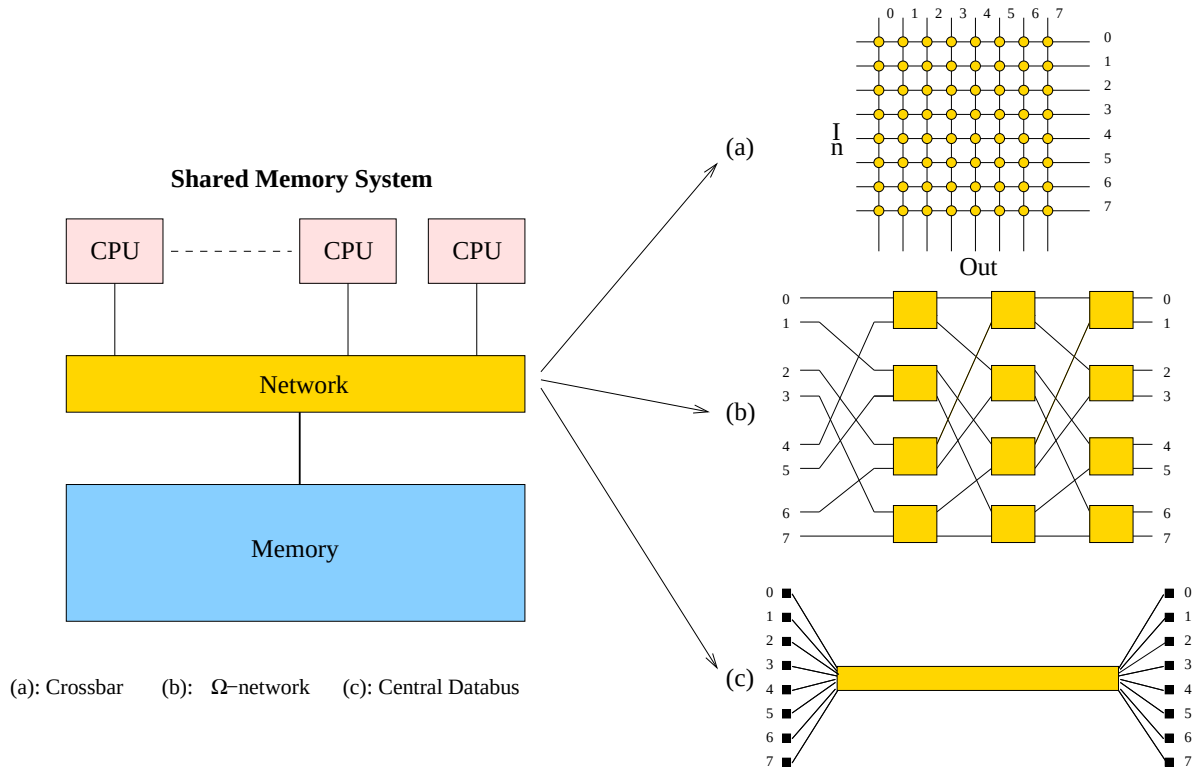


Figure 2.5: *Some examples of interconnection structures used in shared-memory MIMD systems.*

discuss some properties of these accelerators later.

2.4 Shared-memory MIMD machines

In Figure 2.1 already one subclass of this type of machines was shown. In fact, the single-processor vector machine discussed there was a special case of a more general type. The figure shows that more than one FPU and/or VPU may be possible in one system.

In shared-memory MIMD machines several processors access a common memory of which they draw their instructions and data. As such a system is used to perform a common task by executing parts of the programs in parallel there must be some way to coordinate and synchronise the various parts of the program. This is done by a network that connects the CPU cores to each other and to the memory.
In fact, as the present-day CPUs all harbour multiple cores (see, e.g., sections 2.8.1 and 2.8.4), a single multi-core CPU already can be regarded as an SM-SIMD system on a single chip. The following discussion therefore pertains to single CPUs as well although their interconnection is less of a problem.

When more CPUs are added, the collective bandwidth to the memory ideally should increase linearly with the number of processors, while each processor should preferably communicate directly with all others without the much slower alternative of having to use the memory in an intermediate stage. Unfortunately, full interconnection is quite costly, growing with $\mathcal{O}(n^2)$ while increasing the number of processors with $\mathcal{O}(n)$. So, various alternatives have been tried. Figure 2.5 shows some of the interconnection structures that are (and have been) used.

As can be seen from the Figure, a crossbar uses n^2 connections, an Ω-network uses $n \log_2 n$ connections while with the central bus there is only one connection. This is reflected in the use of each connection path for the different types of interconnections: for a crossbar each data path is direct and does not have to be shared with other elements. In case of the Ω-network there are $\log_2 n$ switching stages and as many data items may

have to compete for any path. For the central data bus all data have to share the same bus, so n data items may compete at any time.

The bus connection is the least expensive solution, but it has the obvious drawback that bus contention may occur, thus slowing down the computations. Various intricate strategies have been devised using caches associated with the CPUs to minimise the bus traffic. This leads however to a more complicated bus structure which raises the costs. In practice it has proved to be very hard to design buses that are fast enough, especially where the speed of the processors has been increasing very quickly and it imposes an upper bound on the number of processors thus connected that in practice never have exceeded a number of 10–20. In 1992, a new standard (IEEE P896) for a bus to connect either internal system components or to external systems has been defined. This bus, called the Scalable Coherent Interface (SCI) provides a point-to-point bandwidth of 200-1,000 MB/s. It has been used in the HP Exemplar systems, but also within a cluster of workstations as offered by SCALI. The SCI is much more than a simple bus and it can act as the hardware network framework for distributed computing, see [20]. It now has been effectively superseded by InfiniBand, however (see 2.10).

A multi-stage crossbar is a network with a logarithmic complexity and it has a structure which is situated somewhere in between a bus and a crossbar with respect to potential capacity and costs. The Ω-network as depicted in figure 2.5 is an example. Commercially available machines like the IBM eServer p775, the SGI Altix UV, and many others use(d) such a network structure, but a number of experimental machines also have used this or a similar kind of interconnection. The BBN TC2000 that acted as a virtual shared-memory MIMD system used an analogous type of network (a Butterfly-network) and it is likely that new machines will use it, especially as the number of processors grows. For a large number of processors the $n \log_2 n$ connections quickly become more attractive than the n^2 used in crossbars. Of course, the switches at the intermediate levels should be sufficiently fast to cope with the bandwidth required. Obviously, not only the *structure* but also the *width* of the links between the processors is important: a network using 16-bit parallel links will have a bandwidth which is 16 times higher than a network with the same topology implemented with serial links.

The earlier mentioned stand-alone multi-processor vectorprocessors used crossbars. This was feasible because the maximum number of processors within in a system node was small (16 at most). In the late Cray X2 the number of processors had so much increased, however, that it had to change to a logarithmic network topoly (see 2.10). Not only it becomes harder to build a crossbar of sufficient speed for the larger numbers of processors, the processors themselves generally also increase in speed individually, doubling the problems of making the speed of the crossbar match that of the bandwidth required by the processors.

Whichever network is used, the type of processors in principle could be arbitrary for any topology. In practice, however, bus structured machines cannot support vector processors as the speeds of these would grossly mismatch with any bus that could be constructed with reasonable costs. All available bus-oriented systems use RISC processors as far as they still exist. The local caches of the processors can sometimes alleviate the bandwidth problem if the data access can be satisfied by the caches thus avoiding references to the memory.

The systems discussed in this subsection are of the MIMD type and therefore different tasks may run on different processors simultaneously. In many cases synchronisation between tasks is required and again the interconnection structure is very important here. Some Cray vectorprocessors employed in the past special communication registers within the CPUs (the X-MP and Y-MP/C series) by which they could communicate directly with the other CPUs they had to synchronise with. This is, however, not practised anymore as it is viewed too costly a feature. The systems may also synchronise via the shared memory. Generally, this is much slower but it can still be acceptable when the synchronisation occurs relatively seldom. Of course, in bus-based systems communication also has to be done via a bus. This bus is mostly separated from the data bus to ensure a maximum speed for the synchronisation.

As said, a single CPU can be regarded as small SM-SIMD system in itself: up to sixteen processor cores are connected to each other by high-speed links. Because the number of cores is limited this is mostly done by a crossbar but also high-speed ring connections are employed.

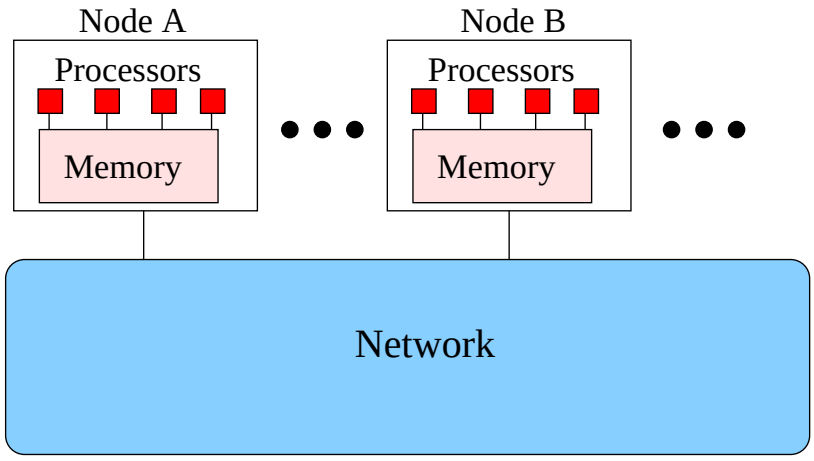


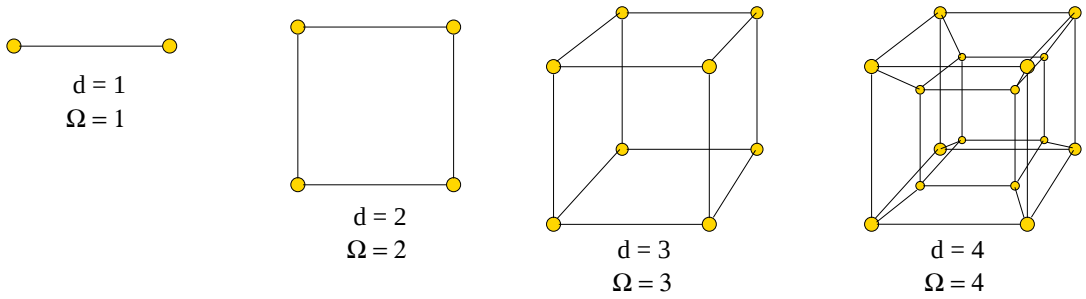
Figure 2.6: *Generic diagram of a DM-MIMD machine.*

2.5 Distributed-memory MIMD machines

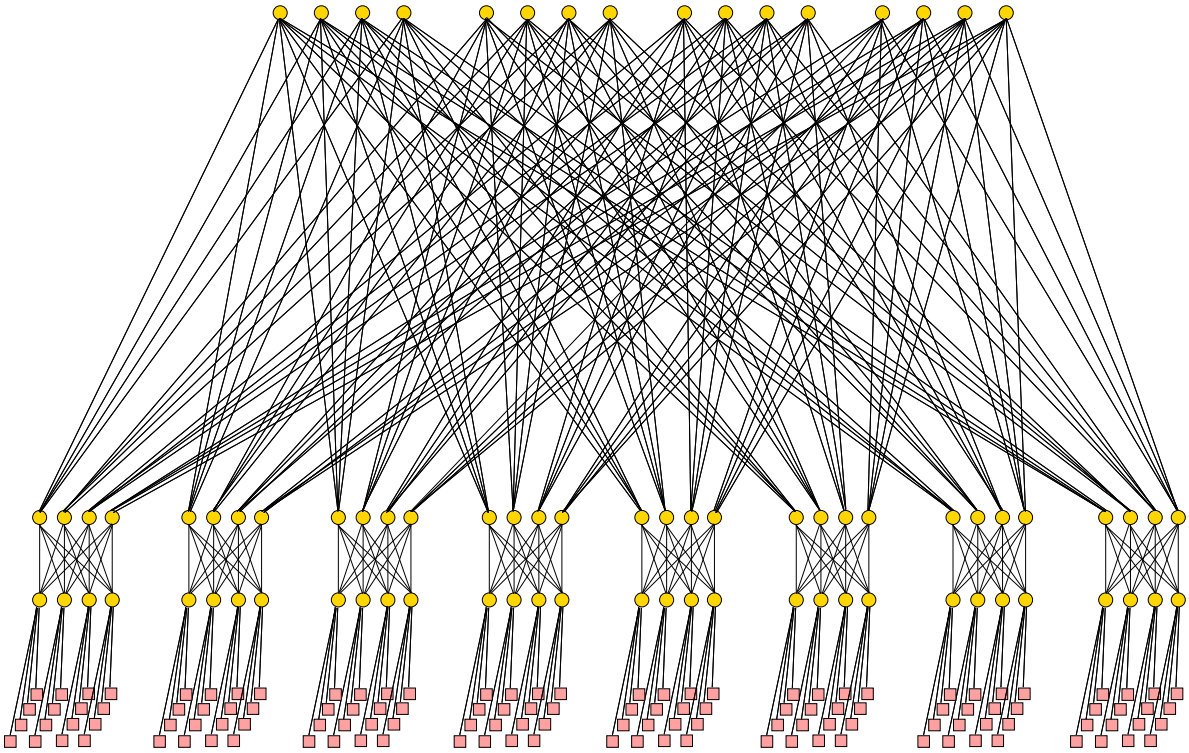
The class of DM-MIMD machines represents undoubtedly the largest fraction in the family of high-performance computers. A generic diagram is given in Figure 2.6. The figure shows that within a computational node A, B, etc., a number of processors (four in this case) draw on the same local memory and that the nodes are connected by some network. Consequently, when a processor in node A needs data present in node B this has to be accessed through the network. Hence the characterisation of the system as being of the distributed memory type. The vast majority of all HPC systems today are a variation of the model shown in Figure 2.6. This type of machines is more difficult to deal with than shared-memory machines and DM-SIMD machines. The latter type of machines are processor-array systems in which the data structures that are candidates for parallelisation are vectors and multi-dimensional arrays that are laid out automatically on the processor array by the system software. For shared-memory systems the data distribution is completely transparent to the user. This is generally quite different for DM-MIMD systems where the user has to distribute the data over the processors and also the data exchange between processors has to be performed explicitly when using the so-called message passing parallelisation model (which is the case in the vast majority of programs). The initial reluctance to use DM-MIMD machines seems to have been decreased. Partly this is due to the now existing standard for communication software ([12, 24, 25]) and partly because this class of systems has proven outperform all other types of machines.
Alternatively, instead of message passing a Partitioned Global Address Space parallelisation model may be used with a programming language like UPC [41] or Co-Array Fortran [4]. In this case one still has to be aware where the relevant data is located but no explicit sending/receiving between processors is necessary. This greatly simplifies the programming but the compilers are still either fairly immature or even in an experimental stage which does not always guarantee a great performance to say the least.

The advantages of DM-MIMD systems are clear: the bandwidth problem that haunts shared-memory systems is avoided because the bandwidth scales up automatically with the number of processors. Furthermore, the speed of the memory which is another critical issue with shared-memory systems (to get a peak performance that is comparable to that of DM-MIMD systems, the processors of the shared-memory machines should be very fast and the speed of the memory should match it) is less important for the DM-MIMD machines, because more processors can be configured without the afore-mentioned bandwidth problems.

Of course, DM-MIMD systems also have their disadvantages: The communication between processors is slower than in SM-MIMD systems and so the synchronisation overhead in case of communicating tasks is generally orders of magnitude higher than in shared-memory machines. Moreover, the access to data that are not in the local memory belonging to a particular processor have to be obtained from non-local memory (or memories). This is again on most systems very slow as compared to local data access. When the structure of a problem dictates a frequent exchange of data between processors and/or requires many processor synchro-



(a) Hypercubes, dimension 1−4.



(b) A 128−way fat tree.

Figure 2.7: *Some often used networks for DM machine types.*

nisations, it may well be that only a very small fraction of the theoretical peak speed can be obtained. As already mentioned, the data and task decomposition are factors that mostly have to be dealt with explicitly, which may be far from trivial.

It will be clear from the paragraph above that also for DM-MIMD machines both the topology and the speed of the data paths are of crucial importance for the practical usefulness of a system. Again, as in the section on SM-MIMD systems, the richness of the connection structure has to be balanced against the costs. Of the many conceivable interconnection structures only a few are popular in practice. One of these is the so-called hypercube topology as depicted in Figure 2.7 (a).

A nice feature of the hypercube topology is that for a hypercube with 2^d nodes the number of steps to be taken between any two nodes is at most d. So, the dimension of the network grows only logarithmically with the number of nodes. In addition, theoretically, it is possible to simulate any other topology on a hypercube: trees, rings, 2-D and 3-D meshes, etc. In practice, the exact topology for hypercubes does not matter too much anymore because all systems in the market today employ what is called “wormhole routing” or variants

thereof. This means that a message is send from node i to node j a header message is sent from i to j, resulting in a direct connection between these nodes. As soon as this connection is established, the data proper is sent through this connection without disturbing the operation of the intermediate nodes. Except for a small amount of time in setting up the connection between nodes, the communication time has become fairly independent of the distance between the nodes. Of course, when several messages in a busy network have to compete for the same paths, waiting times are incurred as in any network that does not directly connect any processor to all others and often rerouting strategies are employed to circumvent busy links if the connecting network supports it. Also the network nodes themselves have become quite powerful and, depending on the type of network hardware may send and re-rout message packages in a way that minimises contention.

Another cost-effective way to connect a large number of processors is by means of a *fat tree*. In principle a simple tree structure for a network is sufficient to connect all nodes in a computer system. However, in practice it turns out that near the root of the tree congestion occurs because of the concentration of messages that first have to traverse the higher levels in the tree structure before they can descend again to their target nodes. The fat tree amends this shortcoming by providing more bandwidth (mostly in the form of multiple connections) in the higher levels of the tree. One speaks of a N-ary fat tree when the levels towards the roots are N times the number of connections in the level below it. An example of a quaternary fat tree with a bandwidth in the highest level that is four times that of the lower levels is shown in Figure 2.7 (b).

A number of massively parallel DM-MIMD systems favour a 2- or 3-D mesh (torus) structure. The rationale for this seems to be that most large-scale physical simulations can be mapped efficiently on this topology and that a richer interconnection structure hardly pays off. However, some systems maintain (an) additional network(s) besides the mesh to handle certain bottlenecks in data distribution and retrieval [17]. Also in the IBM BlueGene/P system this philosophy has been followed.

A large fraction of systems in the DM-MIMD class employ crossbars. For relatively small amounts of processors (in the order of 64) this may be a direct or 1-stage crossbar, while to connect larger numbers of nodes multi-stage crossbars are used, i.e., the connections of a crossbar at level 1 connect to a crossbar at level 2, etc., instead of directly to nodes at more remote distances in the topology. In this way it is possible to connect many thousands of nodes through only a few switching stages. In addition to the hypercube structure, other logarithmic complexity networks like Butterfly, Ω, or shuffle-exchange networks and fat trees are often employed in such systems.

As with SM-MIMD machines, a node may in principle consist of any type of processor (scalar or vector) for computation or transaction processing together with local memory (with or without cache) and, in almost all cases, a separate communication processor with links to connect the node to its neighbours. Nowadays, the node processors are mostly off-the-shelf RISC or x86 processors sometimes enhanced by vector processors. A problem that is peculiar to DM-MIMD systems is the mismatch of communication vs. computation speed that may occur when the node processors are upgraded without also speeding up the intercommunication. In many cases this may result in turning computational-bound problems into communication-bound problems and one sees much research activities to find communication avoiding algorithms that try to minimise the communication even at the expense of recalculating intermediate results.

2.6 ccNUMA machines

As already mentioned in the introduction, a trend can be observed to build systems that have a rather small (up to 16) number of processors that are tightly integrated in a cluster, a Symmetric Multi-Processing (SMP) node. The processors in such a node are virtually always connected by a 1-stage crossbar while these clusters are connected by a less costly network. Such a system may look as depicted in Figure 2.8. Note that in Figure 2.8 all CPUs in a cluster are connected to a common part of the memory. (Figure 2.8 looks functionally identical to Figure 2.6, however, there is a difference that cannot be expressed in the figure: all memory is directly accessible by all processors without the necessity to transfer the data explicitely.)
The most important ways to let the SMP nodes share their memory are S-COMA (Simple Cache-Only Memory Architecture) and ccNUMA, which stands for Cache Coherent Non-Uniform Memory Access. Therefore, such systems can be considered as SM-MIMD machines. On the other hand, because the memory is

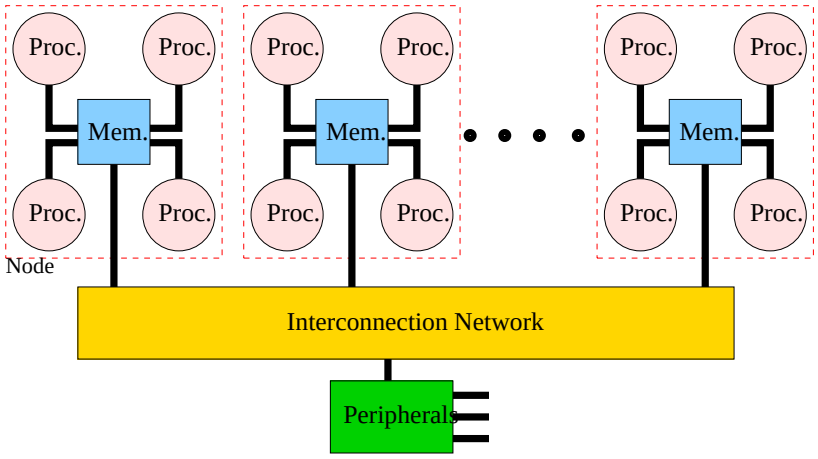


Figure 2.8: *Block diagram of a system with a "hybrid" network: clusters of four CPUs are connected by a crossbar. The clusters are connected by a less expensive network, e.g., a Butterfly network.*

physically distributed, it cannot be guaranteed that a data access operation always will be satisfied within the same time. In S-COMA systems the cache hierarchy of the local nodes is extended to the memory of the other nodes. So, when data is required that does not reside in the local node's memory it is retrieved from the memory of the node where it is stored. In ccNUMA this concept is further extended in that all memory in the system is regarded (and addressed) globally. So, a data item may not be physically local but logically it belongs to one shared address space. Because the data can be physically dispersed over many nodes, the access time for different data items may well be different which explains the term non-uniform data access. The term "Cache Coherent" refers to the fact that for all CPUs any variable that is to be used must have a consistent value. Therefore, is must be assured that the caches that provide these variables are also consistent in this respect. There are various ways to ensure that the caches of the CPUs are coherent. One is the *snoopy bus protocol* in which the caches listen in on transport of variables to any of the CPUs and update their own copies of these variables if they have them and are requested by a local CPU. Another way is the *directory memory*, a special part of memory which enables to keep track of all the copies of variables and of their validity.
Presently, no commercially available machine uses the S-COMA scheme. By contrast, there are several popular ccNUMA systems (like Bull's bullx s6010 series and the SGI Ultraviolet) commercially available. An important characteristic for NUMA machines is the *NUMA factor*. This factor shows the difference in latency for accessing data from a local memory location as opposed to a non-local one. Depending on the connection structure of the system the NUMA factor for various parts of a system can differ from part to part: accessing data from a neighbouring node will be faster than from a distant node in which possibly a number of stages of a crossbar must be traversed. So, when a NUMA factor is mentioned, this is mostly for the largest network cross-section, i.e., the maximal distance between processors.

Since the appearance of the multi-core processors the ccNUMA phenomenon also manifests itself within processors with multiple cores: first and second level cache belong to a particular core and therefore when another core needs data that not resides in its own cache it has to retrieve it via the complete memory hierarchy or the common third level cache of the processor chip. This is typically much (3rd level cache) to orders of magnitude (memory) slower than when it can be fetched from its local cache.

For all practical purposes we can classify these systems as being SM-MIMD machines also because special assisting hardware/software (such as a directory memory) has been incorporated to maintain a single system image although the memory is physically distributed.

2.7 Clusters

The adoption of clusters, collections of workstations/PCs connected by a local network, has virtually exploded since the introduction of the first Beowulf cluster in 1994. The attraction lies in the (potentially) low cost of both hardware and software and the control that builders/users have over their system. The interest for clusters can be seen for instance from the IEEE Task Force on Cluster Computing (TFCC) which reviews on a regular basis the current status of cluster computing [40]. Also books how to build and maintain clusters have greatly added to their popularity [38, 32]. As the cluster scene has become a mature and attractive market, large HPC vendors as well as many start-up companies have entered the field and offer more or less ready out-of-the-box cluster solutions for those groups that do not want to build their cluster from scratch (hardly anyone these days).

