Acta Numerica (2012), pp. 1–96
doi:10.1017/S09624929XXXXXXXX

High-performance computing systems: Status and outlook*

J. J. Dongarra
University of Tennessee
and
Oak Ridge National Laboratory
and
University of Manchester
E-mail: dongarra@eecs.utk.edu

A. J. van der Steen
NCF/HPC Research, L. J. Costerstraat 5,
6827 AR Arnhem, The Netherlands
E-mail: steen@hpcresearch.nl

This article describes the current state of the art of high-performance computing systems, and attempts to shed light on near-future developments that might prolong the steady growth in speed of such systems, which has been one of their most remarkable characteristics. We review the different ways devised to speed them up, both with regard to components and their architecture. In addition, we discuss the requirements for software that can take advantage of existing and future architectures.

* Colour online for monochrome figures available at journals.cambridge.org/anu.

CONTENTS

1. Introduction

High-performance computer systems can be regarded as the most powerful and flexible research instruments today. They are employed to model phenomena in fields as varied as climatology, quantum chemistry, computational medicine, high-energy physics and many others. In this article we present some of the architectural properties and computer components that make up high-performance computers at present, and also give an outlook on systems to come. Even though the speed of computers has increased tremendously over the years (often a doubling in speed every two or three years), the need for ever faster computers is still there and will not disappear in the foreseeable future.

Before going on to descriptions of the machines themselves, it is useful to consider some mechanisms that are or have been used to increase performance. The hardware structure or *architecture* determines to a large extent the possibilities and impossibilities of 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 article we will place most emphasis on hardware architecture. For a description of machines that can be classified as 'high-performance' we refer readers to Culler, Singh and Gupta (1998) or van der Steen (1995).

The rest of the paper is organized as follows. Section 2 discusses the main architectural classification of high-performance computers. Section 3 presents shared-memory vector SIMD machines. Section 4 discusses distributed-memory SIMD machines. Section 5 looks at shared-memory MIMD machines. Section 6 considers distributed-memory MIMD machines. Section 7 discusses ccNUMA machines, which are closely related to shared-memory systems. Section 8 presents clusters. Section 9 overviews processors, and looks at what is currently available. Section 10 presents computational accelerators, GPUs, and FPGAs. Section 11 discusses networks and what is commercially available. Section 12 overviews recent trends in high-performance computing. Section 13 concludes with an examination of some of the challenges we face in the effective use of high-performance computers.

2. The main architectural classes

For many years, the taxonomy of Flynn (1972) has proved to be useful in the classification of high-performance computers. This classification is based on the ways of manipulating instruction and data streams and comprises four main architectural classes. We will first briefly sketch these classes; afterwards we will fill in some details and describe each class separately.

SISD (single instruction, single data) machines. These are the conventional systems that contain one CPU and hence can accommodate one instruction stream that is executed serially. Today almost all large servers have more than one CPU, but each of these executes instruction streams that are unrelated. Therefore, such systems should still be regarded as (several) SISD machines acting on different data spaces. Examples of SISD machines are workstations, offered by many vendors. The definition of SISD machines is given here for the sake of completeness, but we will not discuss this type of machine in our report.

SIMD (single instruction, multiple data) machines. Such systems often have a large number of processing units, so that all may execute the same instruction on different data in lockstep. Thus, a single instruction manipulates many data items in parallel. Examples of SIMD machines in this class are the CPP Gamma II and the Quadrics Apemille, which are no longer marketed. Nevertheless, the concept is still interesting and is returning in the form of a co-processor in HPC systems, albeit in a somewhat restricted form in some computational accelerators such as graphical processing units (GPUs).

Another subclass of SIMD systems comprises vector processors, which act on arrays of similar data, rather than on single data items, using specially structured CPUs. When data can be manipulated by these vector units, results can be delivered with a rate of one, two and – in special cases –

three per clock cycle (a clock cycle being defined as the basic internal unit of time for the system). So, vector processors execute in an almost parallel way, but only when executing in vector mode on their data. In this case they are several times faster than when executing in conventional scalar mode. For practical purposes, therefore, vector processors are usually regarded as SIMD machines. Examples of such systems are the NEC SX-9B and the Cray X2.