The number of vendors that sell cluster configurations has become so large that it is not possible to include all their products in this report. In addition, there is generally a large difference in the usage of clusters and they are more often used for *capability computing* while the integrated machines primarily are used for *capacity computing*. The first mode of usage meaning that the system is employed for one or a few programs for which no alternative is readily available in terms of computational capabilities. The second way of operating a system is in employing it to the full by using the most of its available cycles by many, often very demanding, applications and users. Traditionally, vendors of large supercomputer systems have learned to provide for this last mode of operation as the precious resources of their systems were required to be used as effectively as possible. By contrast, Beowulf clusters used to be mostly operated through the Linux operating system (a small minority using Microsoft Windows) where these operating systems either missed the tools or these tools were relatively immature to use a cluster well for capacity computing. However, as clusters become on average both larger and more stable, there is a trend to use them also as computational capacity servers too, particularly because nowadays there is a plethora of cluster management and monitoring tools. In [35] was looked at some of the aspects that are necessary conditions for this kind of use like available cluster management tools and batch systems. The systems then assessed are now quite obsolete but many of the conlusions are still valid: An important, but not very surprising conclusion was that the speed of the network is very important in all but the most compute bound applications. Another notable observation was that using compute nodes with more than 1 CPU may be attractive from the point of view of compactness and (possibly) energy and cooling aspects, but that the performance can be severely damaged by the fact that more CPUs have to draw on a common node memory. The bandwidth of the nodes is in this case not up to the demands of memory intensive applications.
As cluster nodes have become available with 4–8 processors where each processor also may have 8–16 processor cores, this issue has become all the more important and one might have to choose for capacity-optimised nodes with more processors but less bandwidth/processor core or capability-optimised nodes that contain less processors per node but have a higher bandwidth available for the processors in the node. This choice is not particular to clusters (although the phenomenon is relatively new for them), it also occurs in the integrated ccNUMA systems. Interestingly, as already remarked in the previous section, in clusters the ccNUMA memory access model is turning up now in the cluster nodes as for the larger nodes it is not possible anymore to guarantee symmetric access to all data items for all processor cores (evidently, for a core a data item in its own local cache will be available faster than for a core in another processor).

Fortunately, there is nowadays a fair choice of communication networks available in clusters. Of course Gigabit Ethernet or 10 Gigabit Ethernet are always possible, which are attractive for economic reasons, but have the drawback of a high latency ($\approx$ 10–40 μs). Alternatively, there are networks that operate from user space at high speed and with a latency that approaches these of the networks in integrated systems. These will be discussed in section 2.10.

2.8 Processors

In comparison to 10 years ago the processor scene has become drastically different. While in the period 1980–1990, the proprietary processors and in particular the vectorprocessors were the driving forces of the supercomputers of that period, today that role has been taken over by common off-the-shelf processors. In fact there is only one company left that produces vector systems (NEC) while all other systems that are

offered are based on RISC CPUs or x86-like ones. Therefore it is useful to give a brief description of the main processors that populate the present supercomputers and look a little ahead to the processors that will follow in the coming year. Still, we will be a bit more conservative in this section than in the description of the systems in general. The reason is processors are turned out at a tremendous pace while planning ahead for next generations takes years. We therefore tend to stick to the really existing components in this section or when already a β version of a processor is being evaluated.

The RISC processor scene has shrunken significantly in the last few years. The Alpha and PA-RISC processors have disappeared in favour of the Itanium processor product line and, interestingly, the MIPS processor line that appeared and disappeared again as they were used in the highly interesting SiCortex systems. Unfortunately SiCortex had to close down a few years ago and with it the MIPS processors disappeared again. In addition, the Itanium processor is not used in HPC anymore while IBM has terminated to offer systems with the PowerPC processor (except a modified version in its BlueGene system familiy).
The disappearance of RISC processor families demonstrates a trend that is both worrying and interesting: worrying because the diversity in the processor field is decreasing severely and, with it, the choice for systems in this sector. On the other hand there is the trend to enhance systems having run-of-the-mill processors with special-purpose add-on processors in the form of FPGAs or other computational accelerators because their possibilities in performance, price level, power consumption, and ease of use has improved to a degree that they offer attractive alternatives for certain application fields.

The notion of "RISC processor" alltogether has eroded in the sense that the processors that execute the Intel x86 (CISC) instruction set now have most of the characteristics of a RISC processor. Both the AMD and Intel x86 processors in fact decode the CISC instructions almost entirely into a set of RISC-like fixed-length instructions. Furthermore, both processor lines feature out-of-order execution, both are able to address and deliver results natively in 64-bit length, and the bandwidth from memory to the processor core(s) have become comparable to those of RISC processors. A distinguishing factor is still the mostly much larger set of registers in the RISC processors.
Another notable development of the last few years are the placement of multiple processor cores on a processor chip and the introduction of various forms of multi-threading. We will discuss these developments for each of the processors separately.

There are two processors one perhaps would expect in this section but are nevertheless not discussed: the Godson 3B and the Itanium Tukwila processors. The first processor, a Chinese one, based on the MIPS architecture, is not available in any machine that is marketed now or in the near future. The newest Itanium processor does not play a role anymore in the HPC scene and is therefore also omitted.

2.8.1 AMD Interlagos

All AMD processors are clones with respect to Intel's x86 Instruction Set Architecture. The 16-core Opteron variant called "Abu Dhabi" is no exception. It became available in November 2012. It is built with a feature size of 32 nm. The processor contains a 2-chip package for a total of 8 units, called "modules" by AMD, where each module has two integer units with 4 instruction pipelines each and two 128-bit floating-point units capable of executing 4 64-bit or 8 32-bit fused multiply-add operations each clock cycle. A diagram of a module is given in Figure 2.9. AMD mentions several reasons for this choice of functional units. One is that even for compute-intensive tasks on average more than 50% of the time is spent in integer operations. Also AMD prefers to double the integer units rather than relying on some form of software multithreading. AMD itself calls it Chip Multithreading (CMT) a term coined by SUN for its T2 multicore processors a few years ago. Another reason for this choice is to keep the power budget of the chip within reasonable bounds. The MMX units are able to execute both SSE4.1 and SSE4.2 instructions as well as Intel-compatible AVX vector instructions. In addition, these units are also capable of execution EAS instructions like already was possible in the latest Intel processors for fast en/decryption of data.
The clock cycle of the fastest variant employed in HPC, the X6386SE, has a clock cycle of 2.8 GHz with a boost to 3.5 GHz in turbo mode. Because of the composition of the modules the instruction throughput may vary considerably depending on the workload. As said the Interlagos chip harbours 8 modules as shown in Figure 2.10. Like in the former Interlagos processor HyperTransport 3.1 is used but the bandwidth to/from memory has increased from 28.8 GB/s to 40.3 GB/s due to the faster memory interface. As no

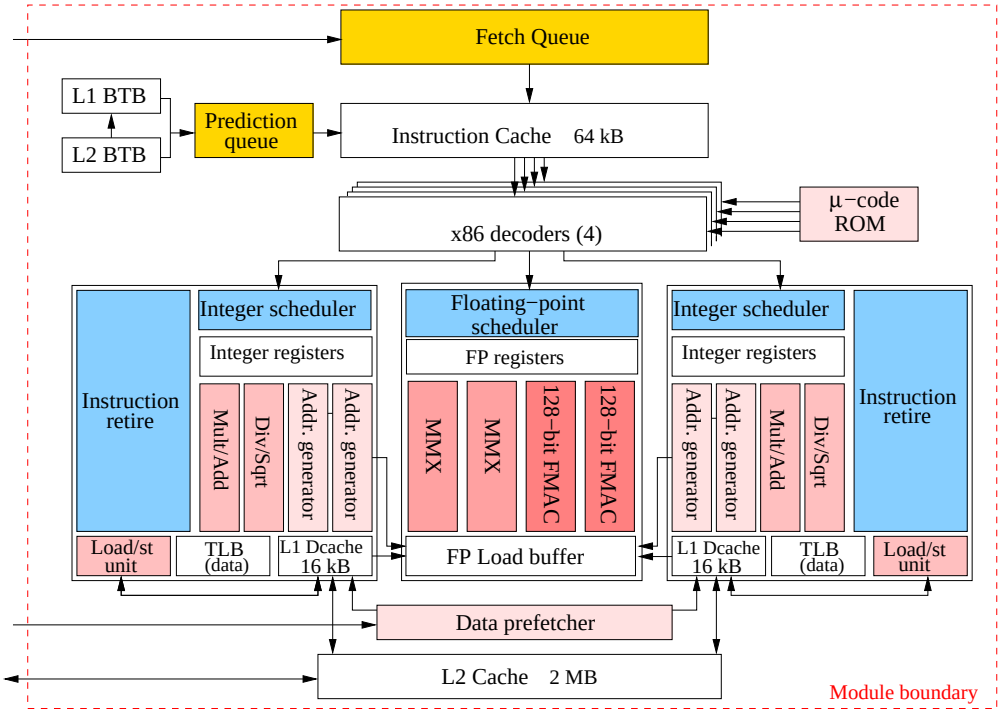


Figure 2.9: *Block diagram of an AMD Abu Dhabi processor module.*

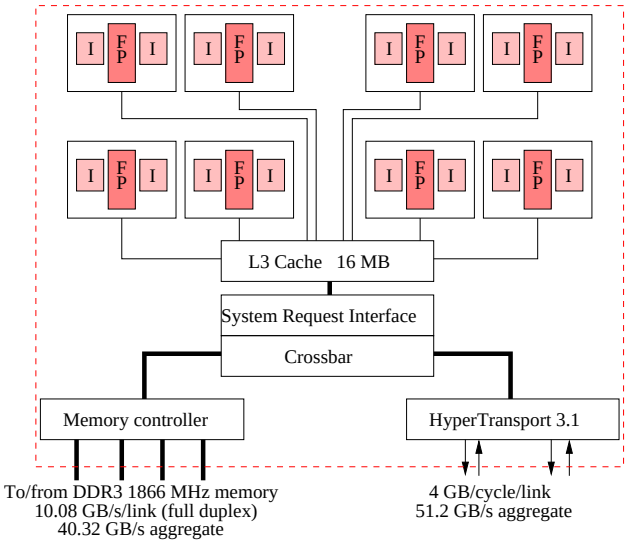


Figure 2.10: *Block diagram of the AMD Abu Dhabi processor lay-out.*

independent and systematic workload experiments are available for this processor yet it is not possible to say how the new ideas for this major change in architecture work out in practice. The Abu Dhabi is an incremental improvement of the former Interlagos processor: branch prediction and data prefetching have been improved. The L1 TLB size has been doubled from 32 to 64 entries and the also the L2 efficiency has improved. Furthermore, the energy efficiency is better than that of the earlier processor. The performance gain is in the order of 10% over the Interlagos, while the energy efficiency has improved by at least 10–20%.

2.8.2 IBM POWER7

The POWER7 processor is presently the processor of IBM's Power 775 HPC system line. In addition, Hitachi is offering a variant of its SR16000 (see 3.1.7) system with the POWER7 processor. Figure 2.11 shows the layout of the cores, caches, and memory controllers on the chip. The technology from which the

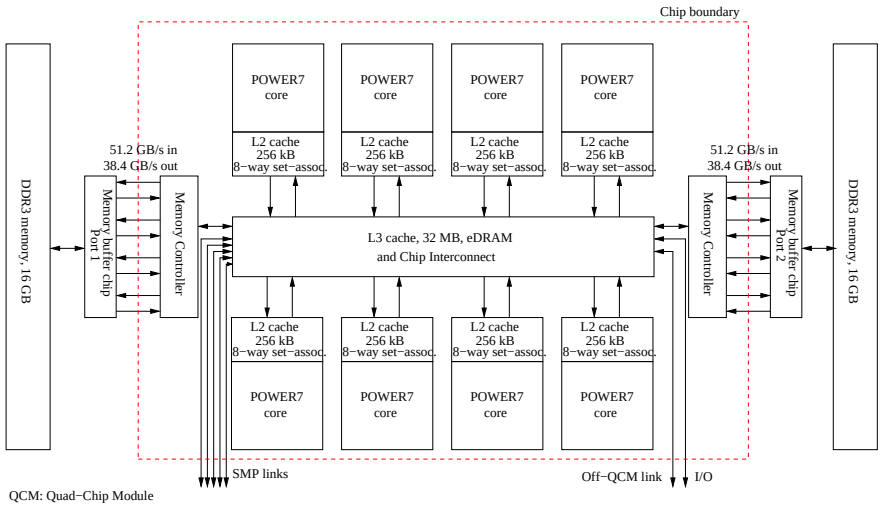


Figure 2.11: *Diagram of the IBM POWER7 chip layout*

chips are built is identical to that of the late POWER6: 45 nm Silicon-On-Insulator but in all other aspects the differences with the former generation are large: not only the number of cores has quadrupled. Also the memory speed has increased going from DDR2 to DDR3 via two on-chip memory controllers. As in earlier POWER versions the inbound and outbound bandwidth from memory to chip are different: 2 B/cycle in and 1.5 B/cycle out. With a bus frequency of 6.4 GHz and 4 in/out channels per controller this amounts to 51.2 GB/s inward and 38.4 GB/s outward. IBM asserts that an aggregate sustained bandwidth of $\approx$ 100 GB/s can be reached. Although this is very high in absolute terms with a clock frequency of 3.84 GHz for the processors this is no luxury. Therefore it is possible to run the chip in so-called TurboCore mode. In this case four of the 8 cores are turned off and the clock frequency is raised to 4.14 GHz thus almost doubling the bandwidth for the active cores. As one core is capable of absorbing/producing 16 B/cycle when executing a fused floating multiply-add operation the bandwidth requirement of one core at 4 GHz is already 64 GB/s. So, the cache hierarchy and possible prefetching are extremely important for a reasonable occupation of the many functional units.

Another new feature of the POWER7 with regard to its predecessor is that the L3 cache has been moved onto the chip. To be able to do this IBM chose to implement the 32 MB L3 cache in embedded DRAM (eDRAM) instead of SRAM as is usual. eDRAM is slower than SRAM but much less bulky and because the cache now is on-chip the latency is considerably lower (about a factor of 6). The L3 cache communicates with the L2 caches that are private to each core. The L3 cache is partitioned in that it contains 8 regions of 4 MB, one region per core. Each partition serves as a victim cache for the L2 cache to which it is dedicated and in addition to the other 7 L3 cache partitions.

Each chip features 5 10-B SMP links that supports SMP operation of up to 32 sockets.

Also at the core level there are many differences with its predecessor. A single core is depicted in Figure

2.12. To begin with, the number of floating-point units is doubled to four, each capable of a fused multiply-

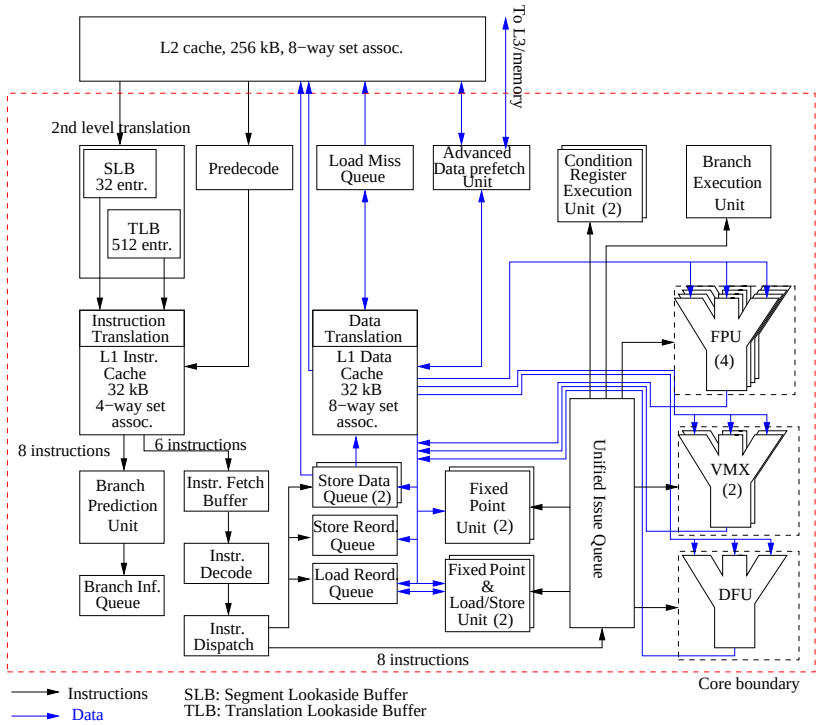


Figure 2.12: *Block diagram of the IBM POWER7 core.*

add operation per cycle. Assuming a clock frequency of 3.83 GHz this means that a peak speed of 30.64 Gflop/s can be attained with these units. A feature that was omitted from the POWER6 core has been re-implemented in the POWER7 core: dynamic instruction scheduling assisted by the load and load reorder queues. As shown in Figure 2.12 there are two 128-bit VMX units. One of them executes vector instructions akin to the x86 SSE instructions. However there is also a VMX permute unit that can order non-contiguous operands such that the VMX execute unit can handle them. The instruction set for the VMX unit is an implementation of the AltiVec instruction set that is also employed in the PowerPC processors. There are also similarities with the POWER6 processor: the core contains a Decimal floating-point unit (DFU)and a checkpoint recovery unit that can re-schedule operations that have failed for some reason.
Another difference that cannot be shown is that the cores now support 4 SMT threads instead of 2. This will be very helpful for the large amounts of functional units to be kept busy. Eight instructions can be taken from the L1 instruction cache. The instruction decode unit can handle 6 instructions simultaneously while 8 instructions can be dispatched every cycle to the various functional units.
The POWER7 core has elaborate power management features that reduces the power usage for parts that are idle for some time. There are two power-saving mode: *nap* mode and *sleep* mode. In the former the caches and TLBs stay coherent to re-activate quickly. In sleep mode, however, the caches are purged and the clock turned off. Only the mininum voltage to maintain the memory contents is applied. Obviously the wake-up time is longer in this case but the power saving can be significant.

As yet there is no POWER8 processor is produced, although most of its features are already known. A follow-on to the POWER7, the POWER7+ is available with a clock frequency of up to 4.4 GHz and an L3 cache that is increased from 32 to 80 MB. However, this chip is not offered in the HPC P775 servers from IBM but rather in there enterprise systems where the large L3 cache may have a larger advantage. In addition, the higher clock rate with the associated higher power consumption would be less favorable in highly floating-point oriented workloads.

2.8.3 BlueGene/Q processor

The BlueGene/Q processor, like its predecessors /L and /P uses a variant of the PowerPC family. This time of the A2 processor that was introduced last year by IBM as a network processor. The technology used is 45 nm SOI. Unlike the two earlier BlueGene processors this one is a full 64-bit processor. Where the BlueGene/P had 4 core/processor the BlueGene/Q processor contains no less than 18 cores. Sixteen of these are used for computation, one for OS tasks, and core 18 acts as a spare that in principle can be activated when one of the other cores fails, thus adding to the resiliency of the chip (although IBM made no promises that it actually will be a hot spare). In Figure 2.13 a scheme of the chip layout is given. In contrast to the earlier BlueGene models there is now only one type of network: a 5-D torus. The connections to the outside world are too complicated to depict in this diagram and further packaging details are deferred to the discussion of the system in section 3.1.8. The crossbar operates at 800 MHz and has a respectable bisection

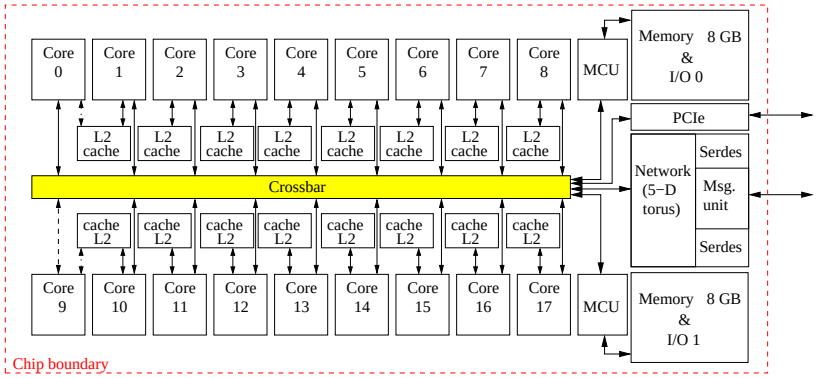


Figure 2.13: *Block diagram of an IBM BlueGene/Q processor chip.*

bandwidth of 563 GB/s, although it is not sufficient to feed all cores simultaneously.

The processor cores operate at a clock speed of 1.6 GHz, almost double that of the former BlueGene/P. As

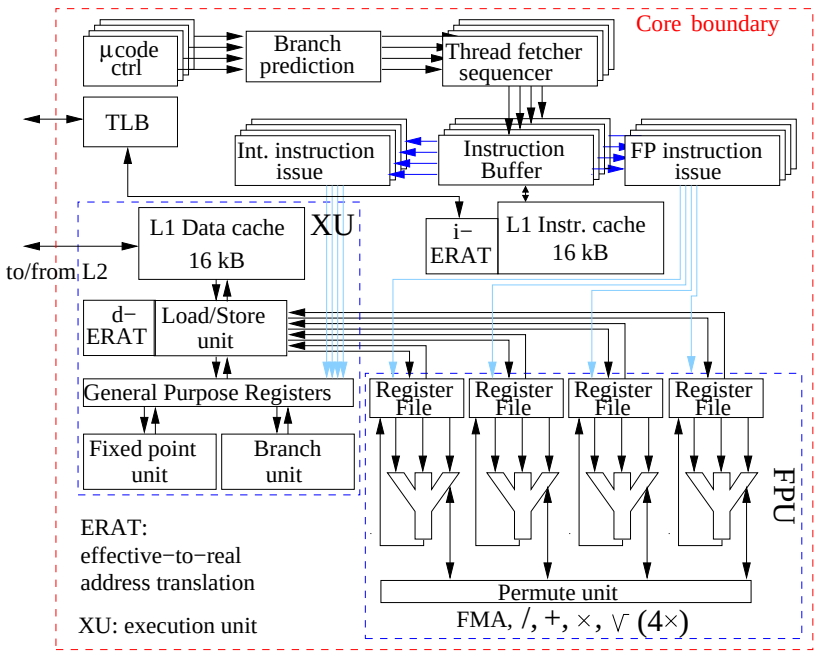


Figure 2.14: *Block diagram of an IBM BlueGene/Q processor core.*

can be seen from the diagram in Figure 2.14 4 FMA instructions can be executed per core, thus delivering a

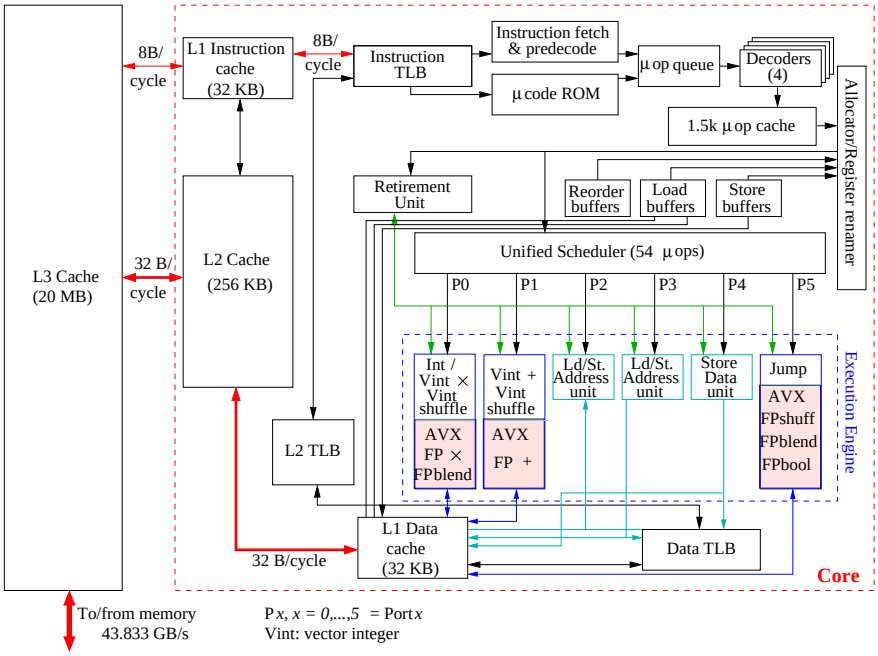


Figure 2.15: *Diagram of a Ivy Bridge core.*

12.8 Gflop/s peak speed per core. Per processor 16 GB of DDR3-1333 MHz memory is available, two times more per core than for the BlueGene/P. The data are fetched/stored through two memory controllers, each servicing 8 cores at an aggregate bandwidth of 47.2 GB/s. The core is capable of 4-way instruction issue, both for integer and load/store processing and for the floaing-point units. Instruction processing is however in-order, unlike in the former BlueGenes. For the floating-point units this potential bottleneck is mitigated by a Permute Unit that enables reordering strided and otherwise permuted operands to be processed in a SIMD way.

2.8.4 Intel Xeon

The current Xeon for HPC servers is the Ivy Bridge processor. It is a technology shrink from 32 nm in its predecessor, the Sandy Bridge, to 22 nm. This enabled the placement of 12 cores on the chip instead of 8. A block diagram of the core is shown (see Figure 2.15). Like its predecessor, the Sandy Bridge, is it made in 32 nm technology. The official name of the processor family is E5-2690. The clock cycle for the fastest models range from 2.4-2.7 GHz in standard mode to 3.2-3.7 GHz in turbo mode respectively.

To stay backwards compatible with the x86 (IA-32) Instruction Set Architecture which comprises a CISC instruction set Intel developed a modus in which these instructions are split in so-called micro operations (μ-ops) of fixed length that can be treated in the way RISC processors do. In fact the μ-ops constitute a RISC operation set. The price to be payed for this much more efficient instruction set is an extra decoding stage. Branch prediction has been improved and also a second level TLB cache been added.

The 1.5 kμ-op cache receives the instructions that are broken down to micro operations by 4 decoders and the schedulers can handle 54 of these simultaneously. Furthermore, the vector units can perform the AVX instruction set in 256-bit wide registers. Unlike in the SSE instructions, that are still supported, the source operands are not overwritten. So, they can be reused in subsequent operations. Like in the Sandy Bridge processor, instructions for the Advanced Encryption Standard (AES) are included that should speed up en/decryption tasks. Also, like in the Sandy Bridge, the use of 1 GB pages is supported.

As in the earlier Core architecture 4 μ-ops/cycle and some macro-instructions as well as some μ-ops can be fused, resulting in less instruction handling, easier scheduling and better instruction throughput because these fused operations can be executed in a single cycle. As can be seen in Figure 2.15 the processor cores have an execution trace cache which holds partly decoded instructions of former execution traces that can

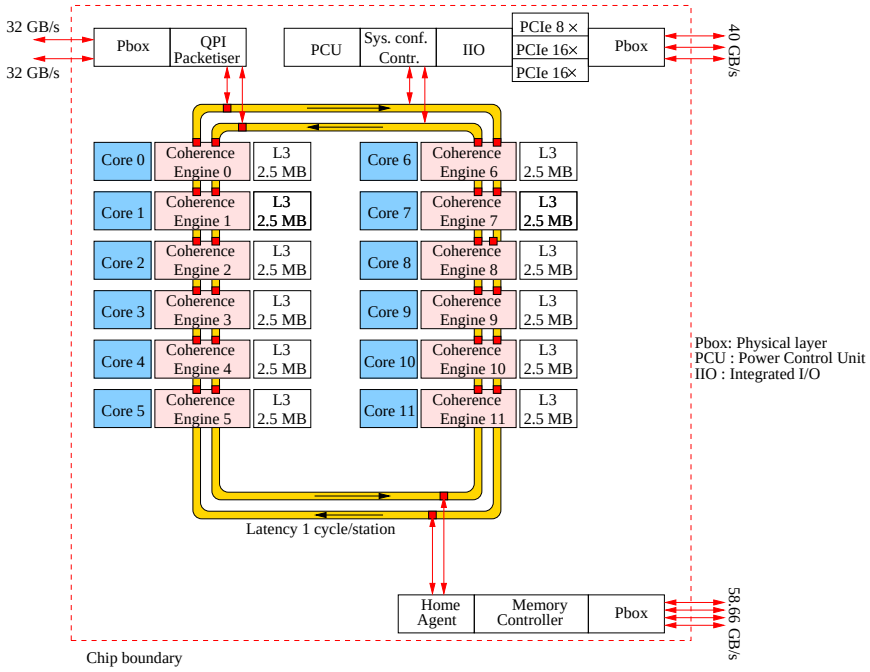


Figure 2.16: *Layout of an E5-2600 Ivy Bridge processor.*

be drawn upon, thus foregoing the instruction decode phase that might produce holes in the instruction pipeline. The allocator dispatches the decoded instructions, the μ-ops, to the unified reservation station that can issue up to 6 μ-ops/cycle to the execution units, collectively called the Execution Engine. Up to 128 μ-ops can be in flight at any time. Figure 2.15 shows that port 0 and port 1 drive two Integer ALUs as well as (vector) floating-point instructions. Port 5 only operates on floating-point instructions while ports 2–4 are dedicated to load/store operations.

The two integer Arithmetic/Logical Units at port 0 and 1 are kept simple in order to be able to run them at twice the clock speed. In addition there is an ALU at port 1 for complex integer operations that cannot be executed within one cycle. The length of the operands for these units is 256 bits.

A feature that cannot be shown in the figures is that the Ivy Bridge supports multi-threading much in the style of IBM's simultaneous multithreading. Intel calls it Hyperthreading. Hyperthreading was earlier introduced in the Pentium 4 but disappeared in later Intel processors because the performance gain was very low. Now with a much higher bandwidth and larger caches speedups of more than 30% for some codes have been observed with Hyperthreading. Another feature that cannot be shown is the so-called Turbo Mode already mentioned above. It means that the clock cycle can be raised from its nominal speed (2.7 GHz for the E5-2690) by steps of 133 MHz to up to 3.7 GHz as long as the thermal envelope of the chip is not exceeded. So, when some cores are relatively idle other cores can take advantage by operating at a higher clock speed.

The L3 cache is inclusive which means that it contains all data that are in the L2 and L1 cache. The consequence is that when a data item cannot be found in the L3 cache it is also not in any of the caches of the other cores and one therefore need not search them.

In the Ivy Bridge processor the cores are not fully interconnected which each other anymore but rather by two counter-rotating rings as shown in Figure 2.16. The red squares in the diagram represent the stations within the rings that inject or draw data into/from them. The latency between stations is 1 clock cycle. The maximum latency for updating an entry in the L3 cache is therefore $(12 + 2)/2 = 7$ cycles. Although the parts of L3 cache are drawn separately in the figure, all cores of course have access to all parts of the cache. The data only have to be transported via the rings to the core(s) that need them.

As can be seen in the diagram there are 2 QPI links that connect to the other CPU on the board at a bandwidth of 32 GB/s/link. The QPI links maintain cache coherency between the caches of the CPUs. The

aggregated memory bandwidth is 58.66 GB/s over 4 channels to 1833 MHz DDR3 memory. The I/O unit is integrated on the chip and with one 8 GB/s and two 16 GB/s PCI3 Gen3 ports have an aggregate bandwidth of 40 GB/s.

2.8.5 The SPARC processors

Since SUN has been taken over by Oracle the development of the SPARC processor family is split in two distinct lines: The SPARC T-series, multi-core heavily multi-threaded processors, of which the present T4 generation is the latest with 8 cores and 8 threads per core at a clock cycle of 2.5–3 GHz. This is the processor line used by Oracle. The processors in this line, however, are not for HPC use. The other processor line is the SPARC64 line of processors developed by Fujitsu which have features that are geared towards heavy computation. The current generation is the SPARC64 IXfx.
The processor was first produced already in end 2011 with a feature size of 40 nm and it is operated at a clock speed of 1.848 GHz somewhat lower than that of its predecessor, the VIIIfx that ran at 2 GHz. However, the number of cores has been doubled form 8 to 16. This results in a usage of only 110 Watt at a peak performance of 236.5 Gflop/s. In many respects the SPARC64 IXfx resembles its predecessor, the SPARC64 VIIIfx but there are some differences in the processor structure and in the instruction set that are meant to speed up floating-point computation. The chip layout is shown in Figure 2.17. The off-chip bandwidth to the memory is very high: 85 GB/s. The L1 instruction and data caches are 32 KB, and both are 2-way set-

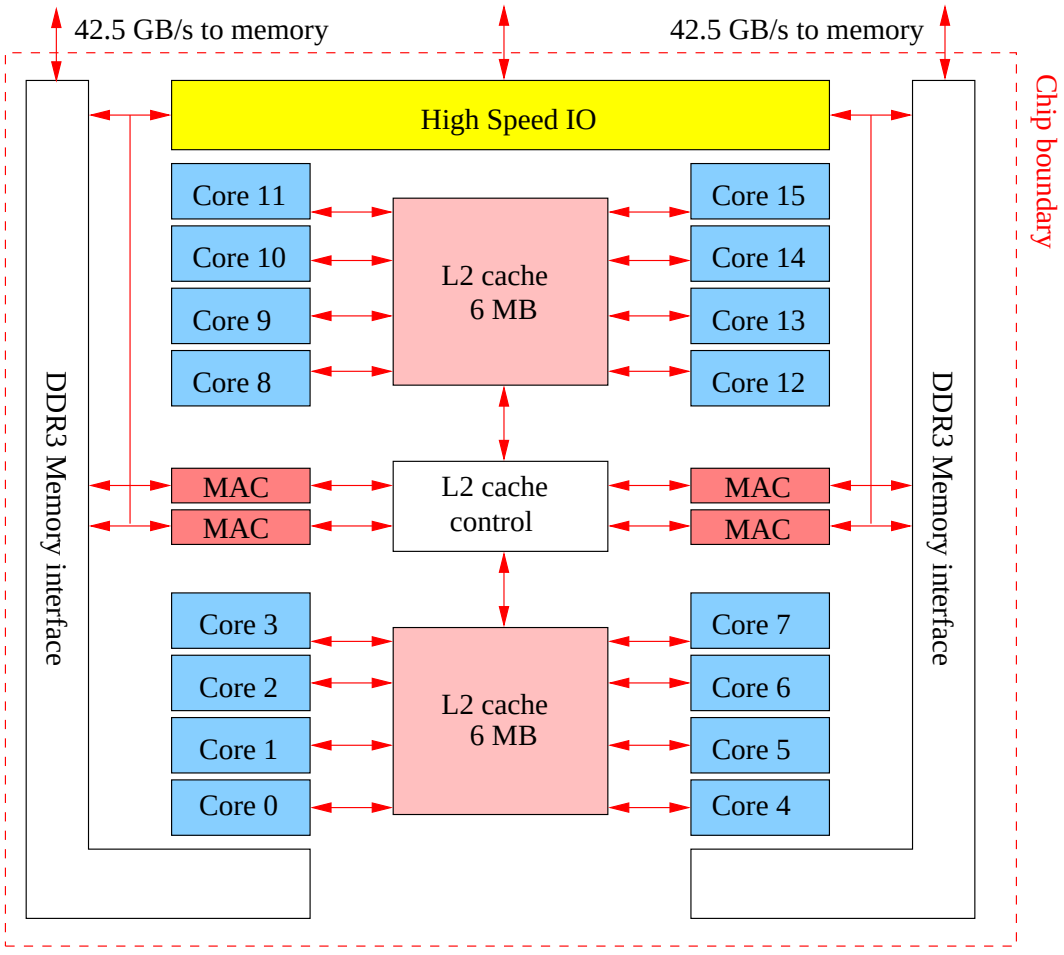


Figure 2.17: *Processor structure of the SPARCIXfx.*

associative. The L2 cache has increased from 6 to 12 MB. The register sizes are increased, however. Like all SPARC64 processors, the IXfx version has no L3 cache. A feature that cannot be displayed is the extension

of the instruction set with vector instructions which greatly reduce the overhead of vectorisable code as is demonstrated in [23]. Furthermore, there is a hardware retry mechanism that re-executes instructions that were subject to single-bit errors.
The Memory Management Unit (not shown in Figure 2.18) contains separate sets of Translation Look aside Buffers (TLB) for instructions and for data. Each set is composed of a 32-entry μTLB and a 1024-entry main TLB. The μTLBs are accessed by high-speed pipelines by their respective caches.
There is also an Instruction Buffer (IBF) than contains up to 48 4-byte instructions and continues to feed

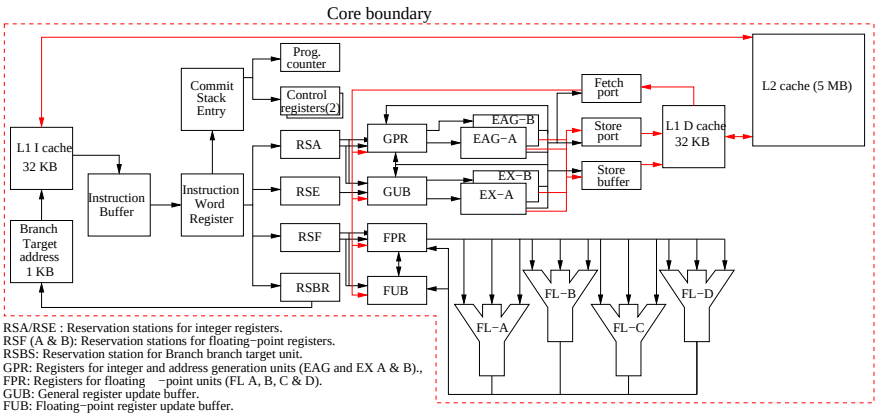


Figure 2.18: *Block diagram of a Fujitsu SPARC64 IXfx processor core.*

the registers through the Instruction Word Register when an L1 I-cache miss has occurred. A maximum of four instructions can be scheduled each cycle and find their way via the reservation stations for address generation (RSA), integer execution units (RSE), and floating-point units (RSF) to the registers. The general register file serves both the two Address Generation units EAG-A, and -B and the Integer Execution units EX-A and -B. The latter two are not equivalent: only EX-A can execute multiply and divide instructions. The floating-point register file (FPR) has, like the GPR, been extended: from 64 entries to 256. This greatly helps in heavy loop unrolling and in the vector operations that are supported. The FPR feeds the four floating-point units FL-A, FL-B, FL-C, and FL-D that all are capable of performing fused multiply-add operations. Consequently, a maximum of 8 floating-point results/cycle can be generated. The feedback from the execution units to the registers is decoupled by update buffers: GUB for the general registers and FUB for the floating-point registers. A few other features should be mentioned: the floating-point units support vectorised trigonometric functions, divide and square root operations, and finding maximum and minimum elements in a vector.
What cannot be shown in the diagrams is that, like the IBM and Intel processors, the SPARC IXfx is multi-threaded: dual-threaded in this case. As already remarked, the floating-point units are capable of a fused multiply-add operation, like the POWER processors, and so the theoretical peak performance with a clock cycle of 1.8.48 GHz, 16 cores per processor and 8 floating-point results per clock cycle is 236.5 Gflop/s/processor.

2.9 Computational accelerators

In the last few years computational accelerators have emerged and they have taken a firm foothold now. They come in various forms of which we will discuss some general characteristics. Accelerators are not a new phenomenon: in the 1980's, for instance, Floating Point Systems sold attached processors like the AP120-B with a peak performance of 12 Mflop/s, easily 10 times faster than the general purpose systems they were connected to. Also the processor array machines described in section 2.3 could be regarded as accelerators for matrix-oriented computations in their time. A similar phenomenon is on us at the moment. HPC users never tend to be content with the performance of the machines they have at their disposal and are continuously looking for ways to speed up their calculations or parts of them. Accelerator vendors are complying to this wish and presently there is a fair amount of products that, when properly deployed, can

deliver significant performance gains.
The scene is roughly divided in three unequal parts:

1. Graphical cards or Graphical Processing Units (GPUs as opposed to the general CPUs).
2. General floating-point accelerators.
3. Field Programmable Gate Arrays.

The appearance of accelerators is believed to set a trend in HPC computing. Namely, that the processing units should be diversified according to their abilities. Not unlike the occurence of different functional units within a CPU core[1]. This is leading to hybrid systems that incorporate different processors for different computational tasks. Of course, processor vendors can choose to (attempt to) integrate such special purpose processing units within their main processor line but for now it is not sure whether this will be a profitable course.
When speaking of special purpose processors, *i.c.*, computational accelerators, one should realise that they are indeed good at some specialized computations while totally unable to perform others. So, not all applications can benefit from them and for those which can, not all to the same degree. Futhermore, using accelerators in practice is mostly not trivial. Although the Software Development Kits (SDKs) for accelerators have improved enormously lately, for many applications it is still a challenge to obtain a significant speedup. An important factor is that data must be shipped in and out of the accelerator and the bandwidth of the connecting bus is in most cases a severe bottleneck. One generally tries to overcome this by overlapping data transport to/from the accelerator with processing. Tuning the computation and data transport task can be cumbersome. This hurdle has been recognised by several software companies like Acceleware, CAPS, and Rapidmind (now absorbed by Intel). They offer products that automatically transform standard C/C++ programs into a form that integrates the functionality of GPUs, multi-core CPUs (which are often also not used optimally), and, in the case of Rapidmind, of Cell processors.
There is one other and important consideration that makes accelerators popular: in comparison to general purpose CPUs they all are very power-effective. Of course they will do only part of the work in a complete system but still the power savings can be considerable which is very attractive these days.
We will now proceed to discuss the three classes of accelerators mentioned above.

2.9.1 Graphical Processing Units

Graphics processing is characterised by doing the same (floating-point) operation on massive amounts of data. To accommodate for this way of processing Graphical Processing Units (GPUs) consist of a large amount of relatively simple processors, fast but limited local memory, and fast internal buses to transport the operands and results. Until recently all calculations, and hence the results, were in 32-bit precision. This is hardly of consequence for graphics processing as the colour of a pixel in a scene may be a shade off without anyone noticing. HPC users often have similar computational demands as those in the graphical world: the same operation on very many data items. So, it was natural to look into GPUs with their many integrated parallel processors and fast memory. The first adopters of GPUs from the HPC community therefore disguised their numerical program fragments as graphical code (e.g., by using the graphical language OpenGL) to get fast results, often with remarkable speedups. Another advantage is that GPUs are relatively cheap because of the enormous amounts that are sold for graphical use in virtually every PC. A drawback was the 32-bit precision of the usual GPU and, in some cases more important, there was no error correction available. By carefully considering which computation really needs 64-bit precision and which does not and adjusting algorithms accordingly the use of a GPU can be entirely satisfactorily, however. GPU vendors have been quick in focussing on the HPC community. They tended to rename their graphics cards to GPGPU, general-purpose GPU, although the product was largely identical to the graphics cards sold in every shop. But there also have been real improvements to attract HPC users: 64-bit GPUs have come onto the market. In addition, it is no longer necessary to reformulate a computational problem into a piece of graphics code. Both ATI/AMD and NVIDIA claim IEEE 754 compatibility (being the floating-point computation standard) but neither of them support it to the full. Error correction as is usual for general purpose CPUs has become available (see

[1] In principle it is entirely possible to perform floating-point computations with integer functional units, but the costs are so high that no one will attempt it.

2.9.1.2). There are C-like languages and runtime environments available that makes the life of a developer for GPUs much easier: for NVIDIA this is CUDA, which has become quite popular with users of these systems. AMD/ATI is concentrating on standard OpenCL (see below). It is somewhat more cumbersome but still provides a much better alternative to emulating graphics code.
When one develops a code for a particular GPU platform it cannot be transferred to another without a considerable effort in rewriting the code. This drawback is taken up by the GPU vendors (and not only them). OpenCL should yield code that in principle is platform independent, thus protecting the development effort put into the acceleration of a program. Presently, Apple, ATI/AMD, Intel, and NVIDIA are members of the consortium that are willing to provide a OpenCL language interface. First experiences with OpenCL version 1.0 as provided by the Khronos Group showed generally low performances but one might expect that these may improve with the new enhanced release of OpenCL 1.2 of which the specification is available from November 2011.
Still, many HPC users/developers do not want to go through the trouble of extensively transforming their program. To meet this large audience Cray, NVIDIA, CAPS, and the Portland Group (recently bought by NVIDIA) have defined a set of comment directives/pragmas under the name of OpenACC the use of which should result in offloading portions of the code to an attached GPU. Although is may not exploit the GPU to the full, it is much easier to use and leaves the original code intact.

Another way to be (relatively) independent of the platform is to employ some language transformer. For instance, CAPS provides such transforming tools that can target different types of accelerators or multi-core CPUs. By CAPS' product HMPP the transformation is brought about by inserting pragmas in the C code or comment directives in Fortran code. Together with the compiler company the Portland Group and HMPP with this ability presently are the only ones that can accelerate Fortran code on general GPU accelerators. The Portland Group's CUDA/Fortran compiler, however, only targets NVIDIA GPUs.

In the following we describe some high-end GPUs that are more or less meant for the HPC community.

2.9.1.1 ATI/AMD

In November 2012 the latest product from ATI (now wholly owned by AMD) was announced. It is the ATI Firestream S10000 card. At the time of writing, almost a year later, only very vague information is available that does not allow to a reliable diagram of its internals to be given. In Table 2.9.1.1 we give some specifications as are now known. The s10000 components are made with 28 nm feature size. The peak power requirement has increased significantly from a 250 W range to 375 W. However, in fact *two* graphics engines are fitted into the s10000, to accomodate both computation and high-end graphics. The specifications given

Table 2.1: *Some specifications for the ATI/AMD Firestream s10000 GPU*

Number of cores	3584
Memory (GDDR5)	6 GB
Clock cycle	825 MHz
Internal memory bandwidth	2×240 GB/s
Peak performance (32-bit)	5.91 Tflop/s
Peak performance (64-bit)	1.48 Tflop/s
Power requirement, peak	$\leq$ 375 W
Interconnect (PCIe Gen3)	×16, 16 GB/s
ECC, Error correction	Yes
Floating-point support	Partial (32/64-bit)

indicate that per core 2 32-bit floating-point results per cycle can be generated, the result of an add and a multiply operation. The 64-bit performance is a quarter of this. Whether these results can be produced independently or result from fused operations is not known because of the lack of information. ATI/AMD has catched up with NVIDIA GPUs (discussed below) in that the Firestream s10000 now also supports error correction.

Like its direct competitor, NVIDIA, ATI offers a free Software Development Kit, SDK which supports OpenCL 1.2, Direct X11 and ComputeX. The earlier languages like BROOK+ and the very low level Close-To-Metal software development vehicles are no longer supported.

2.9.1.2 NVIDIA

NVIDIA is the other big player in the GPU field with regard to HPC. Its latest product is the Tesla K series, with code name Kepler, came out at the end of 2012. A successor may be expected in 2014. Of the K20 series we only discuss the fastest one, the K20X. A simplified block diagram is shown in Figure 2.19. The

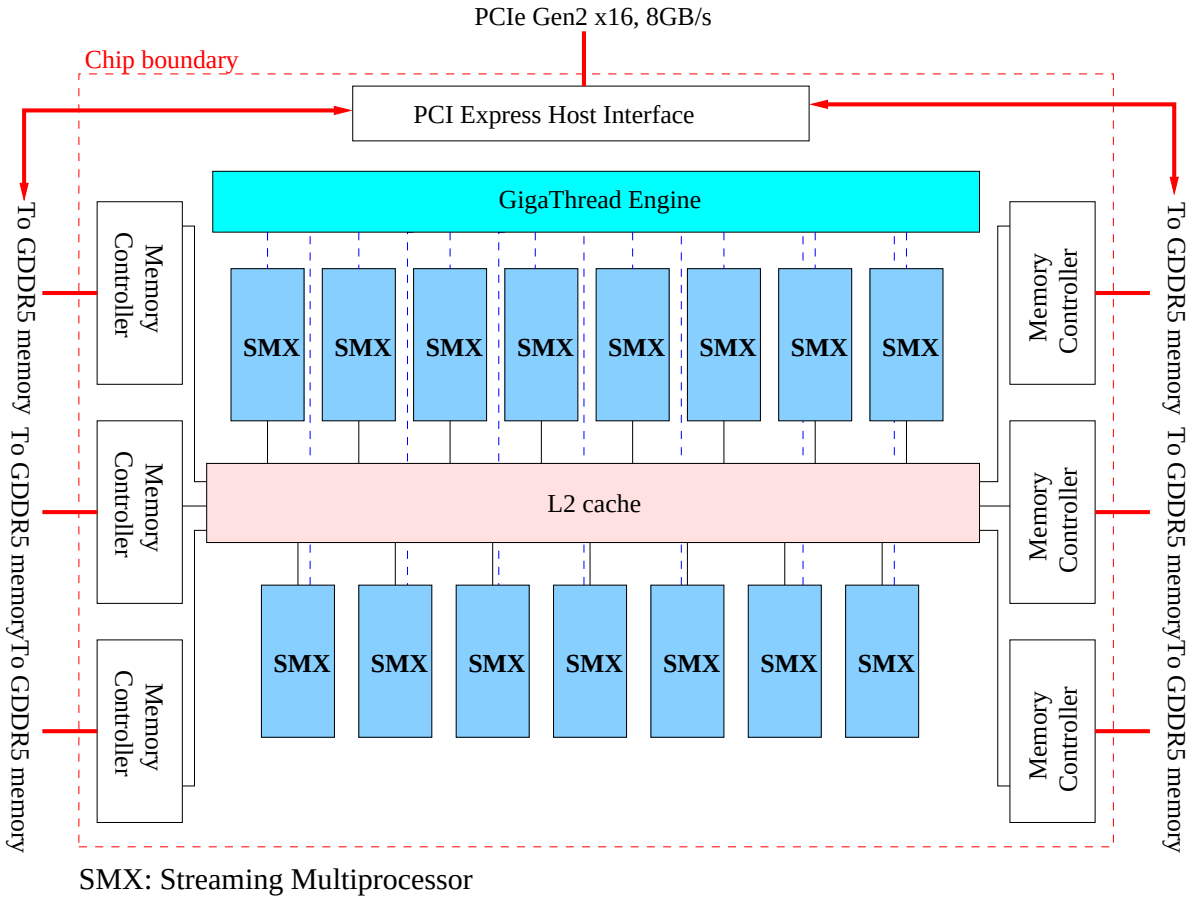


Figure 2.19: *Simplified block diagram of the NVIDIA Tesla K20X GPU.*

GigaThread Engine is able to schedule different tasks in the Streaming Multiprocessors (SMXs) in parallel. This greatly improves the occupation rate of the SMXs and thus the throughput. As shown in Figure 2.19 15 SMXs are present.
Each SMX in turn harbours 192 cores that used to be named Streaming Processors (SPs) but now are called CUDA cores by NVIDIA. A diagram of an SMX with some internals is given in Figure 2.20. Via the instruction cache 2 Warp schedulers (a warp is a bundle 32 threads) the program threads are pushed onto the cores. In addition each SMX has 32 Special Function Units (SFUs) that take care of the evaluation of functions, like goniometric functions, etc., that are more complicated than profitably can be computed by the simple floating-point units in the cores. Before we discuss some new features of the K20X that cannot be expressed in the diagrams we list some properties of the Tesla K20X in Table 2.9.1.2. As can be seen from the table the 64-bit performance is one-third of the single precision performance in accordance with the fact that there is one DP Unit for every core. Another notable item in the table is that the interconnection with the host is still based on PCIe Gen2 where one would expect it be Gen3 as was originally planned. Apparently NVIDIA was not able to make it work with the PCIe Gen3 port on Intel's latest chips

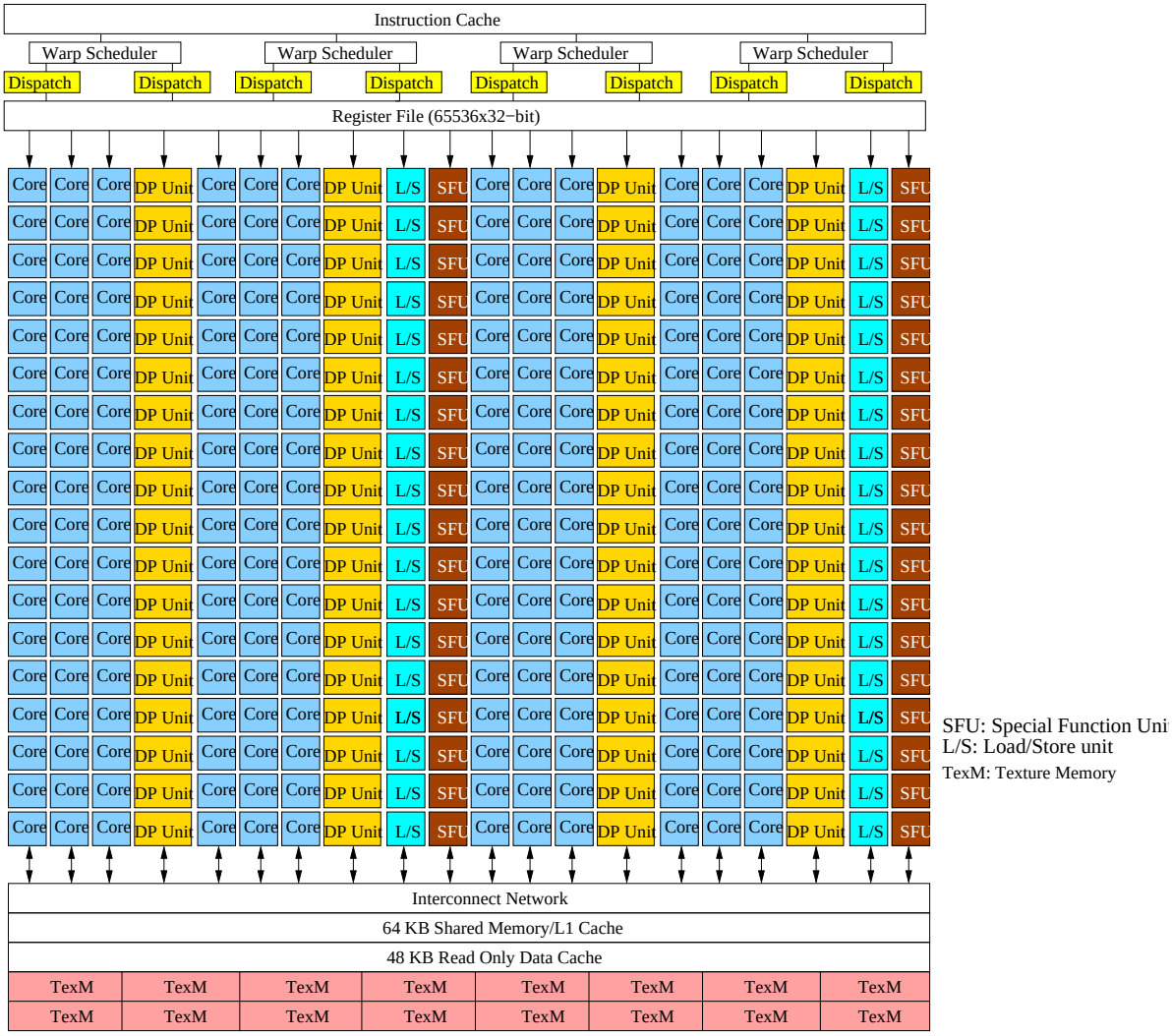


Figure 2.20: *Diagram of a Streaming Processor of the Tesla K20X.*

and NIVIDIA has therefore fallen back to Gen2. The peak power requirement given will probably be an appropriate measure for HPC workloads. A large proportion of the work being done will be from the BLAS library that is provided by NVIDIA, more specifically, the dense matrix-matrix multiplication in it. This operation occupies any computational core to the full and will therefore consume close to the maximum of the power.

The K20X supports some significant improvements over its predecessors that are especially of interest for HPC: one of these is what NVIDIA calls Hyper-Q that allows for 32 MPI tasks to run simultaneously on the GPU instead of just one. Apart from effectively de-serializing MPI tasks in this way it also allows for a better utilisation of the GPU. Another MPI-related feature is GPU Direct that enables MPI data exchange between GPUs without involving the host CPU. This does not only decrease the overhead of the CPU acknowledgment, it also omits the extra copies to the CPUs that host the GPUs which leads to a significant acceleration of the data exchange.

Perhaps the most interesting enhancement is the support of dynamic parallelism.This means that the GPU is able to initiate compute kernels independent from the host CPU. Where formerly each kernel had to be started by the host together with the corresponding data transfer associated with this kernel, with the dynamic parallelism feature the kernels initiated within the GPU already have their data available on the GPU. This cuts back on the data traffic between the GPU and the host, the most severe bottleneck in

Table 2.2: *Some specifications for the NVIDIA Tesla K20X GPU*

Number of cores	2688
Memory	6 GB
Internal bandwidth	250 GB/s
Clock cycle	732 MHz
Peak performance (32-bit)	3.52 Tflop/s
Peak performance (64-bit)	1.17 Tflop/s
Power requirement, peak	235 W
Interconnect (PCIe Gen2)	×16, 8 GB/s
ECC, Error correction	Yes
Floating-point support	Full (32/64-bit)

CPU-GPU computation.
Like ATI, NVIDIA provides an SDK comprised of a compiler named CUDA, libraries that include BLAS and FFT routines, and a runtime system that accomodates both Linux (RedHat and SuSE) and Windows. CUDA is a C/C++-like language with extensions and primitives that cause operations to be executed on the card instead of on the CPU core that initiates the operations. Transport to and from the card is done via library routines and many threads can be initiated and placed in appropriate positions in the card's memory so as not to cause memory congestion on the card. This means that for good performance one needs knowledge of the memory structure on the card to exploit it optimally. This is not unique to the K20X GPU, it pertains to the ATI Firestream GPU and other accelerators as well.
NVIDIA also supports OpenCL, though CUDA is at present much more popular among developers. Furthermore, the OpenACC programming model already mentioned in section 2.9 is supported. For Windows users the NVIDIA Parallel Nsight for Visual Studio is available that should ease the optimisation of the program parts run on the cards.

2.9.2 General computational accelerators

Although we have looked at the GPUs in the former section primarily from the point-of-view of computational accelerators, they are also full-blown high-end graphical processors in the first place. Several vendors have developed accelerators that did not have graphical processing in mind as the foremost application to be served (although they might not have been bad in this respect when compared to general CPUs). The future of the general computational accelerators seemed problematic: in principle it is entirely possible to make such accelerators that can compete with GPUs or with the FPGA-based accelerators discussed in section 2.9.3 but the volume will always be much lower than that of the other two accelerator variants which is reflected in the production cost.
The IBM/Sony/Toshiba Cell processor and the ClearSpeed accelerator have been discontinued for this reason but recently Intel has come up with its Xeon Phi accelerator which we will discuss below.

2.9.2.1 The Xeon Phi

In November 2012 Intel presented its first official many core product, the Xeon Phi. A successor, code name Knights Landing, may be expected in 2014. The Present Xeon Phi contains, depending on the model, 60 or 61 cores. We here discuss the fastest variant, the 7110P model that runs at a clock cycle of 1.1 GHz. The Xeon Phi can be regarded as Intel's answer primarily to the GPU-based accelerators. A distinct advantage for users is that no different programming model is required when the host processor is of the x86 type. A disadvantage which is shared with all other accelerators is that the Xeon Phi has its own (GDDR5) memory and data to be used or produced by the accelerator have to be transported to/from the accelerator via PCI3 Gen3 16×, i.e., at 16 GB/s.
In Figure 2.21 a rough diagram of the Xeon Phi is given as it proved very difficult to get sufficient details: for one thing, all Intel documentation refers to a "high-speed" ring to transport data and instructions between the cores but nowhere a bandwidth is stated. One can get an idea of this bandwidth knowing that the two

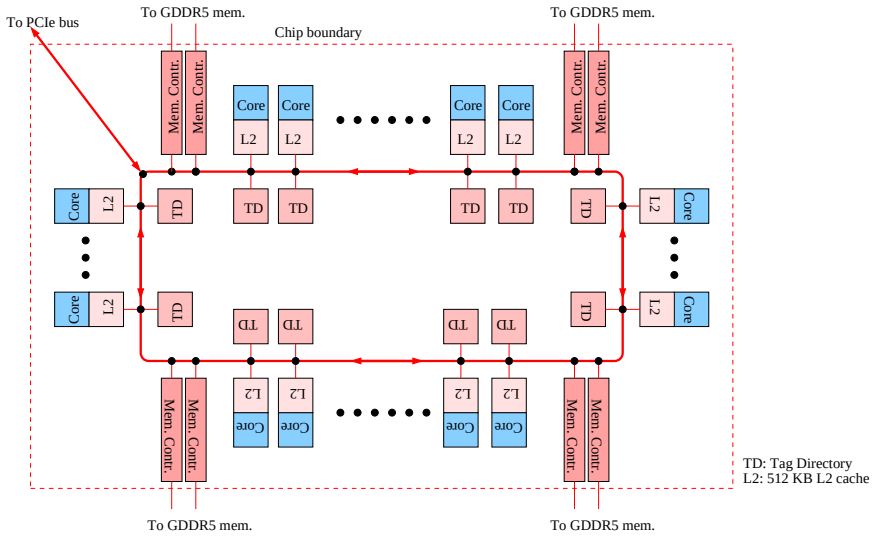


Figure 2.21: *Block diagram of an Intel Xeon Phi processor.*

counter-rotating rings (like in the Xeon Sandy Bridge, see **??**) are 512 bits wide in each direction and the latency between any neighbouring connection points on the ring is 1 clock cycle. With a clock frequency of 1.1 GHz the bandwidth in each direction should therefore be in the order of 70 GB/s. This matches nicely with the peak memory bandwidth of 352 GB/s to/from the 8 GB of GDDR5 memory. Apart from the rings that transport data there are rings for address fetching and cache coherency of the L2 caches that belong to the cores. The tag directories associated with the cores hold the addresses and their validity state. The addresses are spread evenly over the tag directories to avoid bottlenecks in fetching them. Also the memory controllers that give access to the memory are interspersed with the cores to give a smooth access to the data.
The cores in the Xeon Phi are derived from the Intel Pentium P54C but with many enhancements that should give it the peak performance of over 1 Tflop/s that is stated by Intel. As can be seen from Figure 2.22 the core contains a 512-bit wide vector unit capable of yielding 8 64-bit or 16 32-bit floating-point results per cycle. As the vector unit supports fused multiply-add operations actually 16 64-bit operations or 32 32-bit operations may be possible. The AVX instruction set executed in the Vector Processing Unit (VPU) includes mask operations which helps in executing loops with conditionals in them. In addition there are scatter-gather operations to deal with loop strides larger than 1 and an extended math unit that can provide transcendental function results. Instructions in the core are executed in-order which greatly simplifies the core logic. However, 4-wide multi-threading is supported (as suggested in the upper-left corner of Figure 2.22). This helps in executing a good level of instructions per cycle even without out-of-order execution. Instruction pipe 0 drives the VPU as well as the scalar floating-point part of the code while pipe 1 only drives the scalar integer processing in ALU0 and ALU1. According to Intel the x86-specific logic and the associated L2 area only occupy less than 2% of the die area while the core is able to execute the complete range of x86 instructions (be it not always very efficiently).
As remarked before, one can use the same SIMD pragmas/directives as are used for AVX instructions on X86 CPUs but there are also explicit offload pragmas/directives that causes a part of the code to be executed on the Phi processor after transporting the associated data.

2.9.3 FPGA-based accelerators

An FPGA (Field Programmable Gate Array) is an array of logic gates that can be hardware-programmed to fulfill user-specified tasks. In this way one can devise special purpose functional units that may be very efficient for this limited task. Moreover, it is possible to configure a multiple of these units on an FPGA that work in parallel. So, potentially, FPGAs may be good candidates for the acceleration of certain applications. Because of their versatility it is difficult to specify where they will be most useful. In general, though, they

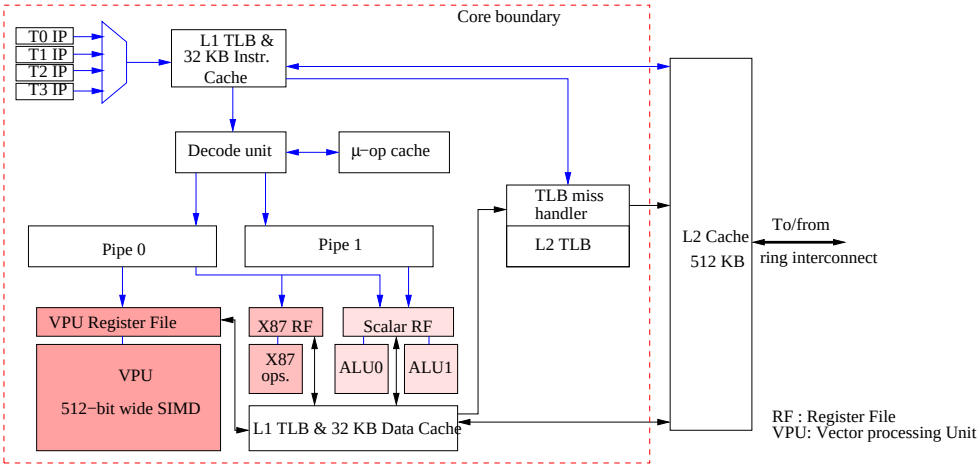


Figure 2.22: *Diagram of a Phi processor core.*

are not used for heavy 64-bit precision floating-point arithmetic. Excellent results have been reported in searching, pattern matching, signal- and image-processing, encryption, etc. The clock cycle of FPGAs is low as compared to that of present CPUs: 100–550 MHz which means that they are very power efficient. Vendors provide runtime environments and drivers that work with Linux as well as Windows.
Traditionally, FPGAs are configured by describing the configuration by means of a hardware description language (HDL), like VHDL or Verilog. This is very cumbersome for the average programmer as one not only has to explicitely define such details as the placement of the configured devices but also the width of the operands to be be operated on, etc. This problem has been recognised by FPGA-based vendors and a large variety of programming tools and SDKs have come into existence. Unfortunately, they differ enormously in approach and the resulting programs are far from compatible. Also for FPGA-based accelerators, like for GPUs, there is an initiative to develop a unified API that will assure compatibility between platforms. The non-profit OpenFPGA consortium is heading this effort. Various working groups concentrate on, for instance, a core library, an application library, and an API definition. There is no unified way to program FPGAs platform independently. However, the scene may change in the next few years as Altera, one of the major FPGA manufacturers, recently announced to provide an OpenCL SDK. This should make it much easier for C programmers to develop programs that take advantage of the FPGA capabilities. For instance, the Scottish FPGA integrator Nallatech is offering various Alterra Stratix V configurations to be used with OpenCL.
The two big players on the FPGA market are Altera and Xilinx. However, in the accelerator business one seldom will find these names mentioned, because the FPGAs they produce are packaged in a form that makes them usable for accelerator purposes.
It is not possible to fully discuss all vendors that offer FPGA-based products. One reason is that there is a very large variety of products ranging form complete systems to small appliances housing one FPGA and the appropriate I/O logic to communicate with the outside world. To complicate matters further, the FPGAs themselves come in many variants, e.g., with I/O channels, memory blocks, multipliers, or DSPs already configured (or even fixed) and one can choose for FPGAs that have for instance a PowerPC405 embedded. Therefore we present the FPGA accelerators here only in the most global way and necessarily incomplete.
In the following we will discuss products of vendors that have gone to great length to not expose their users to the use of HDLs, although for the highest benefits this not always can be avoided. Necessarily, we are here again somewhat arbitrary because this area is changing extremely rapidly.

2.9.3.1 Convey

The HC-2 is an example of the hybrid solutions that has came up to avoid the unwieldy HDL programming of FPGAs while still benefitting from their potential acceleration capabilities. The HC-2 comprises a familiar x86 front-end with a modified Centos Linux distribution under the name of Convey Linux. Furthermore,

there is a co-processor part that contains 4 Xilinx V5 or V6 FPGAs that can be configured into a variety of "personalities" that accomodate users from different application areas. Personalities offered are, e.g., Oil and Gas industry, Financial Analytic market, and the Life Sciences.

The Convey HC-2

The Convey HC-2(^{ex}) was announced in May 2012. It is the second generation of this type of machines. The main difference between the present model and the former HC-1 is the use of more recent Intel host processor (Intel Sandy Bridge, see section **??**) and/or a larger, newer Xilinx FPGA: a Virtex 5 LX330 in the HC-2 model and the larger Virtex 6 LX760 in the HC-2^{ex}.
In Figure 2.23 we give a diagram of the HC-2 co-processors's structure. A personality that may readily be

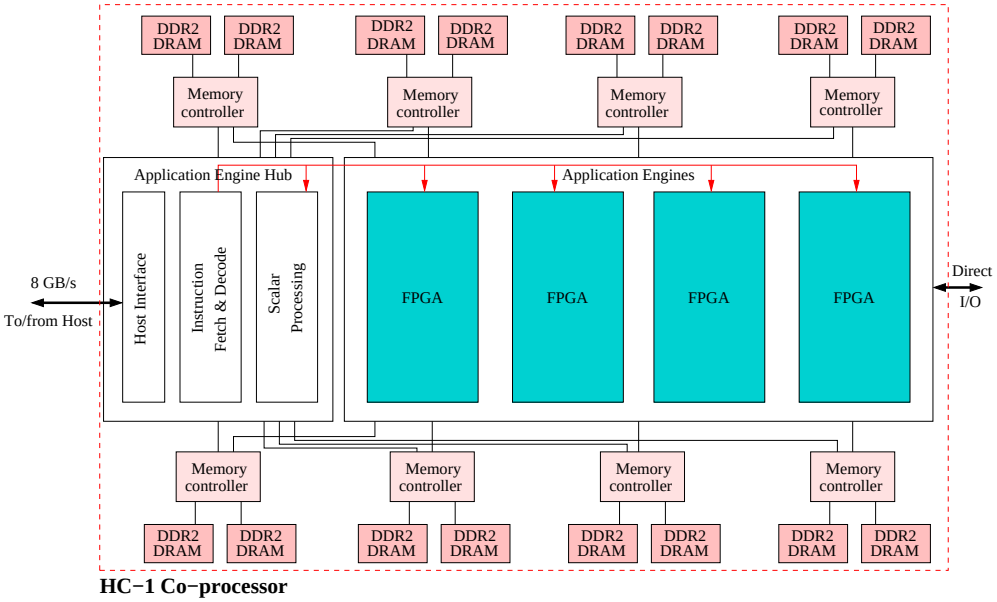


Figure 2.23: *Block diagram of the Convey HC-2 and HC-2^{ex}*

used for scientific an technical work is the vector personality. Thanks to the compilers provided by Convey standard code in Fortran and C/C++ can be automatically vectorised and executed on the vector units that have been configured in the the 4 FPGAs for a total of 32 function pipes. Each of these contains a vector register file, four pipes that can execute Floating Multiply-Add instructions, pipes for Integer, Logical, Divide, and Miscellaneous instructions and a Load/Store pipe. For other selected personalities the compilers will generate code that is optimal for the instruction mix generated for the appropriately configured FPGAs in the Application Engine.
The Application Engine Hub shown in Figure 2.23 contains the interface to the x86 host but also the part that maps the instructions onto the application engine. In addition, it will perform some scalar processing that is not readily passed on to the Application Engine.
Because the system has many different faces, it is hard to speak about *the* peak performance of the system. By now there is some experience with the HC-2 in various application areas and it appears that the system is doing well in a number of Life Science applications and in data analytics.

The Convey MX-100

This system has virtually the same structure as the HC-2 model. It is explicitely targeted at High Performance Data Analysis (HPA). This type of workload is characterised by large amounts of parallelism but highly irregular data access patterns and a low computational content. These are unfavourable conditions for standard RISC processors that try to hide memory latency by means of the cache hierarchy. For irregular data access with no discernable data locality, however, this will not work. Like in the HC-2, the XM-100 accesses data on a 64-bit double word basis from its Scatter-Gather memory, and, in addition, every double word contains a full/empty bit that allows for acceleration of graph searching and atomic in-memory

operations. The XM-100 has these properties in common with the Cray uRIKA systems. However, as the emphasis in this report is on HPC and not on HPA we will refrain from discussing the system in more detail.

2.9.3.2 Kuberre

Since May 2009 Kuberre markets its FPGA-based HANSA system. The information provided is extremely scant. The company has traditionally been involved in financial computing and with the rising need for HPC in this sector Kuberre has built a system that houses 1–16 boards, each with 4 Altera Stratix II FPGAs and 16 GB of memory in addition to one dual core x86-based board that acts as a front-end. The host board runs the Linux or Windows OS and the compilers.
For programming a C/C++ or Java API is available. Lately also the possibility to plug in MATLAB code modules has become avialable. As stated on its website, Kuberre is almost exclusively oriented to the financial analytic market, the little material that is accessible shows that libraries like, ScaLAPACK, Monte-Carlo algorithms, FFTs and Wavelet transforms are available. For the Life Sciences standard applications like BLAST, and Smith-Watermann are present. The standard GNU C libraries can also be linked seamlessly.
The processors are organised in a grid fashion and use a 256 GB distributed shared cache to overcome data access latency. The system comes configured as having 768 RISC CPUs for what are called "generic C/C++ programs" or as 1536 double precision cores for heavy numerical work. Unlike on Convey systems where only one personality at a time can be in execution, is possible to split the system to run up to 16 different "contexts" (equivalent to Convey's personalities, see 2.9.3.1). A part of the machine may be dedicated to a Life Science application where other parts work on encryption and numerical applications.
Like for the Convey HC-2 it is hardly possible to give performance figures but a fully configured machine with 16 boards should be able to obtain 250 Gflop/s on the Linpack benchmark which cannot really be regarded as "High Performance" these days. However, it may do very well on specialised workloads.
The material that is openly available does not allow to show a reliable block diagram but this may come about later when the system might be installed at sites that want to publicly evaluate it.

2.9.3.3 SRC

Until a few years ago SRC was the only company that sold a full stand-alone FPGA accelerated system, named the SRC-7. Now it has to share this space with Convey and Kuberre. Besides that, the so-called SRC-7 MAP station is sold, the MAP being the processing unit that contains an Altera Stratix IV EP4SE530 FPGA. Furthermore, SRC has the IMAP card as a product that can be plugged in a PCIe slot of any PC.
SRC has gone to great length to ban the term FPGA from its documentation. Instead it talks about implicit vs. explicit computing. In SRC terms implicit computing is performed on standard CPUs while explicit computing is done on its (reconfigurable) MAP processor. The SRC-7 systems have been designed with the integration of both types of processors in mind and in this sense it is a hybrid architecture also because shared extended memory can be put into the system that is equally accessible by both the CPUs and the MAP processors. We show a sketch of the machine structure in Figure 2.24. It shows that CPUs and MAP processors are connected by a 16×16 so-called Hi-Bar crossbar switch with a link speed of 7.2 GB/s. The maximum aggregate bandwidth in the switch 115.2 GB/s, enough to route all 16 independent data streams. The CPUs must be of the x86 or x86_64 type. So, both Intel and AMD processors are possible. As can be seen in the Figure the connection to the CPUs is made through SRCs proprietary SNAP interface. This accommodates the 7.2 GB/s bandwidth but isolates it from the vendor-specific connection to memory. Instead of configuring a MAP processor, also common extended memory can be configured. This allows for shared-memory parallelism in the system across CPUs and MAP processors.
The MAP station is a shrunk version of the SRC-7: it contains a x86(_64) CPU, a MAP processor, and a 4×4 Hi-Bar crossbar that allows Common Extended memory to be configured.
SRC and Convey are the only accelerator vendors that support Fortran. SRC does this through its development environment Carte. Like with Convey and Kuberre, also C/C++ is available. The parallelisation and acceleration are largely done by putting comment directives in Fortran code and equivalent pragmas in C/C++ code. Also, explicit memory management and prefetching can be done in this way. The directives/pragmas cause a bitstream to be loaded onto the FPGAs in one or more MAP processors that configures them and executes the target code. Furthermore, there is an extensive library of functions, a debugger and a performance analyzer. When one wants to employ specific non-standard funtionality, e.g.,

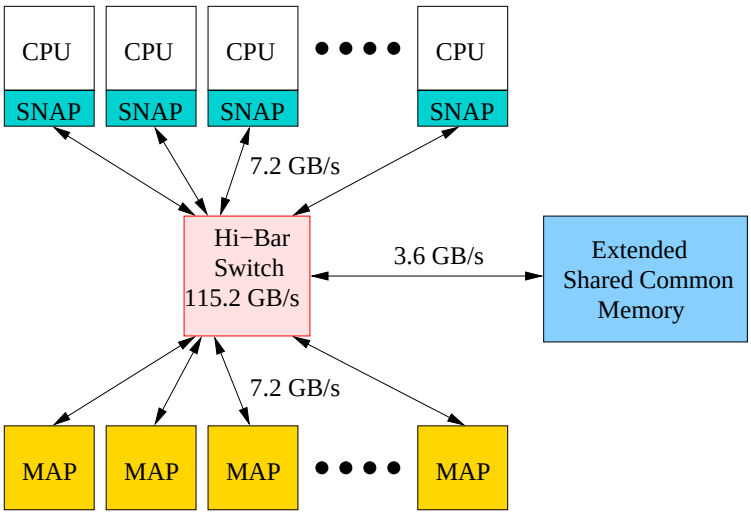


Figure 2.24: *Approximate machine structure of the SRC-7.*

computing with arithmetic of non-standard length, one can create a so-called Application Specific Funtional Unit. In fact, one then configures one or more of the FPGAs directly and until recently one had to fall back on VHDL or Verilog for this configuration. However, since Altera's announcement of OpenCL for its products one might attempt to make a C/OpenCL implementation.

2.10 Interconnects

Fast interprocessor networks are, together with fast processors, the decisive factors for both good integrated parallel systems and clusters. In the early days of clusters the interprocessor communication, and hence the scalability of applications, was hampered by the high latency and the lack of bandwidth of the network that was used (mostly Ethernet). This situation has changed very much and to give a balanced view of the possibilities opened by the improved networks a discussion of some of these networks is in order. The more so as some of these networks are, or have been employed also in "integrated" parallel systems.
Of course Gigabit Ethernet (GbE) is universally available and with a maximum theoretical bandwidth of 125 MB/s would be able to fulfill a useful role for some applications that are not latency-bound. Furthermore, also 10 Gigabit Ethernet (10 GigE) is increasingly offered. The adoption of Ethernet is hampered by the latencies that are incurred when the TCP/IP protocol is used for the message transmission. In fact, the transmission latencies without this protocol are much lower: about 5 μs for GbE and 0.5 μs for 10GbE. Using the TCP/IP protocol, however, gives rise to latencies of somewhat less than 40 μs and in-switch latencies of 30–40 μs for GbE and a 4–10 μs latency for 10GbE. As such it is not quite at par with the ubiquitous Infiniband innterconnects with regard to latency and bandwidth. However, the costs are lower and may compensate for a somewhat lower performance in many cases. Various vendors, like Myrinet and SCS, have circumvented the problem with TCP/IP by implementing their own protocol thus using standard 10GigE equipment but with their own network interface cards (NICs) to handle the proprietary protocol. In this way latencies of 2–4 μs can be achieved: well within the range of other network solutions. Mellanox now provides 40 GbE with an application latency of 4μs.

We restrict ourselves here to networks that are independently marketed as the proprietary networks for systems like those of Cray and SGI are discussed together with the systems in which they are incorporated. We do not pretend to be complete because in this new field players enter and leave the scene at a high rate. Rather we present main developments which one is likely to meet when one scans the high-performance computing arena.

A complication with the fast networks offered for clusters is the connection with the nodes. Where in integrated parallel machines the access to the nodes is customised and can be made such that the bandwidth

of the network matches the internal bandwidth in a node, in clusters one has to make do with the PCI bus connection that comes with the PC-based node. The type of PCI bus which ranges from 32-bit wide at 33 MHz to 64-bit wide at 66 MHz determines how fast the data from the network can be shipped in and out the node and therefore the maximum bandwidth that can be attained in internode communication. In practice the available bandwidths are in the range 110–1024 MB/s. In 1999 the coupling started with PCI-X 1.0 250 MB/s 64-bit wide, 66 MHz per lane. It was replaced by PCI-X 2.0 at double speed and subsequently by PCI Express (PCIe). Currently the third generation, PCIe Gen3 is the most common form at 1 GB/s for a single lane but often up to 16 lanes are employed, up to 16 GB/s. This PCIe Gen3 ×16 is also often used to connect accelerators to the host system. In September 2011 the first PCI Gen3 became available. As 1×, 2×, 4×, 8×, 12×, 16×, and 32×multiple data lanes are supported this is fast enough for the host bus adapters of any communication network vendor sofar. Presently, PCI Gen3 is for instance used by Cray (see 3.1.2) and in Mellanox's Infiniband (see below).

An idea of network bandwidths and latencies for some networks, both propriety and vendor-independent is given in Table 2.10. Warning: The entries are only approximate because they also depend on the exact switch and host bus adapter characteristics as well as on the internal bus speeds of the systems. The circumstances under with these values were obtained was very diverse. So, there is no guarantee that these are the optimum attainable results. Obviously, we cannot give maximum latencies as these depend on the system size and the interconnect topology.

Table 2.3: *Some bandwidths and minimal latencies for various networks as measured with an MPI Ping-Pong test.*

	Bandwidth	Latency
Network	GB/s	μs
Arista 10GbE (stated)	1.2	4.0
BLADE 10GbE (measured)	1.0	4.0
Cray SeaStar2+ (measured)	6.0	4.5
Cray Gemini (measured)	6.1	1.0
Cray Aries (measured)	9.5	1.2
SGI NumaLink 5 (measured)	5.9	0.4
Infiniband, FDR14 (measured)	5.8	< 1

2.10.1 Infiniband

Infiniband has become rapidly the most widely used medium for internode networks. The specification was finished in June 2001. From 2002 on a number of vendors has started to offer their products based on the Infiniband standard. A very complete description (1200 pages) can be found in [31]. Infiniband is employed to connect various system components within a system. Via Host Channel Adapters (HCAs) the Infiniband fabric can be used for interprocessor networks, attaching I/O subsystems, or to multi-protocol switches like Gbit Ethernet switches, etc. Because of this versatility, the market is not limited just to the interprocessor network segment and so Infiniband has become relatively inexpensive because the high volume of sellings that is presently realised. The characteristics of Infiniband are rather nice: there are product definitions both for copper and glass fiber connections, switch and router properties are defined and for high bandwidth multiple connections can be employed. Also the way messages are broken up in packets and reassembled as well as routing, prioritising, and error handling are all described in the standard. This makes Infiniband independent of a particular technology and it is, because of its completeness, a good basis to implement a communication library (like MPI) on top of it.
Conceptually, Infiniband knows of two types of connectors to the system components, the Host Channel Adapters (HCAs), already mentioned, and Target Channel Adapters (TCAs). The latter are typically used to connect to I/O subsystems while HCAs are the connectors used in interprocessor communication.
In Table 2.10.1 we list the bandwidth for the 1 and 4 links and data rates that are presently of interest as in interconnects mostly 4-wide Infiniband is used. Of the data rates[2] listed in the table presently DDR, QDR

Table 2.4: *Theoretical bandwidth for some Infiniband data rates for 1 and 4 links.*

	SDR GB/s	DDR GB/s	QDR GB/s	FDR-10 GB/s	FDR GB/s	EDR GB/s
1 link	0.25	0.5	1.0	1.25	1.71	3.13
4 links	1.0	2.0	4.0	5.0	6.82	12.5

and FDR(-10) are in use. At this moment mostly QDR and FDR are offered. The bandwidth difference between FDR-10 and FDR (also called FDR-14) lies in the different error correction coding schemes: from SDR up till FDR-10 an 8/10 bit error correction scheme is used. Meaning that of every 10 bits 8 carry data while 2 bits are dedicated to error correction. In FDR and EDR a different scheme is used in which of every 66 bits 64 bits represent data and 2 bits are used for error correction.

Messages can be sent on the basis of Remote Memory Direct Access (RDMA) from one HCA/TCA to another: a HCA/TCA is permitted to read/write the memory of another HCA/TCA. This enables very fast transfer once permission and a write/read location are given. A port together with its HCA/TCA provide a message with a 128-bit header which is IPv6 compliant and that is used to direct it to its destination via cut-through wormhole routing: In each switching stage the routing to the next stage is decoded and send on. Short messages of 32 B can be embedded in control messages which cuts down on the negotiation time for control messages.
To take advantage of the speed of QDR at least PCIe Gen2 $\times 8$ must be present at the nodes to which the HCAs are connected and for FDR PCIe Gen3 is required. The switches can be configured in any desired topology but in practice a fat tree topology is mostly preferred (see Figure 2.7b, section 2.5). It depends of course on the quality of the MPI implementation put on top of the Infiniband specifications how much of the raw speed can be realised. FDR-based interconnects are now becoming routinely deployed and bandwidths of 5–6 GB/s and an MPI latency of < 1 μs for small messages is quoted by Mellanox, one of the large Infiniband vendors. The in-switch latency is typically about 200 ns. Until early 2012 QDR Infiniband products were available from Mellanox and Qlogic. However, Qlogic has during 2012 been absorbed by Intel. Presumably with the intent of producing their own interconnect technology in the near future. For the moment we are in the rather uncomfortable situation that only Mellanox is left as an Infiniband product vendor.

[2]For the full names of the abbrevated data rates see the glossary.

3 Recount of (almost) available systems

In this section we give a recount of the types of systems discussed in the former section and that are marketed presently or will appear within 6 months from now. When vendors market more than one type of machine we will discuss them in distinct subsections. So, for instance, we will discuss IBM systems under entries BlueGeneP/Q, and eServer p775 because they have a different structure.

As already remarked in the Introduction we will not discuss clusters here but restrict ourselves to "integrated" parallel systems. However, the distinction between clusters and integrated systems becomes less and less clear as the network speed of clusters and integrated systems are (almost) comparable and the number of cores in a cluster node also approaches (or in the case of the Fujitsu FX10 is equal to) the number found in the integrated systems. The criterion we use consists of the special measures a vendor has taken to more tightly integrate the nodes in the system than what is found in the standard cluster. Such measures may be in hardware, like special barrier registers, or software, like a vendor-optimised fast intra-node MPI, or both. It may well be that even these kinds of distinctions will disappear over time. In that case the difference will have disappeared completely and we will add the (former) clusters to this overview. Still, we want to mention a few systems here that do not fall within the criteria maintained in the regarding clusters because they represent important systems but cannot be described easily in the usual terms for one or more reasons. So, below one will find a few entries that give some information about such configurations but cannot always provide the details one should like to give. Just because we want the reader to be aware of these systems we include them. Because the criteria are so vague, we cannot claim consistency in this respect.
A fair number of vendors now include GPU-enhanced systems in their portfolio. If sufficient information is available on these systems we also will discuss them, although it will be often impossible to provide reliable performance results for such systems, mainly because of the narrow range of algorithms where they can usefully be applied. If, at all credible we will quote the theoretical peak performance as given by the vendors for these machines.

A reservation with respect to the word "available" in the heading of this section is in order: rather *theoretically* available is a better description of the situation. This has nothing to do with the technical ability of the vendors to produce the systems described but everything with the ability of the vendor to invest in selling the system: when a large system is purchased this ought to come with the required maintenance and support. Some vendors cannot or will not sell a system in certain parts of the world because they are not willing or able to provide the necessary maintenance and/or support for the system that is theoretically available. For instance, it is not possible to buy a large Hitachi SR16000 in Europe. Nevertheless, we still discuss these systems in the section below for general interest and for those that are located in a part of the world where these vendors deem the selling of their systems economically viable.

3.1 System descriptions

The systems are presented alphabetically. The "Machine type" entry shortly characterises the type of system as discussed in the former chapter: Processor Array, ccNUMA, etc.

3.1.1 The Bull bullx systems.

3.1.1.1 The Bull bullx B50x, B70x and R42x systems.

Machine type: Hybrid distributed-memory system.
Models: bullx blade system B51x, B70x; bullx R42x E2/F2.
Operating system: Linux, Windows Server 2008
Connection structure: Variable.
Compilers: Intel's Fortran 95, C(++).
Vendors information Web page: `www.bull.com/bullx/`
Year of introduction: 2012.

System parameters:

Model	Blade B51x and B70x/R42x E2/F2
Clock cycle	up to 2.7 GHz
Theor. peak performance	3.53 Tflop/s/blade chassis
	820 Gflop/s/R424 server unit
Accelerator	NVIDIA Tesla S2050 (Kepler K20X)
	(Intel Xeon Phi)
Main Memory	$\leq$ 256 GB/blade on server unit
No. of processors	Variable
Communication bandwidth	
Point-to-point	4–6.8 GB/s (QDR or FDR Infiniband)
Aggregate peak	Variable

Remarks:
As already stated before, it becomes more and more difficult to distinguish between clusters and what used to be called "integrated" parallel systems as in the latter type increasingly standard components are employed that also can be found in any cluster. For the new bullx systems, available from Bull since spring 2009 this is certainly the case. There are, however, a number of distiguishing features of the bullx systems that made us decide to discuss them in this overview.
The systems come in three variants: two blade systems with 18 blades in a 7U chassis. The two blade systems are similar except in the cooling: the B510 series is air-cooled while the equivalent B700 series has direct liquid cooling, i.e., water is run through a copper heat sink blade fitted onto the board that holds the compute components like the CPUs (and/or accelerators, see below) and the memory. The other type of system is based on 1U units that pack 2 boards together, each containing 2 processors. The processor employed in both models is Intel's 8-core Sandy Bridge processor discussed in section 2.8.4. The density of both types of systems is equal and it is up to the preference of the customer what type is chosen.
The blade B510 system becomes hybrid, i.e., it integrates GPU accelerators or by putting B515 blades in the system. The B505s have a double-blade form factor that contain 2 Sandy Bridge processors and two NVIDIA Tesla S2050 parts, also known as the NVIDIA Fermi card. Although not yet officially announced, Bull will replace the Teslas by the recent Kepler K20X in the near future (see 2.9.1.2). Likewise, Intel's Xeon Phi accelerators will be offered (see 2.9.2.1) in the same form factor, both having a peak speed of over 1 Tflop/s.
For the R42x E2-based system there is 1U enclosure containing an S2050 GPU processor in which case it is called a R423 unit. The R424 packs four processors in 2U. The same goes for the R425 which contains 4 Sandy Bridge processors and 2 S2050 GPUs. The F2 model is identical to the E2 model, except that it allows for extended storage with SAS disks and RAID disks. In all cases shortly NVIDIA's K20X GPU and Intel Xeon Phi are available. Both for the blade and the rack systems instead of spinning disks also SSD storage is supported.

For the rack systems QDR Infiniband (see section 2.10.1) is available as an interconnection medium. For in the blade-based models a 36 QDR or FDR port module is integrated in the 7U chassis holding the blades. Of course the topology between chassis or rack units is up to the customer and therefore variable with respect to global bandwidth and point-to-point latencies.

Measured Performances:
For a 77184-core B510-based system with 2.7 GHz processors at CEA, France, a performance of 1.359 Pflop/s was measured in solving a dense linear system an unkown size. See [39].

3.1.1.2 The Bull bullx S6010/S6030 systems.

Machine type: Hybrid distributed-memory system.
Models: bullx S6010/S6030 system
Operating system: Linux, Windows Server 2008
Connection structure: Variable.
Compilers: Intel's Fortran 95, C(++).
Vendors information Web page: `www.bull.com//bullx/bullxS.html`
Year of introduction: 2010/2011.

System parameters:

Model	Blade S6010, S6030
Clock cycle	2.26, 2.4 or 2.66 GHz
Theor. peak performance	9.0, 11.6, 12.8 Gflop/s/core
	145–515 Gflop/s/3U drawer
Accelerator	NVIDIA Tesla S2050
Main Memory	$\leq$512 GB/node
No. of processors	Variable
Communication bandwidth	
Point-to-point	2.5 GB/s (QDR Infiniband)
Aggregate peak	Variable

Remarks:
Bull calls the S6010/S6030 building blocks for this system "Supernodes" because of the amount of memory that can be accomodated. A Bull-proprietary switch enables 4 4-processor nodes to work as one ccNUMA system with up to 160 cores (depending on what processor is chosen). In that respect there is much choice, because one choose from the 4- to 8-core Nehalem EX at 2.26 GHz to the 6- to 10-core Westmere EX processor running at either 2.4 or 2.66 GHz. This is also the reason for the wide variety of performances for a 3U drawer unit: the contents can range from 4 4-core Nehalem processors to 4 10-core Westmere EX processors.
The packaging of the S6010 is rather odd: The node is L-shaped and by flipping it over one can fit it on top of another S6010 node such that it fits in a 3U rack space. The S6030 has a height of 3U and contains the same components as two S6010s but in addition more PCIe slots: 2 PCIe Gen2 $\times$16 and 4 PCIe $\times$8 against 1 PCIe $\times$16/S6010. Furthermore it can house much more disk storage: 6 SATA disks against 1 in the S6010 and up to 8 SAS disks or SATA SSD units. Clearly, the S6010 it more targeted at the computational tasks, while the S630 also is well-equiped for server tasks.
The nodes are connected via QDR Infiniband at 2.5 GB/s in a topology that can be chosen by the customer.

Measured Performances:
The TERA 100 system of CEA in France is a S60*x*0-based machine containing 8-core Nehalem EX processors. In [39] a performance was reported for this 138368-core system of 1.05 Pflop/s out of 1.254 Pflop/s on a linear system of size 4926336, amounting to an efficiency of 83.7%.

3.1.2 The Cray Inc. XC30

Machine type: Distributed-memory multi-processor.
Models: XC30.
Operating system: CNL, Cray's microkernel Unix (for the compute nodes).
bf Connection structure: Dragonfly network.
Compilers: Fortran 95, C, C++, UPC, Co-Array Fortran, Chapel.
Vendors information Web page: `www.cray.com/Products/XC/resources.aspx`

Year of introduction: 2012.

System parameters:

Model	Cray XC30
Clock cycle	$\leq$ 2.7 GHz
Theor. peak performance	
Per processor	8×21.6 Gflop/s
Per Cabinet	92 Tflop/s (CPU-only)
Memory	
Per Cabinet	$\leq$ 24.6 TB
No. of processors	
Per Cabinet	384
Max. Configuration	$>$ 50000
Communication bandwidth	
Point-to-point	$\leq$10 GB/s

Remarks:
The XC30 is the commercial version of the Cascade system that Cray developed for the DARPA HPCS program. It is the new top-of-the-line system in Cray's product portfolio and as such replaces the Cray XE6 system (which still is marketed, see below). The structure of the machine is quite different from the XE6 in that it both uses different processors and a different interconnect. The only part that is largely the same is the software stack.
To start with the processor: in the XC30 Intel Ivy Bridge processors are used instead of AMD Opteron processors. Apart from generally being somewhat higher performing than the AMD Interlagos processors, the main and decisive factor is the presence of a PCIe Gen3 ×16 interface in the Intel processors that enables a direct connection to the new Aries interconnect chip at a speed of 16 GB/s. The second and more important difference is the Aries interconnect chip itself. The Aries is based on a 48-port YARC chip that has a 500 GB/s internal aggregate bandwidth. The Gemini interconnect chip in the XE6 was the first generation of the YARC chip with a considerably lower total bandwidth (160 GB/s). More important, where the Gemini implements a 3-D torus, the Aries chip implements a so-called dragonfly topology [21]. A dragonfly topology is hierarchical in nature and is made up of 3 levels: router, group, and system. An important condition is that the interconnectivity within the router and between routers is high[1] to enable routing with a small number of hops. In the Aries router chip this condition is fulfilled by dedicating 15 links to a backplane, a so-called Rank-1 network that interconnects the blades in a cabinet without cables and also 15 links to a neighbouring cabinet with a Rank-2 network using copper cables. In this way two adjacent cabinets form a group. Another 10 links, using fibre optic cables connect the groups in a Rank-3 network system-wide. In this way a message between any two nodes in a group requires a maximum of two hops. The number of hops between groups depends on the connection between these groups. For the Cray XC30 it is an all-to-all connection which means only one hop between groups is added. The lowest bandwidth is that between groups at nominally 10 GB/s. In practical experiments with MPI an actual bandwidth of 9.5 GB/s was observed.
As mentioned, the Aries chip has 4 PCIe Gen3 ×16 links which means that they can connect to any PCIe Gen3 enabled device. In the case of the XC30 it is possible to interchange a standard Sandy Bridge chip for an accelerator chip: either an NVIDIA K20X GPU or an Intel Phi.
Besides the compute nodes there are I/O nodes that can be configured as interactive nodes or nodes that connect to background storage. The I/O nodes only contain one Intel processor and run a full Linux operating system. The compute nodes run a special Linux variant, called Extreme Scalability Mode, that greatly reduces the variability of the runtimes of similar tasks. This so-called OS-jitter can be quite detrimental to overall performance, especially for very large machine configurations. Cray claims a scalability of over a million cores for this streamlined OS. In the IBM BlueGene systems (see 3.1.8) a similar separation between compute and service nodes is employed.
Cray offers the usual Intel compiler set and its MKL numerical library but also its own scientific library and

[1]The number of incoming/outgoing ports of a router chip is called the radix of that router. Hence, the 48-port YARC chip can be regarded as having a high radix.

compilers supporting the PGAS languages UPC, Co-Array Fortran (CAF), and Chapel, the PGAS language that was developed by Cray for DARPA's HPCS program. Besides Cray's MPI implementation also its `shmem` library for one-sided communication is available.
A one-cabinet or one-group midsize XC30 is planned to come to market somewhere in 2013. As yet Cray has not given details of such a system. For the present XC30 system no maximum configuration is given. It is only stated to scale to more than 50,000 nodes.

Measured Performances:
In the TOP500 list of June 2013 a speed of 627 out of 745.5 Tflop/s was reported for the Swiss Piz Daint system, solving a linear system of order 35480 with an efficiency of 84%.

3.1.3 The Cray Inc. XE6/XE6m

Machine type: Distributed-memory multi-processor.
Models: XE6, XE6m.
Operating system: CNL, Cray's microkernel Unix (for the compute nodes).
bf Connection structure: XE6:3-D Torus, XE6m: 2-D torus.
Compilers: Fortran 95, C, C++, UPC, Co-Array Fortran.
Vendors information Web page: `www.cray.com/Products/XE/Systems/XE6.aspx`, `www.cray.com/Products/XE/Systems/XE6m.aspx`
Year of introduction: XE6: 2010, XE6m: 2011.

System parameters:

Model	Cray XE6m	Cray XE6
Clock cycle	2.0–2.8 GHz	2.0–2.8 GHz
Theor. peak performance		
Per processor	105.2 Gflop/s	105.2 Gflop/s
Per Cabinet	12.2–20.2 Tflop/s	12.2–20.2 Tflop/s
Max. Configuration	121.2 Tflop/s	—
Memory		
Per Cabinet	$\leq$ 30.1 TB	$\leq$ 30.1 TB
Max. Configuration	$\leq$ 184 TB	—
No. of processors		
Per Cabinet	192	192
Max. Configuration	1152	—
Communication bandwidth		
Point-to-point	$\leq$8.3 GB/s	$\leq$8.3 GB/s
Bisectional/cabinet	2.39 TB/s	2.39 TB/s

Remarks:
Until the introduction of the XC30 (see 3.1.2) the structure of the Cray machines was very stable over the years: a 3-D torus that connects the processor nodes. The XE6 is the last in this line. The nodes as well as the routers have made it through quite a development, however. From the earliest XT-systems with a single AMD core to the XE6 with the 16-core Interlagos processors in the XE6 node. Also the interconnect routers have gone through an evolution from the first SeaStar router to the Gemini router, perhaps the most distinguishing factor of the system. The Gemini is based on the 48-port YARC chip that boasts a 160 GB/s internal aggregate bandwidth. Since the Gemini Network Interface Card (NIC) operates at 650 MHz and the NIC is able to transfer 64 B every 5 cycles, the bandwidth per direction is 8.3 GB/s while the latency varies from 0.7–1.4 μs depending on the type of transfer [1]. In practice bandwidths of over 6 GB/s per direction were measured, compatible with the claim in Cray's brochure of an injection bandwidth of over 20 GB/s/node. A nice feature of the Gemini router is that it supports adaptive routing, even on a packet to packet basis. As the 3-D torus topology is vulnerable with regard to link failures this makes the network

much more robust.
Besides the compute nodes there are I/O nodes that can be configured as interactive nodes or nodes that connect to background storage. The I/O nodes only contain one Opteron processor and run a full Linux operating system. The compute nodes run a special Linux variant, called Extreme Scalability Mode, that greatly reduces the variability of the runtimes of similar tasks. This so-called OS-jitter can be quite detrimental to overall performance, especially for very large machine configurations. Cray claims a scalability of over a million cores for this streamlined OS. In the IBM BlueGene systems (see 3.1.8) a similar separation between compute and service nodes is employed.
Cray offers the usual compilers and AMD's ACML numerical library but also its own scientific library and compilers. Also the PGAS languages UPC and Co-Array Fortran (CAF) are supported. Besides Cray's MPI implementation also its `shmem` library for one-sided communication is available.
In 2011 the XE6m model has become available, where "m" stands for midrange. The XE6m has at most 6 cabinets with a peak speed of just over 120 Tflop/s. A further rationalisation is that not a 3-D but a 2-D torus is employed as the interconnection network. For the XE6 model itself no maximum configuration is given. The Cray documentation suggests that more than a million cores would be possible.

Measured Performances:
In [39] a speed of 1.11 Pflop/s was reported for a 142272-core XE6, based on 2.4 GHz Instanbul processors for the solution of a linear system of unspecified size. The efficiency was 81.3%.

3.1.4 The Cray Inc. XK7

Machine type: Distributed-memory multi-processor.
Models: XK7.
Operating system: CNL, Cray's microkernel Unix (for the compute nodes).
Connection structure: 3-D Torus.
Compilers: Fortran 95, C, C++, UPC, Co-Array Fortran, CUDA, OpenCL.
Vendors information Web page: `www.cray.com/Products/XK/XK7.aspx`
Year of introduction: 2012.

System parameters:

Model	Cray XK7
Clock cycle CPU	2.0–2.8 GHz
Clock cycle GPU	0.732 GHz
Theor. peak performance	
Per Cabinet	$>$ 100 Tflop/s
Max. Configuration	—
Memory	
Per Cabinet	$\leq$ 15.4 TB
Max. Configuration	—
No. of processors	
Per Cabinet	96 CPUs; 96 GPUs
Max. Configuration	—
Communication bandwidth	
Point-to-point	$\leq$8.3 GB/s
Bisectional/cabinet	2.39 TB/s

The XK7 machine has the structure of the Cray XE6 (see above) but in a node two of the Opteron processors have been replaced by NVIDIA GPUs. For appropriate applications this will boost the performance more than 5-fold. Because the application speed is so dependent on the application and the the amount of data to be shipped back and forth between the GPU's memory and the system memory no sensible speed estimate can begiven, except that for a cabinet the performance may well exceed 100 Tflop/s when the application is

right.
Apart from the usual software stack for Cray products of coarse CUDA and OpenCL are supported for the GPUs and also OpenACC, the OpenMP-like directive/pragma-based library and runtime that should make it easier for the general programmer to take advantage of the GPUs.

Measured Performances:
In [39] a speed of 17.59 Pflop/s was reported on the 560640-core XK7 Titan machine of ONRL, for the solution of a linear system of unspecified size. The efficiency was 64.9%; surprisingly high for a GPU-based system.

3.1.5 The Eurotech Aurora.

Like Bull, Eurotech markets HPC systems that in some ways enhance the concept of the standard cluster. We therefore dedicate an entry for these systems in this report.

3.1.5.1 The Eurotech AuroraHPC 10-10

Machine type: Distributed-memory multi-processor.
Models: AuroraHPC 10-10.
Operating system: Linux.
Connection structure: 3D torus + Tree network.
Compilers: Fortran 90, OpenMP, C, C++.
Vendors information Web page: `http://www.eurotech.com/en/hpc/hpc+solutions/aurora+hpc+systems`
Year of introduction: 2012.

System parameters:

Model	AuroraHPC 10-10
Clock cycle	3.1 GHz
Theor. peak performance	
Per core (64-bits)	24.8 Gflop/s
Per 8-chassis rack	≈100 Tflop/s
Maximal	—
Memory/node	≤ 32 GB
Memory/maximal	—
Communication bandwidth	
Point-to-point (theor.)	2.5 GB/s
Aggregate per node	60 GB/s

Remarks:
We only discuss the latest model, the AuroraHPC 10-10 here as the earlier model has the same macro-architecture but less powerful Nehalem EP instead of Sandy Bridge processors.
The Aurora system has most characteristics of the average cluster but there are a number of (optional) distinguishing factors that warrant its description in this report. For instance one can choose for SSD storage instead of spinning disks. The liquid cooling on a per node basis also contributes to the energy efficiency as no power is used for memory that is not active.
The interconnect infrastructure is also out of the ordinary in comparison with the standard cluster. It has a QDR Infiniband network in common with other clusters but on can opt for an additional 3-D torus network. Eurotech calls this its Unified Network Architecture with a latency of about 1 μs and a point-to-point bandwidth of 2.5 GB/s. The network processor is in fact a rather large Altera Stratix IV FPGA that provides the possibility of reconfiguration of the network and hardware synchonisation of MPI primitives.
An Aurora node consists of two 8-core E5 Sandy Bridge processors, each with their associated DDR3 memory. So, 32 GB at a maximum per node. Via the Tylersburg bridge the network processor is connected through PCIe Gen2 to the network processor, containing the Stratix FPGA, that drives the 3-D network

and a Mellanox ConnectX Infiniband HCA.
In principle the FPGA has sufficient capacity also to be used as a computational accelerator but Eurotech has no fixed plans yet to offer it as such. Eurotech does not give a maximum configuration for the Aurora but the brochures suggest that it considers building a Petaflop system (10 racks) is easily possible.
Although the Aurora documentation is not very clear on the software that is available it is evident that Linux is the OS and the usual Intel compiler suite is available. The MPI version is optimised for the architecture but system-agnostic MPI versions can also be used.

Measured Performances:
At the time of writing this report no official performance figures from the AuroraHPC 10-10 are available.

3.1.5.2 The Eurotech Aurora Tigon

Machine type: Distributed-memory multi-processor.
Models: Aurora Tigon.
Operating system: Linux.
Connection structure: 3D torus + Tree network.
Compilers: Fortran 90, OpenMP, C, C++.
Vendors information Web page: `http://www.eurotech.com/en/hpc/hpc+solutions/↩ aurora+hpc+systems/Aurora+Tigon`
Year of introduction: 2012.

System parameters:

Model	Aurora Tigon
Clock cycle CPU	3.1 GHz
Accelerator	NVIDIA Kepler K20X Xeon Phi
Theor. peak performance	
Per core (64-bits)	24.8 Gflop/s (CPU)
Per 8-chassis rack	≤350 Tflop/s
Maximal	—
Memory/node	≤ 32 GB
Memory/maximal	—
Communication bandwidth	
Point-to-point (theor.)	2.5 GB/s
Aggregate per node	60 GB/s

Remarks:
Like in the Bull systems Eurotech markets an accelerator-enhanced sytem called the Tigon. In the Tigon 2 of the standard CPUs in a node can be replaced by either NVIDIA Kepler K20Xs or by Intel Xeon Phis (or any mix thereof). This should lead to a peak performance that is about 3.5 higher than of a CPU-only rack: ≈ 350 Tflop/s at a power consumption of about 100 kW/rack.

Measured Performances:
At the time of writing this report no official performance figures from the Aurora Tigon are available.

3.1.6 The Fujitsu PRIMEHPC FX10.

Machine type: RISC-based distributed-memory multi-processor.
Models: PRIMEHPC FX10.
Operating system: Solaris (Sun's Unix variant).
Connection structure: 6-D torus.
Compilers: Fortran 90, OpenMP, C, C++.

Vendors information Web page: http://www.fujitsu.com/global/services/solutions/tc/hpc/↩ products/primehpc/
Year of introduction: 2011.

System parameters:

Model	FX10
Clock cycle	1.848 GHz
Theor. peak performance	
Per core (64-bits)	14.8 Gflop/s
Per processor	236.5 Gflop/s
Maximal	23.2 Pflop/s
Memory/node	$\leq$ 64 GB
Memory/maximal	6 PB
Communication bandwidth	
Point-to-point	5 GB/s/direction
Aggregate	—

Remarks:
A few months after the introduction of the K-computer a commercial version was presented on the market. The structure of the system was the same as that of the K-computer but the processor was slightly improved (see 2.8.5). The 8-core processor runs at 2 GHz which amounts to a peak performance of 128 Gflop/s because every core delivers 8 floating-point results/cycle.
The FX10 has the same interconnect as the K-computer: a 6-D torus. Apart from a very high bandwidth of 5 GB/s in each direction, the extra dimensions (the user experiences only 3 of them) allow for a high resiliency for failures and an easy way of rerouting in case of contention.
The system can be made very large: a maximum of 98,304 nodes can be configured for a peak performance of 23.2 Pflop/s. Like in the Cray and IBM BlueGene systems the compute nodes are not bothered by system tasks. These tasks are diverted to dedicated I/O nodes. This greatly reduces the OS-jitter in the system, ultimately improving application scalability for large amount of nodes.
Where most HPC vendors (except IBM) offer Lustre as their HPC file system, Fujitsu has its own brand: FEFS. Fujitsu also has its own suite of compilers, a scientific library, MPI, OpenMP, and a proprietary Fortran extension XPFortran that, like OpenMP is directive-based. In addition, Fujitsu has its own tuner/debugger to help in the development of large-scale applications.

Measured Performances:
When one would regard the K-computer as an instantiation of an FX10 system in [39] the K-computer enters with a speed of 10.51 Pflop/s in solving a linear system of unknown size with the very high efficiency of 93.1%.

3.1.7 The Hitachi SR16000.

Machine type: RISC-based distributed-memory multi-processor.
Models: SR16000 XM1, M1, and VM1.
Operating system: AIX (IBM's Unix variant).
Connection structure: Multi-dimensional crossbar (see remarks).
Compilers: Fortran 77, Fortran 95, Parallel Fortran, C, C++.
Vendors information Web page:
www.hitachi.co.jp/Prod/comp/hpc/SR_series/sr16000/index.html (only in Japanese).
Year of introduction: 2010 (XM1), 2011 (M1,VM1).

System parameters:

Model	SR16000 XN1	SR16000 M1	SR16000 VM1
Clock cycle	4.76 GHz	3.70 GHz	5.0 GHz
Theor. peak performance			
Per Proc. (64-bits)	1049.6 Gflop/s	818.4 Gflop/s	—
Maximal	537 Tflop/s	502 Tflop/s	8192 Gflop/s
Memory/node	≤256 GB	≤256 GB	≤8 TB
Memory/maximal	131 TB	131 TB	—
No. of processors	1–512	32–512	1
Communication bandwidth			
Point-to-point	16 GB/s	16 GB/s	16 GB/s

Remarks:
The SR16000 is the fourth generation of distributed-memory parallel systems of Hitachi. It replaces its predecessor, the SR11000 (see 4). We discuss here the latest models, the SR16000 XM1, M1 and VM1. All three systems are water cooled. All processors used in the SR16000 models are IBM's POWER7s but the packaging is different from what is used in IBM's p775 systems (see 3.1.9).
Unlike in their predecessor, the SR11000, the processors in all three models are fit for Hitachi's Pseudo Vector Processing, a technique that enables the processing of very long vectors without the detrimental effects that normally occur when out-of-cache data access is required.
The peak performance per basic processor, or IP, can be attained with 2 simultaneous multiply/add instructions resulting in a speed of 17.8 Gflop/s on the SR16000 in the M1 (and 20 Gflop/s in the VM1). However, 32 basic processors in the M1 and 64 processors in the VM1 are coupled to form one processing node all addressing a common part of the memory. For the user this node is the basic computing entity with a peak speed of 844.8 anf 980.8 Tflop/s, resp. 1280 Gflop/s. Hitachi refers to this node configuration as COMPAS, Co-operative Micro-Processors in single Address Space. In fact this is a kind of SMP clustering as discussed in sections 2.1 and 2.6. In contrast to the preceding SR8000 the COMPAS processors do not contain an SP anymore, a system processor that performed system tasks, managed communication with other nodes and a range of I/O devices. These tasks are now performed by the processors in the SMP nodes themselves. The structure of the XM1 model is identical to that of the M1 model, except that POWER7 processors are employed at a clock frequency of 3.83 GHz instead of 4.44 GHz.
The SR16000 has a multi-dimensional crossbar with a single-directional link speed of 4–16 GB/s. For this QDR InfiniBand is used in a torus topology. From 4–8 nodes the cross-section of the network is 1 hop. For configurations of 16–64 it is 2 hops and from 128-node systems on it is 3 hops.
Like in some other systems as the Cray XE6 (3.1.3), and the late AlphaServer SC, (4) and NEC Cenju-4, one is able to directly access the memories of remote processors. Together with the fast hardware-based barrier synchronisation this should allow for writing distributed programs with very low parallelisation overhead.
The usual communication libraries like PVM and MPI are provided. In case one uses MPI it is possible to access individual IPs within the nodes. Furthermore, in one node it is possible to use OpenMP on individual IPs. Mostly this is less efficient than using the automatic parallelisation as done by Hitachi's compiler but in case one offers coarser grained task parallelism via OpenMP a performance gain can be attained. Hitachi provides its own numerical libraries to solve dense and sparse linear systems, FFTs, etc. As yet it is not known whether third party numerical libraries like NAG is available.

Note: The SR16000 models are not sold in Europe as this is judged to be of insufficient economical interest by Hitachi.

Measured Performances:
In [39] a SR16000 XM1 system is listed. A 10340-core system attained a speed of 253 Tflop/s on the LinPACK benchmark on a system of unspecified size. The efficiency was 80.0%.

3.1.8 The IBM BlueGene/Q.

Machine type: RISC-based distributed-memory multi-processor.
Models: IBM BlueGene/Q.

Operating system: Linux.
Connection structure: 5-D torus.
Compilers: XL Fortran 90, XL C, C++.
Vendors information Web page:
`www-1.ibm.com/servers/deepcomputing/bluegene/.`
Year of introduction: 2012.

System parameters:

Model	BlueGene/Q
Clock cycle	1.6 GHz
Theor. peak performance	
Per Proc. (64-bits)	204.8 Gflop/s
Maximal	—
Memory/card	16 GB
Memory/maximal	—
No. of processors	—
Communication bandwidth	
Point-to-point	2 GB/s

Remarks:
The BlueGene Q is the last generation sofar in the BlueGene family. The performance per processor has increased hugely with respect to its predecessor, the BleuGene/P: from 3.4 to 204.8 Gflop/s. This is due to several factors: the clock frequency almost doubled and also the floating-point output doubled because of the 4 floating-point units/core capable of turning out 4 fused multiply-add results per cycle. Furthermore there are 16 instead of 4 cores per processor. However, the amount of memory per core has not increased: where in the P model 2 GB on a card feeds 8 cores (there were two processors on a card), in the Q model 32 cores draw on 16 GB of memory (again with 2 processors on a card).
Another deviation from the earlier models is the interconnect. It is now a 5-D torus with a link speed of 2 GB/s while the tree network present in the former L model and P models has disappeared. The two extra dimensions will compensate for this loss while the resiliency of the network is increased: a 3-D torus is rather vulnerable in terms of link failures. A processor has 11 links of which 10 are necessary for the 5-D torus directions and one spare link that can be used for other purposes or in case of failure of another link. This is all the more critical for the very large systems that are envisioned to be built from the components. Although there is no official maximum size given for BlueGene/Q systems, the 20 Pflop/s Sequoia system was commissioned and delivered for Lawrence Livermore Laboratory and a 10 Pflop/s system for Argonne National Lab. Like with the earlier models this can be achieved because of the high density. A BlueGene/Q node card houses 32 2-processor compute cards while 16 node cards are fitted onto a midplane. A rack contains two of these populated midplanes which therefore delivers almost 420 Tflop/s. Consequently, tens of racks are needed to build systems of such sizes and reliability features become extremely important.

In the BlueGene/Q the compute nodes run a reduced-kernel type of Linux to reduce the OS-jitter that normally occurs when very many nodes are involved in computation. Interface nodes for interaction with the users and providing I/O services run a full version of the operating system. In the BlueGene/Q jitter reduction is also achieved by the 17^{th} core that is dedicated to OS tasks (see 2.8.3).

Measured Performances:
In [39] a speed of 17.17 Pflop/s was measured for the BlueGene/Q Sequoia system using 1,572,864 cores to solve a dense linear system with an efficiency of 85.3%.

3.1.9 The IBM eServer p775.

Machine type: RISC-based distributed-memory multi-processor.
Models: IBM eServer p775.
Operating system: AIX (IBM's Unix variant), Linux (Red Hat EL).
Connection structure: Variable (see remarks).

Compilers: XL Fortran (Fortran 90), (HPF), XL C, C++.
Vendors information Web page:
`http://www-03.ibm.com/systems/power/hardware/775/`
Year of introduction: 2011.

System parameters:

Model	eServer p775
Clock cycle	3.83 GHz
Performance	
Per Proc. (8 cores)	245.1 Gflop/s
Per node (32 proc.s)	7.84 Tflop/s
Per 12-node rack	94.1 Tflop/s
Maximal	16.05 Pflop/s
Memory	
Memory/node	$\leq$2 TB
Memory maximal	$\leq$4.096 PB
Communication bandwidth	
Node-to-node (see remarks)	—

Remarks:
There is a multitude of high end servers in the eServer p-series. However, IBM singles out the POWER7 based p775 model specifically for HPC. The eServer p775 is the successor of the earlier p575 POWER6-based systems. It retains much of the macro structure of this system: multi-CPU nodes are connected within a frame either by a dedicated switch or by other means, like switched Ethernet. The structure of the nodes, however, has changed considerably, see 2.8.2. Four octo-core POWER7 processors are housed in a Quad-Chip Module (QCM) while eight of these constitute a p775 node. So, 256 cores make up a node. Within a node the four QCMs are directly connected to each other by copper wires.
In contrast to its earlier p575 clusters, IBM does now provide a proprietary interconnect for the system based on in-house optical technology. Each node contains 224 optical transceivers that each constitute 12 1.25 GB/s send- and recieve lanes. Ten out of these 12 lanes are used for normal communication where the two other lanes can act as a fall-back when one of the regular links fails. The number of links/node is sufficient to directly connect to 127 others and to achieve the connection to the maximal configuration of 2048 nodes a second level of interconnection can be realised through hub modules. Depending on the relative position of the nodes the bandwidth varies: 336 GB/s to 7 other QCMs, 320 GB/s to remote nodes, and 240 GB/s from local to remote nodes. Note that these are the aggregate bandwidth from all lanes together. Like the former p575 the p775 system is accessed through a front-end control workstation that also monitors system failures. Failing nodes can be taken off line and exchanged without interrupting service. Because of the very dense packaging of the units that house the POWER7 processors are water cooled.
Applications can be run using PVM or MPI. IBM used to support High Performance Fortran, both a proprietary version and a compiler from the Portland Group. It is not clear whether this is still the case. IBM uses its own PVM version from which the data format converter XDR has been stripped. This results in a lower overhead at the cost of generality. Also the MPI implementation, MPI-F, is optimised for the p775-based systems. As the nodes are in effect shared-memory SMP systems, within the nodes OpenMP can be employed for shared-memory parallelism and it can be freely mixed with MPI if needed. In addition to its own AIX OS IBM also supports one Linux distributions: the professional version of RedHat Linux is available for the p775 series.

Measured Performances:
In [39] a speed of 1.52 Pflop/s was reported for a 63,360-core system. The efficiency for solving the dense linear system was 77.9%.

3.1.10 The NEC SX-9 series.

Machine type: Shared-memory multi-vectorprocessor.
Models: SX-9B SX-9A, SX-9xMy, $y = 2, \ldots, 512$.
Operating system: Super-UX (Unix variant based on BSD V.4.3 Unix).
Compilers: Fortran 90, HPF, ANSI C, C++.
Vendors information Web page: `http://www.nec.com/en/de/en/prod/solutions/hpc-solutions/sx-9/index.html`
Year of introduction: 2007.

System parameters:

Model	SX-9B	SX-9A	SX-9xMy
Clock cycle	3.2 GHz	3.2 GHz	3.2 GHz
Theor. peak performance			
Per Proc. (64-bits)	102.4 Gflop/s	102.4 Gflop/s	102.4 Gflop/s
Maximal			
Single frame:	819.2 Gflop/s	1.6 Tflop/s	—
Multi frame:	—	—	838.9 Tflop/s
Main Memory, DDR2-SDRAM	256–512 GB	512–1024 GB	≤512 TB
Main Memory, FCRAM	128–256 GB	256–512 GB	≤256 TB
No. of processors	4–8	8–16	32–8192

Remarks:
The NEC SX-9 is a technology shrunken version of its predecessor the SX-8 (see 4). As a result the clock cycle has increased from 2.0 to 3.2 GHz, the density of processors/frame has doubled and the power consumption almost halved. The structure of the CPUs, however, has stayed the same. The SX-9 series is basically offered in three models as displayed in the table above. All models are based on the same processor, an 8-way replicated vector processor where each set of vector pipes contains a logical, mask, add/shift, multiply, and division pipe (see section 2.2 for an explanation of these components). As multiplication and addition can be chained (but not division) and two of each are present, the peak performance of a pipe set at 3.2 GHz is 12.8 Gflop/s. Because of the 8-way replication a single CPU can deliver a peak performance of 102.4 Gflop/s. The official NEC documentation quotes higher peak performances because the peak performance of the scalar processor (rated at 6.4 Gflop/s, see below) is added to the peak performance of the vector processor to which it belongs. We do not follow this practice as a full utilisation of the scalar processor along with the vector processor in reality will be next to non-existent. The scalar processor that is 2-way super scalar and at 3.2 GHz has a theoretical peak of 6.4 Gflop/s. The peak bandwidth per CPU is 160 B/cycle. This is sufficient to ship 20 8-byte operands back or forth, enough to feed 5 operands every 2 cycles to each of the replicated pipe sets.
Unlike from what one would expect from the naming the SX-9B is the simpler configuration of the two single-frame systems: it can be had with 4–8 processors but is in virtually all other respects equal to the larger SX-9A that can house 8–16 processors, 16 being the maximum number of processors fitting in a frame. There is one difference connected to the maximal amount of memory per frame: NEC now offers the interesting choice between the usual DDR2-SDRAM or FCRAM (Fast Cycle Memory. The latter type of memory can a factor of 2–3 faster than the former type of memory. However, because of the more complex structure of the memory, the density is about two times lower. Hence that in the system parameters table, the entries for FCRAM are two times lower than for SDRAM. For the very memory-hungry applications that are usually run on vector-type systems, the availability of FCRAM can be beneficial for quite some of these applications.
In a single frame of the SX-9A models fit up to 16 CPUs. Internally the CPUs in the frame are connected by a 1-stage crossbar with the same bandwidth as that of a single CPU system: 512 GB/s/port. The fully configured frame can therefore attain a peak speed of 1.6 Tflop/s.
In addition, there are multi-frame models (SX-9xMy) where $x = 32, \ldots, 8192$ is the total number of CPUs and $y = 2, \ldots, 512$ is the number of frames coupling the single-frame systems into a larger system. To couple the SX-9 frames NEC provides a full crossbar, the so-called IXS crossbar to connect the various frames together at a speed of 128 GB/s for point-to-point unidirectional out-of-frame communication. When connected by

the IXS crossbar, the total multi-frame system is globally addressable, turning the system into a NUMA system. However, for performance reasons it is advised to use the system in distributed memory mode with MPI.
For distributed computing there is an HPF compiler and for message passing optimised MPI (MPI/SX) is available. In addition for shared memory parallelism, OpenMP is available.

Measured Performances:
In [39] the SX-9/E based Earth Simulator shows a performance of 122.4 Tflop/s for a linear system of order N = 1,556,480, attaining an efficiency of 93%.

3.1.11 The SGI Altix UV series.

Machine type: RISC-based ccNUMA system.
Models: Altix UV 2000.
Operating system: Linux (SuSE SLES9/10, RedHat EL4/5) + extensions.
Connection structure: Unspecified.
Compilers: Fortran 95, C, C++, CUDA, OpenCL.
Vendors information Web page: `www.sgi.com/products/servers/altix/uv/`.
Year of introduction: 2012.

System parameters:

Model	Altix UV 2000
Clock cycle	2.0–2.9 GHz
Theor. peak performance	
Per core (64-bits)	16–23.2 Gflop/s
Maximal	47.5 Tflop/s (CPU only)
Memory	
Memory per blade	$\leq$128 GB
Memory maximal	8.2 TB
No. of cores	4096
Communication bandwidth	
Point-to-point	6.7 GB/s/direction

Remarks:
The Altix UV 2 series is the latest (^{6th})generation of ccNUMA shared-memory systems made by SGI and the second in the Altix UV series. Apart from the UV 2000 there is also a 4-socket UV 20, but this is too small to be discussed here. The processor used is the E5-4600 (Sandy Bridge) (see 2.8.4). The distinguising factor of the UV systems is their distributed shared memory that can be up to 8.2 PB. Every blade can carry up to 128 GB of memory that is shared in a ccNUMA fashion through hubs and the 6^{6th} generation of SGI's proprietary NumaLink6. A very high-speed interconnect with a point-to-point bandwidth of 6.7 GB/s per direction; doubled with respect to the former NumaLink5.
Like Bull, Cray, and Eurotech SGI offers the possibility to exchange two CPUs on a blade by either nVIDIA's Tesla K20X GPUs (see 2.9.1.2) or by Xeon Phi accelerators (see 2.9.2.1).
A UV blade contains 4 8-core E5 processors, connected to each other by two QPI links while each processor also connects to the Northbridge chipset for I/O, etc. Lastly all processors are connected via QPI links to the UV hub that takes care of the communication with the rest of the system. The bandwidth from the hub to the processors is 25.6 GB/s while the 4 ports for outside communication are approximately 13.5 GB/s each.
The hub does much more than acting as a simple router. It ensures cache coherency in the dirstibuted shared memory. There is an Active Memory Unit that supports atomic memory operations and takes care of thread synchonisation. The Global Register Unit (GRU) within the hub also extends the x86 addressing mode (44-bit physical, 48-virtual) to 53, resp. 60 bits to accomodate the potentially very large global address space of the system. In addition it houses an external TLB cache that enables large memory page support. Furthermore, it can perform asynchronous block copy operations akin to the block transfer unit in Cray's Gemini and Aries routers. In addition the GRU is accomodates scatter/gather operations which greatly can

speed up cache-unfriendly sparse algorithms. Lastly, MPI operations can be off-loaded from the CPU and barriers and synchonisation for reduction operations are taken care of in the MPI Offload Engine (MOE). The UV systems come with the usual Intel stack of compilers and tools. To take full advantage of the facilities of the hub it is advised to use SGI's MPI version based on its Message Passing Toolkit although independent implementations, like OpenMPI, also will work.

Measured performances:
By contrast to the SGI ICE X cluster systems, there are no measured performance data known to the author for this system.

4 Systems disappeared from the list

As already stated in the introduction the list of systems is not complete. On one hand this is caused by the sheer number of systems that are presented to the market and are often very similar to systems described above (for instance, the Volvox system not listed was very similar but not equivalent to the former C-DAC system and there are numerous other examples). On the other hand, there are many systems that are still in operation around the world, often in considerable quantities that for other reasons are excluded. The most important reasons are:

- The system is not marketed anymore. This is generally for one of two reasons:
  1. The manufacturer is out of business.
  2. The manufacturer has replaced the system by a newer model of the same type or even of a different type.
- The system has become technologically obsolete in comparison to others of the same type. Therefore, listing them is not sensible anymore.

Below we present a table of systems that fall into one of the categories mentioned above. We think this may have some sense to those who come across machines that are still around but are not the latest in their fields. It may be interesting at least to have an indication how such systems compare to the newest ones and to place them in context.
It is good to realise that although systems have disappeared from the section above they still may exist and are actually sold. However, their removal stems in such cases mainly from the fact that they are not serious candidates for high-performance computing anymore.
The table is, again, not complete and admittedly somewhat arbitrary. Furthermore, we have reduced the contents of this section (in comparison to the former version) to systems that very probably are still actually in use in this printed (PostScript) version.
A more comprehensive, "full" version is available in the web version of this document at:

`www.hpcresearch.nl/euroben/Overview/web12/gone.html`

This full information may be of interest for a subset of readers that want to know about the historical development of supercomputing over the last 20 years.
The data are in a highly condensed form: the system name, system type, theoretical maximum performance of a fully configured system, and the reason for their disappearance is given. The arbitrariness lies partly in the decision which systems are still sufficiently of interest to include and which are not.
We include also both the year of introduction and the year of exit of the systems when they were readily accessible. These time-spans could give a hint of the dynamics that governs this very dynamical branch of the computer industry.

4.1 Recently disappeared systems

The set of HPC systems included in this report has proven to be remarkably stable this year. We only remove one system from the list and that is for another reason than that it was really obsolete (see below). All other systems are still available, be it that some of them have been upgraded with newer processors. The only system we remove from the list is discussed below.

4.1.1 Cray XMT

This system has been replaced by the Cray uRIKA system. It is essentially the same as the former Cray XMT, be it that the proprietary Threadstorm processor is replaced by the Threadstorm+ processor with

the same clock cycle (500 MHz) but with a few improvements. As the uRIKA system now is marketed completely for the High Performance Data Analysis (HPA), we do not include it in this report anymore because it is primarily directed to HPC.

4.2 Disappeared machines

Machine: Avalon A12.
Year of introduction: 1996.
Year of exit: 2000.
Type: RISC-based distributed memory multi-processor, max. 1680 processors.
Theoretical Peak performance: 1.3 Tflop/s.
Reason for disappearance: Avalon is not in business anymore.

Machine: Cambridge Parallel Processing DAP Gamma.
Year of introduction: 1986.
Year of exit: 1995.
Type: Distributed-memory processor array system, max. 4096 processors.
Theoretical Peak performance: 1.6 Gflop/s (32-bit).
Reason for disappearance: replaced by newer Gamma II Plus series (4.2).

Machine: Cambridge Parallel Processing DAP Gamma II Plus.
Year of introduction: 1995.
Year of exit: 2003.
Type: Distributed-memory processor array system, max. 4096 processors.
Theoretical Peak performance: 2.4 Gflop/s (32-bit).
Reason for disappearance: system became too slow, even for its specialised tasks. (4.2).

Machine: C-DAC PARAM 9000/SS.
Year of introduction: 1995.
Year of exit: 1997.
Type: Distributed-memory RISC based system, max. 200 processors.
Theoretical Peak performance: 12.0 Gflop/s.
Reason for disappearance: replaced by newer OpenFrame series (see below).

Machine: C-DAC PARAM Openframe series.
Year of introduction: 1996.
Year of exit: 1999.
Type: Distributed-memory RISC based system, max. 1024 processors.
Theoretical Peak performance: Unspecified.
Reason for disappearance: The system is not actively marketed anymore by C-DAC.

Machine: C-DAC PARAM 10000 Openframe series.
Year of introduction: 2000.
Year of exit: 2002.
Type: Distributed-memory RISC based system, max. processors: Unspecified.
Theoretical Peak performance: Unspecified.
Reason for disappearance: The system is replace by the newer C-DAC PARAM Padma, see section below.

Machine: C-DAC PARAM Padma.
Year of introduction: 2003.
Year of exit: 2010.
Type: Distributed-memory RISC based system, max. processors: Unspecified.
Theoretical Peak performance: 992 Gflop/s.

Reason for disappearance: The system might be still in production but its performance has become too low to be relevant.

Machine: Cray T3E Classic.
Year of introduction: 1996.
Year of exit: 1997.
Type: Distributed-memory RISC based system, max. 2048 processors.
Theoretical Peak performance: 1228 Gflop/s.
Reason for disappearance: replaced Cray T3Es with faster clock. (4.2).

Machine: Cray MTA-2.
Year of introduction: 2001.
Year of exit: 2005.
Type: Distributed-memory multi-processor, max. 256 processors.
Theoretical Peak performance: 192 Gflop/s.
Reason for disappearance: The system is not actively marketed anymore by Cray.

Machine: Cray XMT.
Year of introduction: 2007.
Year of exit: 2012.
Type: Distributed-memory multi-processor, max. 8024 processors.
Theoretical Peak performance: 120 Tflop/s.
Reason for disappearance: The system is re-branded to the Cray uRIKA system that is entirely directed at High Performance Data Analysis.

Machine: Cray T3E 1350.
Year of introduction: 2000.
Year of exit: 2003.
Type: Distributed-memory RISC based system, max. 2048 processors.
Theoretical Peak performance: 2938 Gflop/s.
Reason for disappearance: Cray does not market the system anymore.

Machine: Cray J90.
Year of introduction: 1994.
Year of exit: 1998.
Type: Shared-memory vector-parallel, max. 32 processors.
Theoretical Peak performance: 6.4 Gflop/s.
Reason for disappearance: replaced by newer Cray SV1 (4.2).

Machine: Cray Y-MP T90.
Year of introduction: 1995.
Year of exit: 1998.
Type: Shared-memory vector-parallel, max. 32 processors.
Theoretical Peak performance: 58 Gflop/s.
Reason for disappearance: replaced by newer Cray SV1 (see below).

Machine: Cray SV-1(ex).
Year of introduction: 2000.
Year of exit: 2004.
Type: Shared-memory vector-parallel, max. 32 processors(per frame).
Theoretical Peak performance: 64 Gflop/s.
Reason for disappearance: replaced by newer Cray X1 (4.2).

Machine: Cray X1.
Year of introduction: 2000.
Year of exit: 2004.
Type: Shared-memory vector-parallel, max. 64 MSP processors.
Theoretical Peak performance: 819 Gflop/s.
Reason for disappearance: replaced by newer Cray X1E (4.2).

Machine: Cray X1E.
Year of introduction: 2004.
Year of exit: 2007.
Type: Shared-memory vector-parallel, max. 8192 MSP processors.
Theoretical Peak performance: 147.2 Tflop/s.
Reason for disappearance: replaced by newer Cray X2 (4.2).

Machine: Cray X2.
Year of introduction: 2007.
Year of exit: 2009.
Type: Shared-memory vector-parallel, max. 32576 MSP processors.
Theoretical Peak performance: 3.3 Pflop/s.
Reason for disappearance:Cray considers vector-based systems not economically viable anymore.

Machine: Cray XD1.
Year of introduction: 2004.
Year of exit: 2006.
Type: Distributed-memory multi-processor, max. 144 processors.
Theoretical Peak performance: 663 Gflop/s.
Reason for disappearance: relevant components will re-appear in Cray's heterogeneous systems in the near future.

Machine: Cray XR1.
Year of introduction: 2007.
Year of exit: 2009.
Type: FPGA-based accelerator blade.
Theoretical Peak performance: variable
Reason for disappearance: Cray deems it not economically viable anymore.

Machine: Cray XT5.
Year of introduction: 2008.
Year of exit: 2010.
Type: Distributed-memory multi-processor, maximum configuration not specified.
Theoretical Peak performance: 12 Tflop/s per cabinet.
Reason for disappearance: Superseded by the Crary XE6 (see 3.1.3.

Machine: Digital Equipment Corp. AlphaServer 8200 & 8400.
Year of introduction: —.
Year of exit: 1998.
Type: Distributed-memory RISC based systems, max. 6 processors (AlphaServer 8200) or 14 (AlphaServer 8400).
Theoretical Peak performance: 7.3 Gflop/s, resp. 17.2 Gflop/s.
Reason for disappearance: after take-over by the HP AlphaServer SC and presently by the newer HP Integrity Superdome.(4.2).

Machine: Fujitsu-Siemens M9000.
Year of introduction: 2008.

Year of exit: 2009.
Type: Distributed memory RISC based system, max. 128 processors.
Theoretical Peak performance: 2.58 Tflop/s.
Reason for disappearance: Fujitsu does not target this machine to the HPC market anymore. This rôle is taken over by the FX1 (see **??**).

Machine: Fujitsu AP3000.
Year of introduction: 1996.
Year of exit: 2003.
Type: Distributed memory RISC based system, max. 1024 processors.
Theoretical Peak performance: 614 Gflop/s.
Reason for disappearance: Fujitsu does not market the system anymore.

Machine: Fujitsu AP1000.
Year of introduction: 1991.
Year of exit: 1996.
Type: Distributed memory RISC based system, max. 1024 processors.
Theoretical Peak performance: 5 Gflop/s.
Reason for disappearance: replaced by the AP3000 systems (4.2).

Machine: Fujitsu VPP300/700 series.
Year of introduction: 1995/1996.
Year of exit: 1999.
Type: Distributed-memory multi-processor vectorprocessors, max. 256 processors.
Theoretical Peak performance: 614 Gflop/s.
Reason for disappearance: replaced by the VPP5000 series (4.2).

Machine: Fujitsu VPP5000 series.
Year of introduction: 1999.
Year of exit: 2002.
Type: Distributed-memory multi-processor vectorprocessors, max. 128 processors.
Theoretical Peak performance: 1.22 Tflop/s.
Reason for disappearance: Fujitsu does not market vector systems anymore, this machine line is replaced by the PRIMEQUEST series. (4.2).

Machine: Fujitsu PRIMEQUEST 500.
Year of introduction: 2006.
Year of exit: 2010.
Type: Distributed-memory multi-processor vectorprocessors, max. 32 processors.
Theoretical Peak performance: 409.6 Gflop/s.
Reason for disappearance: Fujitsu does not target this system to the HPC market anymore. (4.2).

Machine: Hitachi BladeSymphony.
Year of introduction: 2005.
Year of exit: 2010.
Type: Distributed-memory RISC based system, max. 64 processors.
Theoretical Peak performance: 850 Gflop/s.
Reason for disappearance: Hitachi does not target this system to the HPC market anymore.

Machine: Hitachi S-3800 series.
Year of introduction: 1993.
Year of exit: 1998.
Type: Shared-memory multi-processor vectorprocessors, max. 4 processors.
Theoretical Peak performance: 32 Gflop/s.

Reason for disappearance: Replaced by the SR8000 (4.2).

Machine: Hitachi SR2201 series.
Year of introduction: 1996.
Year of exit: 1998.
Type: Distributed-memory RISC based system, max. 1024 processors.
Theoretical Peak performance: 307 Gflop/s.
Reason for disappearance: Replaced by the newer SR8000 (4.2).

Machine: Hitachi SR8000 series.
Year of introduction: 1998.
Year of exit: 2000–2003 (different models).
Type: Distributed-memory RISC based system, max. 512 processors.
Theoretical Peak performance: 7.3 Tflop/s.
Reason for disappearance: Replaced by the newer SR11000 (4.2).

Machine: Hitachi SR11000 series.
Year of introduction: 1998.
Year of exit: 2005–2009 (different models).
Type: Distributed-memory RISC based system, max. 512 processors.
Theoretical Peak performance: 72 Tflop/s.
Reason for disappearance: Replaced by the newer SR16000 (3.1.7).

Machine: The HP Integrity Superdome.
Year of introduction: 1999.
Year of exit: 2000.
Type: Distributed-memory RISC-based ccNUMA system, max. 64 processors.
Theoretical Peak performance: 819.2 Gflop/s.
Reason for disappearance: No market interest from HP (and customers) anymore.

Machine: The HP Exemplar V2600.
Year of introduction: 1999.
Year of exit: 2000.
Type: Distributed-memory RISC based system, max. 128 processors.
Theoretical Peak performance: 291 Gflop/s.
Reason for disappearance: Replaced by the HP Integrity Superdome. (4.2).

Machine: IBM SP1 series.
Year of introduction: 1992.
Year of exit: 1994.
Type: Distributed-memory RISC based system, max. 64 processors.
Theoretical Peak performance: 8 Gflop/s.
Reason for disappearance: Replaced by the newer RS/6000 SP (4.2).

Machine: IBM RS/6000 SP series.
Year of introduction: 1999.
Year of exit: 2001.
Type: Distributed-memory RISC based system, max. 2048 processors.
Theoretical Peak performance: 24 Gflop/s.
Reason for disappearance: Replaced by the newer eServer p575. (**??**).

Machine: IBM eServer p575.
Year of introduction: 2008.
Year of exit: 2011.

Type: Distributed-memory RISC based system, max. size unspecified
Theoretical Peak performance: unspecified.
Reason for disappearance: Replaced by the newer eServer p775, see 3.1.9.

Machine: IBM BlueGene/P.
Year of introduction: 2007.
Year of exit: 2013.
Type: Distributed-memory RISC based system, max. 2×65,536 processors
Theoretical Peak performance: 367 Tflop/s.
Reason for disappearance: Replaced by the newer BlueGene/Q, see 3.1.8.

Machine: Liquid IQ.
Year of introduction: 2006.
Year of exit: 2009.
Type: Distributed-memory multi-processor system, max. 960 processors.
Theoretical Peak performance: 20 Tflop/s.
Reason for disappearance: No competitive edge anymore with respect to other systems in the area of energy effiency.

Machine: Meiko CS-2.
Year of introduction: 1994.
Year of exit: 1999.
Type: Distributed-memory RISC based system.
Theoretical Peak performance: 200 Mflop/s per processor.
Reason for disappearance: Quadrics Supercomputers World Ltd. does not market the system anymore. The updated network technology is now offered for other systems like Bull NovaScale (see 3.1.1.1).

Machine: NEC Cenju-3.
Year of introduction: 1994.
Year of exit: 1996.
Type: Distributed-memory system, max. 256 processors.
Theoretical Peak performance: 12.8 Gflop/s.
Reason for disappearance: replaced by newer Cenju-4 series (4.2).

Machine: NEC Cenju-4.
Year of introduction: 1998.
Year of exit: 2002.
Type: Distributed-memory system, max. 1024 processors.
Theoretical Peak performance: 410 Gflop/s.
Reason for disappearance: NEC has withdrawn this machine in favour of a possible successor. Specifics are not known, however.

Machine: NEC Express5800/1000.
Year of introduction: 2006.
Year of exit: 2010.
Type: Distributed-memory multi-processor vectorprocessors, max. 32 processors.
Theoretical Peak performance: 409.6 Gflop/s.
Reason for disappearance: NEC does not target this system to the HPC market anymore.

Machine: NEC SX-4.
Year of introduction: 1995.
Year of exit: 1996.
Type: Distributed-memory cluster of SM-MIMD vector processors, max. 256 processors.
Theoretical Peak performance: 1 Tflop/s.

Reason for disappearance: replaced by newer SX-5 series (see below).

Machine: NEC SX-5.
Year of introduction: 1998.
Year of exit: 2002.
Type: Distributed-memory cluster of SM-MIMD vector processors, max. 512 processors.
Theoretical Peak performance: 5.12 Tflop/s.
Reason for disappearance: replaced by newer SX-6 series (see below).

Machine: NEC SX-6.
Year of introduction: 2002.
Year of exit: 2005.
Type: Distributed-memory cluster of SM-MIMD vector processors, max. 1024 processors.
Theoretical Peak performance: 9.2 Tflop/s.
Reason for disappearance: replaced by newer SX-8 series (see below).

Machine: NEC SX-8.
Year of introduction: 2004.
Year of exit: 2007.
Type: Distributed-memory cluster of SM-MIMD vector processors, max. 4096 processors.
Theoretical Peak performance: 90.1 Tflop/s.
Reason for disappearance: replaced by newer SX-9 series (3.1.10).

Machine: Quadrics Appemille.
Year of introduction: 1999.
Year of exit: 2004.
Type: Processor array, max. 2048 processors.
Theoretical Peak performance: 1 Tflop/s.
Reason for disappearance: Not marketed anymore.

Machine: Silicon Graphics Origin2000.
Year of introduction: 1996.
Year of exit: 2000.
Type: Shared-memory multi-processor, max. 128 processors.
Theoretical Peak performance: 102.4 Gflop/s.
Reason for disappearance: replaced by the SGI Origin 3000 (see below).

Machine: Silicon Graphics Origin3000.
Year of introduction: 2003.
Year of exit: 2005.
Type: Shared-memory multi-processor, max. 512 processors.
Theoretical Peak performance: 819 Gflop/s.
Reason for disappearance: replaced by the Itanium-based SGI Altix 4700 (3.1.11).

Machine: SGI Altix 4700.
Year of introduction: 2006.
Year of exit: 2010.
Type: Shared-memory multi-processor, max. 512 processors.
Theoretical Peak performance: 6.8 Tflop/s.
Reason for disappearance: replaced by the Nehalem EX-based SGI Altix UV (3.1.11).

Machine: SiCortex SC series.
Year of introduction: 2006.
Year of exit: 2009.

Type: Distributeded-memory multi-processor, max. 5832 processors.
Theoretical Peak performance: 5.8 Tflop/s.
Reason for disappearance: No new investments, company had to close down.

Machine: SUN M9000.
Year of introduction: 2008.
Year of exit: 2009.
Type: Shared-memory multi-processor, max. 64 processors.
Theoretical Peak performance: 2.58 Tflop/s.
Reason for disappearance: Re-labelled Fujitsu-Siemens machine. SUN does not target the HPC market anymore with this type of systems.

Machine: SUN E10000 Starfire.
Year of introduction: 1997.
Year of exit: 2001.
Type: Shared-memory multi-processor, max. 64 processors.
Theoretical Peak performance: 51.2 Gflop/s.
Reason for disappearance: replaced by the Fire 3800-15K (see below).

Machine: SUN Fire 3800-15K.
Year of introduction: 2001.
Year of exit: 2004.
Type: Shared-memory multi-processor, max. 106 processors.
Theoretical Peak performance: 254 Gflop/s.
Reason for disappearance: replaced by the Fire E25K (see below).

Machine: SUN Fire E25K.
Year of introduction: 2004.
Year of exit: 2007.
Type: Shared-memory multi-processor, max. 72 processors.
Theoretical Peak performance: 432 Gflop/s.
Reason for disappearance: replaced by the M9000 models (see above).

Machine: Thinking Machine Corporation CM-5.
Year of introduction: 1991.
Year of exit: 1996.
Type: Distributed-memory RISC based system, max. 16K processors.
Theoretical Peak performance: 2 Tflop/s.
Reason for disappearance: Thinking Machine Corporation has stopped manufacturing hardware and hopes to keep alive as a software vendor.

5 Systems and components under development

Although we mainly want to discuss real, marketable systems and no experimental, special purpose, or even speculative machines, it is good to look ahead a little and try to see what may be in store for us in the near future.
Below we discuss systems that may lead to commercial systems or components (i.e., mainly processors) to be introduced on the market between somewhat more than half a year to a year from now. The commercial systems that result from it will sometimes deviate significantly from the original research models depending on the way the development is done (the approaches in Japan and the USA differ considerably in this respect) and the user group which is targeted.
In section 2.9 we already noted the considerable interest generated by systems that provide acceleration by means of GPUs, FPGAs or other special computational accelerators like the Intel Phi, etc. Within the near future a HPC vendor cannot afford *not* to include somehow such accelerators into their architectures. The reaction of many vendors has been to offer systems in which accelerators are incorporated (Bull, Cray, Eurotech, SGI, etc.) Also chip vendors may try to incorporate accelerating devices on the chips themselves (as seems the way AMD and Intel are going), or provide ways to thightly integrate accelerator hardware with a CPU and memory via a fast direct connection. We briefly review the status of these developments below.

5.1 AMD

AMD took great pains the last few years to increase the number of standard general purpose x86_64 cores on its chips. The present number being 16 in the Piledriver processor. This processor is a second implementation of its "Bulldozer" type of processors that can house many cores of different type. In view of the fact that AMD acquired the GPU manufacturer ATI a few years back it stands to reason that AMD wants to make the most in combining the technologies of both branches. AMD speaks of its Fusion architecture in connection to these plans. In fact, the first of these processors called Accelerated Processing Units (APUs) are already on the market for desktop systems for some time but until now no APUs fit for HPC server systems are available. The number of GPU cores and the number of SIMD units within them are presently still too low to be a match for the high-end GPUs as delivered by both AMD and NVIDIA. This is partly due to the packaging and for another part to the memory. When the number of graphical units increases the power usage increases with it. In addition, the standard DDR3 memory or the coming DDR4 memory do not deliver enough bandwidth to feed many graphical cores. The way forward probably lies in shrinking the feature size and the availability of 3-D memory in the next few years. Still it will be a challenge to strike a feasible (and commercially viable) balance between the many conflicting design parameters that are involved in such a tight integration.

5.2 Cray Inc.

With the introduction of the XC30 (Cascade) system Cray has made a switch to a very close connection with Intel: not only houses the XC30 Intel processors (presently Sandy Bridge, next Ivy Bridge and then Haswell), Cray also sold its crown jewel, the Aries router, to Intel. For the moment Cray is the only vendor that is allowed to use it and also the next generation interconnect, the Pisces router, which is developed together with Intel will only available for Cray. However, that will not be the case for generations after that. So, it will be difficult for Cray to make a distinction with other vendors from then on.
It might be that Cray wants to diversify its services as one might deduct from its Sonexion storage products and its uRIKA systems for HPA workloads. Indeed, the latter field seems to become more important every

year and providing special systems for this area could be a good prospect. For HPC, however, it seems hard to keep standing out between the many other vendors in a few years.

5.3 IBM

IBM has been working for years on its three generations of BlueGene systems. Many of these first models, the BlueGene/L, that have been installed a few years ago now have been replaced by its follow-up, the BlueGene/P or the latest generation the BlueGene/Q (see 3.1.8).
The other line of HPC sysytems of IBM is based on its POWERx processors. Presently this is the POWER7 which in due course will be replaced by its technology shrink, the POWER7+. In the same year the POWER8 will see the the light, be it not yet for HPC systems. The most interesting part of the current p775 system (3.1.9) is, however, the proprietary optical interconnect, giving it a definite advantage over Infiniband in terms of bandwidth. For the moment, however, nothing is known about a further development of this interconnect which is quite advanced technology-wise but also quite expensive. So, IBM will attempt to lower the cost of this interconnect, which should be possible with the advance in optronic technology while also improving its already impressive characteristics.

5.4 Intel-based systems

As mentioned before in this report, the Itanium (IA-64) line of processors has become irrelevant for the HPC area. Intel instead is focussing on its multi-core x86_64 line of general processors of which the Ivy Bridge and Haswell will be the next generations with the server versions to come in 2013 and 2014, respectively.
As Intel is not insensitive with respect to the fast increase of the use of computational accelerators, an answer might be the many-core processors of which the Knights Bridge (Xeon Phi) is the first product that has to compete with the current generation of GPUs. As the peak performances of both are in the same ball park this may work for the next few years. The next generation, code name Knights Landing, is rumored to be made in 14 nm technology which already would increase to 4 Tflop/s for 64-bit floating-point arithmetic apart from any architectural improvements the peak speed. As such architectural improvements certainly will occur, this can be regarded as a lower bound. Presently it is not expected that the Knights Landing can be placed directly in a socket, which would increase the bandwidth. Whether it will still need separate GDDR memory or able to share with the CPUs is still a guess. It is sure, however, that with an increased core count the high speed ring connection will not be able to sustain a sufficient bandwidth. So, another way of interconnecting the cores will occur, although in what form is still in the dark.
As already remarked above (see 5.2), Intel has bought the Aries switch technology from Cray while it already had acquired the Infiniband vendor Qlogic. Together with the fact that Intel has a PCIe Gen3 connection on chip it is safe to presume that Intel is interested in bringing the interconnect technology towards the chip. In this way no interconnect switches are required anymore. A fact that should worry Infiniband vendors like Mellanox.

5.5 nVIDIA

It is known for some time that NVIDIA in its Denver project plans to incorporate an ARM processor within the GPU. Where in the Kepler series (see 2.9.1.2) there is already possible to launch compute kernels within the GPU itself, the inclusion of the ARM processor can be seen as an extension that will enable a still more independent operation of the GPU with respect to the host CPU. As such it could become a serious threat to the Intel Phi accelerators as one will undoubtly will be able to access the ARM processor as any RISC CPU and so make programmability much easier. The first Denver-based products are to be expected in 2015, while the direct successor of the Kepler GPU, the Maxwell will appear in 2014. There is a debate whether this will be in the first quarter or later in the year. When Maxwell in brought out early, it will be in 28 nm technology. This is because TMSC, that produces the chips, will not be ready for mass production in 20 nm technology in 1Q2014.

5.6 Energy-efficient developments

There is already a considerable activity with regard to build Exaflop/s systems, foremost in the IESP (International Exascale Software Project,[16]). The time frame mentioned for appearance of such systems until recently was 2019–2020. However, it seems that in the USA the development money to keep this schedule is hard to come by and a probable year would be rather 2022. However, both in Asia and Europe there are also plans for developing Exascale systems, be it not necessarily within the constraints as were set by USAs DoD. So, we still may see such (a) system(s) emerging before 2020.
A main concern, however, stays the power consumption of such machines. With the current technologies, even with improvements caused by reduced feature size factored in, would be roughly in the 120-150 MW range. I.e., the power consumption of a mid-sized city. This is obviously unacceptable and the power requirement circulating in the IESP discussions is ≤ 20 MW, the constraint put in by the US Department of Defense. Even if one ignores such a hard constraint it is obvious one cannot proceed with business as usual energy-wise.
This has fuelled the interest for alternative processor architectures like those used in embedded processors and at the moment a a reseach project, "Green Flash", is underway at Lawrence Berkeley Laboratory to build an Exaflops/s system using $20{\cdot}10^6$ Tensilica Xtensa embedded processors. Besides that a host of other maufacturers are now exploring this direction. Among them ARM, Texas Instruments, Adapteva, Tilera and many others.
Although the processor is an important constituent in the power budget there are others that are becoming increasingly important. Among them are memory and the processor interconnect. Directions to decrease the power consumption of these components lie in transition to a form of non-volatile memory, i.e., memory that does not use energy to maintain its contents and the use of photonic interconnects. With respect to the latter development: the first step is taken by IBM with its p775 eServer (see 3.1.9).
With regard to non-volatile memory one has to distinguish between storage class memory that can replace spinning disks (or the current generation of SSD Flash Memory) and the technologies that should replace DRAM as we currently use it. The former may be replaced by Phase Change RAM (PCRAM) in the very near future, indeed the first offerings of PCRAM are done at the time of writing this report. For the latter type of memory there are a number of candidate technologies, like MRAM, FeRAM and memristors. MRAM is already used in embedded applications but at an insufficient density to be applicable in present systems. FeRAM and memristors may become available in a 2–3 year time frame.

6 Glossary of terms

This section contains the explanation of some often-used terms that either are not explained in the text or, by contrast, are described extensively and for which a short description may be convenient.

6.1 Glossary

API: API stands for Application Program(mer) Interface. Ususally it consists of a set of library functions that enable to access and/or control the functionality of certain non-standard devices, like an I/O device or a computational accelerator.

Architecture: The internal structure of a computer system or a chip that determines its operational functionality and performance.

Architectural class: Classification of computer systems according to its architecture: e.g., distributed memory MIMD computer, symmetric multi processor (SMP), etc. See this glossary and section 2.1 for the description of the various classes.

ASCI: Accelerated Strategic Computer Initiative. A massive funding project in the USA concerning research and production of high-performance systems. The main motivation is said to be the management of the USA nuclear stockpile by computational modeling instead of actual testing. ASCI has greatly influenced the development of high-performance systems in a single direction: clusters of SMP systems.

ASIC: Application Specific Integrated Circuit. A chip that is designed to fulfill a specific task in a computer system, e.g. for routing messages in a network.

Bank cycle time: The time needed by a (cache-)memory bank to recover from a data access request to that bank. Within the bank cycle time no other requests can be accepted.

Beowulf cluster: Cluster of PCs or workstations with a private network to connect them. Initially the name was used for do-it-yourself collections of PCs mostly connected by Ethernet and running Linux to have a cheap alternative for "integrated" parallel machines. Presently, the definition is wider including high-speed switched networks, fast RISC-based processors and complete vendor-preconfigured rack-mounted systems with either Linux or Windows as an operating system.

Bit-serial: The operation on data on a bit-by-bit basis rather than on byte or 4/8-byte data entities in parallel. Bit-serial operation is done in processor array machines where for signal and image processing this mode is advantageous.

Cache — data, instruction: Small, fast memory close to the CPU that can hold a part of the data or instructions to be processed. The primary or level 1 (L1) caches are virtually always located on the same chip as the CPU and are divided in a cache for instructions and one for data. A secondary or level 2 (L2) cache is sometimes located off-chip and holds both data and instructions. Caches are put into the system to hide the large latency that occurs when data have to be fetched from memory. By loading data and or instructions into the caches that are likely to be needed, this latency can be significantly reduced.

Capability computing: A type of large-scale computing in which one wants to accommodate very large and time consuming computing tasks. This requires that parallel machines or clusters are managed with the highest priority for this type of computing possibly with the consequence that the computing resources in the system are not always used with the greatest efficiency.

Capacity computing: A type of large-scale computing in which one wants to use the system (cluster) with the highest possible throughput capacity using the machine resources as efficient as possible. This may have adverse effects on the performance of individual computing tasks while optimising the overall usage of the system.

ccNUMA: Cache Coherent Non-Uniform Memory Access. Machines that support this type of memory access have a physically distributed memory but logically it is shared. Because of the physical difference of the location of the data items, a data request may take a varying amount of time depending on the location of the data. As both the memory parts and the caches in such systems are distributed a mechanism is necessary to keep the data consistent system-wide. There are various techniques to enforce this (directory memory, snoopy bus protocol). When one of these techniques is implemented the system is said to be cache coherent.

Clock cycle: Fundamental time unit of a computer. Every operation executed by the computer takes at least one and possibly multiple cycles. Typically, the clock cycle is now in the order of one to a quarter of a nanosecond.

Clock frequency: Reciproke of the clock cycle: the number of cycles per second expressed in Hertz (Hz). Typical clock frequencies nowadays are 1–4 GHz.

Clos network: A logarithmic network in which the nodes are attached to switches that form a *spine* that ultimately connects all nodes.

Co-Array Fortran: Co-Array Fortran (CAF) is a so-called Partitioned Global Address Space programming language (PGAS language, see below) that extends Fortran by allowing to specify a processor number for data items within distributed data structures. This allows for processing such data without explicit data transfer between the processors. CAF will be incorporated in Fortran 2003, the upcoming Fortran standard.

Communication latency: Time overhead occurring when a message is sent over a communication network from one processor to another. Typically the latencies are in the order of a few μs for specially designed networks, like Infiniband or Myrinet, to about 40 μs for Gbit Ethernet.

Control processor: The processor in a processor array machine that issues the instructions to be executed by all the processors in the processor array. Alternatively, the control processor may perform tasks in which the processors in the array are not involved, e.g., I/O operations or serial operations.

CRC: Type of error detection/correction method based on treating a data item as a large binary number. This number is divided by another fixed binary number and the remainder is regarded as a checksum from which the correctness and sometimes the (type of) error can be recovered. CRC error detection is for instance used in SCI networks.

Crossbar (multistage): A network in which all input ports are directly connected to all output ports without interference from messages from other ports. In a one-stage crossbar this has the effect that for instance all memory modules in a computer system are directly coupled to all CPUs. This is often the case in multi-CPU vector systems. In multistage crossbar networks the output ports of one crossbar module are coupled with the input ports of other crossbar modules. In this way one is able to build networks that grow with logarithmic complexity, thus reducing the cost of a large network.

Distributed Memory (DM): Architectural class of machines in which the memory of the system is distributed over the nodes in the system. Access to the data in the system has to be done via an interconnection network that connects the nodes and may be either explicit via message passing or implicit (either using HPF or automatically in a ccNUMA system).

Dragonfly topology A hierarchical three-level interconnect topology for the levels router, group, and system. The crux in this topology is to have high-radix routers that tightly interconnect on a local and group level. This leads to a low number of hops for traversing the system.

EPIC: Explicitly Parallel Instruction Computing. This term is coined by Intel for its IA-64 chips and the Instruction Set that is defined for them. EPIC can be seen as Very Large Instruction Word computing with a few enhancements. The gist of it is that no dynamic instruction scheduling is performed as is done in RISC processors but rather that instruction scheduling and speculative execution of code is determined beforehand in the compilation stage of a program. This simplifies the chip design while potentially many instructions can be executed in parallel.

Fat tree: A network that has the structure of a binary (quad) tree but that is modified such that near the root the available bandwidth is higher than near the leafs. This stems from the fact that often a root processor has to gather or broadcast data to all other processors and without this modification contention would occur near the root.

Feature size: The typical distance between the various devices on a chip. By now processors with a feature size of 45 nanometer (10^{-9} m, nm) are on the market. Lowering the feature size is beneficial in that the speed of the devices on the chip will increase because of the smaller distances the electrons have to travel. The feature size cannot be shrunk much more though, because of the leaking of current that can go up to unacceptable levels. The lower bound is now believed to be around 7–8 nm for Silicon.

Flop: floating-point operation. Flop/s are used as a measure for the speed or performance of a computer. Because of the speed of present day computers, rather Megaflop/s (Mflop/s, 10^6 flop/s), Gigaflop/s (Gflop/s, 10^9 flop/s), Teraflop/s (Tflop/s, 10^{12} flop/s), and Petaflop/s (Pflop/s, 10^{15} flop/s) are used.

FMA Fused Multiply-Add. A combined floating-point operation that can be extcuted within one clock cycle and only has to be rounded once instead of twice for each separate operation.

FPGA: FPGA stands for Field Programmable Gate Array. This is an array of logic gates that can be hardware-programmed to fulfill user-specified tasks. In this way one can devise special purpose functional units that may be very efficient for this limited task. As FPGAs can be reconfigured dynamically, be it only 100–1,000 times per second, it is theoretically possible to optimise them for more complex special tasks at speeds that are higher than what can be achieved with general purpose processors.

Frontside Bus: Bus that connects the main memory with the CPU core(s) via a memory controller (only in scalar processors, not in vector processors). The frontside bus has increasingly become a bottleneck in the performance of a system because of the limited capacity of delivering data to the core(s).

Functional unit: Unit in a CPU that is responsible for the execution of a predefined function, e.g., the loading of data in the primary cache or executing a floating-point addition.

Grid — 2-D, 3-D: A network structure where the nodes are connected in a 2-D or 3-D grid layout. In virtually all cases the end points of the grid are again connected to the starting points thus forming a 2-D or 3-D torus.

HBA: HBA stands for Host Bus Adapter, also known as Host Channel Adaptor (specifically for InfiniBand). It is the part in an external network that constitutes the interface between the network itself and the PCI bus of the compute node. HBAs usually carry a good amount of processing intelligence themselves for initiating communication, buffering, checking for correctness, etc. HBAs tend to have different names in different networks: HCA or TCA for Infiniband, LANai for Myrinet, etc.

HPA: High Performance (Data) Analytics. The analysis of very large amounts of data in order to find useful/interesting patterns. First used in marketing circles to track customer profiles but increasingly used for scientific data analysis, e.g., in genomics, climatology records, astronomy data, etc.

HPCS: Abbrevation of High Productivity Computer Systems: a program initiated by DARPA, the US Army Agency that provides large-scale financial support to future (sometimes futuristic) research that might benefit the US Army in some way. The HPCS program was set up to ensure that by 2010 computer systems will exist that are capable of a performance of 1 Pflop/s in real applications as opposed to the Theoretical Peak Performance which might be much higher. Initially Cray, HP, IBM,

SGI, and SUN participated in the program. After repeated evaluations only Cray and IBM still get support from the HPCS program.

HPF: High-Performance Fortran. A compiler and run time system that enables to run Fortran programs on a distributed memory system as on a shared memory system. Data partition, processors layout, etc. are specified as comment directives that makes it possible to run the processor also serially. Present HPF available commercially allow only for simple partitioning schemes and all processors executing exactly the same code at the same time (on different data, so-called Single Program Multiple Data (SPMD) mode).

Hypercube: A network with logarithmic complexity which has the structure of a generalised cube: to obtain a hypercube of the next dimension one doubles the perimeter of the structure and connect their vertices with the original structure.

HyperTransport: An AMD-developed bus that directly connects a processor to its memory without use of a Frontside Bus at high speed. It also can connect directly to other processors and because the specification is open, also other types of devices, like computational accelerators can be connected in this fashion to the memory. Intel provides a similar type of bus, called QPI.

IDE Integrated Development Environment. A software environment that integrates several tools to write, debug, and optimise programs. The added value of an IDE lies (should lie) in the direct feedback from the tools. This should shorten the development time for optimised programs for the target architecture. The most well-kown IDE is probably IBM's Eclipse.

Instruction Set Architecture: The set of instructions that a CPU is designed to execute. The Instruction Set Architecture (ISA) represents the repertoire of instructions that the designers determined to be adequate for a certain CPU. Note that CPUs of different making may have the same ISA. For instance the AMD processors (purposely) implement the Intel IA-32 ISA on a processor with a different structure.

LUT: Look-up table. A measure for the amount of memory cells on an FPGA.

Memory bank: Part of (cache) memory that is addressed consecutively in the total set of memory banks, i.e., when data item $a(n)$ is stored in bank b, data item $a(n+1)$ is stored in bank $b+1$. (Cache) memory is divided in banks to evade the effects of the bank cycle time (see above). When data is stored or retrieved consecutively each bank has enough time to recover before the next request for that bank arrives.

Message passing: Style of parallel programming for distributed memory systems in which non-local data that is required explicitly must be transported to the processor(s) that need(s) it by appropriate send and receive messages.

MPI: A message passing library, Message Passing Interface, that implements the message passing style of programming. Presently MPI is the *de facto* standard for this kind of programming.

Multi-core chip: A chip that contains more than one CPU core and (possibly common) caches. Due to the progression of the integration level more devices can be fitted on a chip. AMD, Fujitsu, IBM, and Intel make multi-core chips. Currently the maximum amount of cores per chip is around 16.

Multithreading: A capability of a processor core to switch to another processing thread, i.e., a set of logically connected instructions that make up ((a part of) a process. This capability is used when a process thread stalls, for instance because necessary data are not yet available. Switching to another thread that has instructions that can be executed will yield a better processing utilisation.

NUMA factor: The difference in speed of accessing local and non-local data. For instance when it takes 3 times longer to access non-local data than local data, the NUMA factor is 3.

OpenMP: A shared memory parallel programming model in which shared memory systems and SMPs can be operated in parallel. The parallelisation is controlled by comment directives (in Fortran) or pragmas (in C and C++), so that the same programs also can be run unmodified on serial machines.

PCI bus: Bus on PC node, typically used for I/O, but also to connect nodes with a communication network. The highest bandwidth PCI-X, a common PCI bus version is ≈ 1 GB/s, while its successor, PCI Express, Generartion 2 now normally is available with a 4–8 GB/s bandwidth. The newest version PCI Express, Generation 3 allows for a maximum of 16 GB/s for a 16×connection.

PGAS Languages: Partitioned Global Address Space languages. A family of languages that allow to specify how data items are distributed over the available processes. This gives the opportunity to process these data items in a global fashion without the need for explicit data transfer between processors. It is believed that at least for a part of the HPC user community this makes parallel programming more accessible. The most known languages are presently Unified Parallel C (UPC) and Co-Array Fortran (CAF). Also Titanium, a Java-like language is employed. Apart from these languages that are already in use (be it not extensively) Chapel, developed by Cray and X10, developed by IBM, both under a contract with the US Department of Defense, have PGAS facilities (and more). However, the latter languages are still in the development phase and no complete compilers for these languages are available yet.

Pipelining: Segmenting a functional unit such that it can accept new operands every cycle while the total execution of the instruction may take many cycles. The pipeline construction works like a conveyor belt accepting units until the pipeline is filled and than producing results every cycle.

Processor array: System in which an array (mostly a 2-D grid) of simple processors execute its program instructions in lock-step under the control of a Control Processor.

PVM: Another message passing library that has been widely used. It was originally developed to run on collections of workstations and it can dynamically spawn or delete processes running a task. PVM now largely has been replaced by MPI.

Quad-core chip: A chip that contains four CPU cores and (possibly common) caches. Due to the progression of the integration level more devices can be fitted on a chip. AMD, Fujitsu, IBM, and Intel made quad-core chips that have been followed by so-called many-core chips that hold 8–16 cores.

QPI: QuickPath Interface: Formerly known as Common System Interface (CSI). A bus structure developed by Intel and available since early 2009 that directly connects a (variety of) processor(s) of a system at high speed to its memory without the need for a Frontside Bus. It also can be used for connecting processors to each other. A similar type of interconnection, HyperTransport, is already provided for some years by AMD for its Opteron processors (see **??** for details).

Radix The amount of incoming and outgoing ports in a router within a system interconnect. For instance, in a 2-D torus the radix is 4 (bi-directional) and in a tree topology it is 3.

Register file: The set of registers in a CPU that are independent targets for the code to be executed possibly complemented with registers that hold constants like 0/1, registers for renaming intermediary results, and in some cases a separate register stack to hold function arguments and routine return addresses.

RISC: Reduced Instruction Set Computer. A CPU with its instruction set that is simpler in comparison with the earlier Complex Instruction Set Computers (CISCs). The instruction set was reduced to simple instructions that ideally should execute in one cycle.

SDK: Software Development Kit. The term has become more common as many vendors that sell computational accelerator hardware also need to provide the software that is necessary to make the accelerator effectively usable for non-specialist HPC users.

SERDES Serialiser/deserialiser. Unit that encodes data to be sent or received from/to a (compute) node via parallel links to another part of the system.

Shared Memory (SM): Memory configuration of a computer in which all processors have direct access to all the memory in the system. Because of technological limitations on shared bandwidth generally not more than about 16 processors share a common memory.

`shmem`: One-sided fast communication library first provided by Cray for its systems. However, `shmem` implementations are also available for SGI and some other systems.

SIMD Single Instruction stream Multiple Data stream. Used to characterise a class of machines (see 2.2, 2.3). However, presently also used for characterising the working of the vector units in CPUs and the units in GPUs that drive the compute cores; both types of units operate indeed in SIMD mode.

SMP: Symmetric Multi-Processing. This term is often used for compute nodes with shared memory that are part of a larger system and where this collection of nodes forms the total system. The nodes may be organised as a ccNUMA system or as a distributed memory system of which the nodes can be programmed using OpenMP while inter-node communication should be done by message passing.

TLB: Translation Look-aside Buffer. A specialised cache that holds a table of physical addresses as generated from the virtual addresses used in the program code.

Torus: Structure that results when the end points of a grid are wrapped around to connect to the starting points of that grid. This configuration is often used in the interconnection networks of parallel machines either with a 2-D grid or with 3-D grid.

U: A unit used in defining the height of a component in a standard 19-inch wide rack system. 1 U is 44.5 mm or 1.75 inch.

UPC: Unified Parallel C (UPC). A PGAS language (see above). UPC is an extension of C that offers the possibility to specify how data items are distributed over the available processors, thus enabling processing these data without explicit data transfers between the processors.

Vector register: A multiple entry register (typically 128–256) that hold the operands to be processed by a vector unit. Using vector registers controlled by vector instructions guarantees the production of results every cycle for the amount of operands in the register.

Vector unit (pipe): A pipelined functional unit that is fed with operands from a vector register and will produce a result every cycle (after filling the pipeline) for the complete contents of the vector register.

Virtual Shared Memory: The emulation of a shared memory system on a distributed memory machine by a software layer.

VLIW processing: Very Large Instruction Word processing. The use of large instruction words to keep many functional units busy in parallel. The scheduling of instructions is done statically by the compiler and, as such, requires high quality code generation by that compiler. VLIW processing has been revived in Intel's IA-64 chip architecture, there called EPIC (see above).

Acknowledgments

Again, it is not possible to thank all people that have been contributing to this overview. Many vendors and people interested in this project have been so kind to provide us with the vital information or to correct us when necessary. Therefore, we will have to thank them here collectively but not less heartily for their support.

References

[1] R. Alverson, D. Roweth, L. Kaplan, *The Gemini Interconnect*, 18th IEEE Symposium on High Performance Interconnects, August 2010, 83–87.

[2] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, W. Zwaenepoel, *TreadMarks: Shared Memory Computing on Networks of Workstations*, to appear in IEEE Computer (also: `www.cs.rice.edu/~willy/TreadMarks/papers.htm`)

[3] The ASCI program: `http://www.sandia.gov/ASC/`.

[4] The home page for Co-array Fortran can be found at: `http://www.co-array.org/`.

[5] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon, *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers Inc., January 2001.

[6] B. Chapman, G. Jost, R. van der Pas, *Using OpenMP*, MIT Press, Boston, 2007.

[7] D.E. Culler, J.P. Singh, A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers Inc., August 1998.

[8] D.W. Doerfler, *An Analysis of the Pathscale Inc. Infiniband Host Channel Adapter, InfiniPath*, Sandia Report, SAND2005-5199, August 2005.

[9] Bibliography page of Distributed Shared Memory systems:
`http://www.cs.umd.edu/~keleher/bib/dsmbiblio/dsmbiblio.html`.

[10] Directory with EuroBen results: `www.euroben.nl/results`.

[11] M.J. Flynn, *Some computer organisations and their effectiveness*, IEEE Trans. on Computers, Vol. C-21, 9, (1972) 948–960.

[12] A. Geist, A. Beguelin, J. Dongarra, R. Manchek, W. Jaing, and V. Sunderam, *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Boston, 1994.

[13] D.A. Bader, J. Berry, S. Kahan, R. Murphy, E.J. Riedy, J. Wilcock, *The Graph 500 list*, edition June 2011. The Graph 500 site has URL `www.graph500.org`.

[14] D.B. Gustavson, Q. Li, *Local-Area MultiProcessor: the Scalable Coherent Interface*, SCIzzL Report, Santa Clara University, Dept. of Computer Engineering, 1995. Available through: `www.scizzl.com`.

[15] R. W. Hockney, C. R. Jesshope, *Parallel Computers II*, Bristol: Adam Hilger, 1987.

[16] Most material on Exascale computing can be found at `www.exascale.org`.

[17] T. Horie, H. Ishihata, T. Shimizu, S. Kato, S. Inano, M. Ikesaka, *AP1000 architecture and performance of LU decomposition*, Proc. Internat. Symp. on Supercomputing, Fukuoka, Nov. 1991, 46–55.

[18] HPC Challenge Benchmark, `icl.cs.utk.edu/hpcc/`.

[19] High Performance Fortran Forum, *High Performance Fortran Language Specification*, Scientific Programming, **2**, 13, (1993) 1–170.

[20] D.V. James, A.T. Laundrie, S. Gjessing, G.S. Sohi, *Scalable Coherent Interface*, IEEE Computer, **23**, 6, (1990),74–77.

[21] J. Kim, W.J. Dally, S. Scott, D. Abts, *Technology-Driven, Highly-Scalable Dragonfly Topology*, IEEE Intl. Symposium on Computer Architecture, (2008), 77–88.

[22] Julie Langou, Julien Langou, P. Luszczek, J. Kurzuk, J.J. Dongarra, *Exploiting the Performance of 32-Bit Floating Point Arithmetic in Obtaining 64-Bit Accuracy*, Proceedings of SC06, Tampa, Nov. 2006.

[23] T. Maruyama, T. Yoshida, R. Kan, I. Yamazaki, S. Yamamura, N. Takahashi, M. Hondou, H. Okano, *SPARC64 VIIIfx: A New-generation Octocore Processor for Petascale Computing*, IEEE Micro, **30**, 2, (2010), 30–40.

[24] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI: The Complete Reference Vol. 1, The MPI Core*, MIT Press, Boston, 1998.

[25] W. Gropp, S. Huss-Ledermann, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, M. Snir *MPI: The Complete Reference, Vol. 2, The MPI Extensions*, MIT Press, Boston, 1998.

[26] `http://www.myrinet.com`

[27] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, Wen-King Su, *Myrinet – A Gigabit-per-second Local Area Network*, IEEE Micro, **15**, No. 1, Jan. 1995, 29–36.

[28] Web page for the NAS Parallel benchmarks NPB2: `http://www.nas.nasa.gov/Software/NPB/`

[29] OpenMP Forum, *OpenMP Application Interface, version 2.5*, Web page: `www.openmp.org/`, May 2005.

[30] C. Schow, F. Doany, J. Kash, *Get on the Optical Bus*, IEEE Spectrum, September 2010, 31–35.

[31] T. Shanley, *Infiniband Network Architecture*, Addison-Wesley, Nov. 2002.

[32] D.H.M. Spector, *Building Unix Clusters*, O'Reilly, Sebastopol, CA, USA, July 2000.

[33] A.J. van der Steen, *Exploring VLIW: Benchmark tests on a Multiflow TRACE 14/300*, Academic Computing Centre Utrecht, Technical Report TR-31, April 1990.

[34] A.J. van der Steen, ed. *Aspects of computational science*, NCF, The Hague, 1995.

[35] A.J. van der Steen, *An evaluation of some Beowulf clusters*, Technical Report WFI-00-07, Utrecht University, Dept. of Computational Physics, December 2000. (Also available through `www.euroben.nl`, directory `reports/`.)

[36] A.J. van der Steen, *Overview of recent supercomputers*, June 2005, `www.euroben.nl`, directory `reports/`.)

[37] A.J. van der Steen, *Overview of recent supercomputers*, October 2011, `www.euroben.nl`, directory `reports/`.)

[38] T.L. Sterling, J. Salmon, D.J. Becker, D.F. Savaresse, *How to Build a Beowulf*, The MIT Press, Boston, 1999.

[39] H.W. Meuer, E. Strohmaier, J.J. Dongarra, H.D. Simon, *Top500 Supercomputer Sites*, 38th Edition, June 2013, The report can be downloaded from: `www.top500.org/`.

[40] Task Force on Cluster Computing home page: `http://www.cloudbus.org/~raj/tfcc/`.

[41] The home page of UPC can be found at: `http://upc.gwu.edu/`.