MISD (multiple instruction, single data) machines. Theoretically, in this type of machine 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 (multiple instruction, multiple data) machines. These machines execute several instruction streams in parallel on different data. The difference from the multiprocessor 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. Thus, MIMD systems may run many sub-tasks in parallel in order to shorten the time-to-solution for the main task to be executed. There is a large variety of MIMD systems, and, particularly for this class, the Flynn taxonomy does not quite suffice for the classification of systems. Systems that behave very differently, such as a 4-processor NEC SX-9 vector system and a 100 000-processor IBM BlueGene/P, fall into 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 are 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 either SIMD or MIMD. Single-CPU vector processors can be regarded as an example of the former, while the multi-CPU models of these machines are examples of the latter. 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. 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 lockstep, all have distributed memories associated with the processors. As we will see, distributed-memory MIMD systems exhibit a large variety

of topologies in 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 may have an impact on performance. For distributed-memory systems we will sometimes use the terms DM-SIMD and DM-MIMD to indicate the two subclasses.

As already mentioned, although the difference between shared and distributed-memory machines seems clear-cut, this is not always the case from the 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 (High Performance Fortran Forum 1993) 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 (MPP) systems, such as 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 7. In addition, packages such as TreadMarks (Amza *et al.* 1995) provide a 'distributed-shared-memory' environment for networks of workstations.[1] Since 2006, Intel has marketed its 'Cluster OpenMP' (based on TreadMarks) as a commercial product. It allows the use of the shared-memory OpenMP parallel model to be used on distributed-memory clusters. For the last few years companies such as ScaleMP and 3Leaf have provided products to aggregate physical distributed memory into virtual shared memory.

please check

Lastly, so-called partitioned global address space (PGAS) languages such as Co-Array Fortran (CAF) and Unified Parallel C (UPC) are gaining popularity due to the recent emergence of multi-core processors. With proper implementation this allows a global view of the data, and language facilities make it possible to specify the processing of data associated with a (set of) processor(s) without the need to explicitly move 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 (Gigabit) Ethernet, or other, faster networks, and set to work concurrently on tasks in the same program. Conceptually, this is no different from DM-MIMD computing, but the communication between processors can be much slower. Packages that initially were

[1] A comprehensive bibliography of distributed shared memory systems can found at www.cs.umd.edu/~keheler/dsmbiblio/dsmbiblio.html.

made to realize distributed computing, such as PVM (standing for parallel virtual machine: Geist *et al.* 1994), and MPI (message-passing interface: Snir *et al.* 1998, Gropp *et al.* 1998) 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. For example, the NEC SX-9 and the Cray X2 both have this structure. Thus, 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 no longer used very much and the development has stopped. MPI has now more or less become the *de facto* standard.

For SM-MIMD systems we mention OpenMP (Chandra *et al.* 2001, Chapman, Jost and van der Pas 2007),[2] which can be used to parallelize Fortran and C/C++ programs by inserting comment directives (Fortran 77/90/95) or pragmas (C/C++) into the code. OpenMP has quickly been adopted by all major vendors and has become a well-established standard for shared-memory systems.

Note, however, that for both MPI-2 and OpenMP 2.5 many systems/compilers only implement part of the latest standards. Therefore one has to enquire carefully whether a particular system enjoys the full functionality of these standards. The standard vendor documentation will almost never be clear on this point.

3. Shared-memory SIMD machines

This subclass of machines is practically equivalent to single-processor vector processors, although other interesting machines in this subclass have existed (*i.e.*, VLIW machines: van der Steen 1990), and may emerge again in the near future. In the block diagram in Figure 3.1 we depict a generic model of a vector architecture. The single-processor vector machine will have only one of the vector processors 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. Vector processors may have a cache but in many cases the vector unit cannot take advantage of it, and execution speed may in some cases be unfavourably affected because of frequent cache overflow. Of late, however, this tendency has been reversed because of the increasing gap in

[2] OpenMP Application Interface, version 2.5: www.openmp.org/.

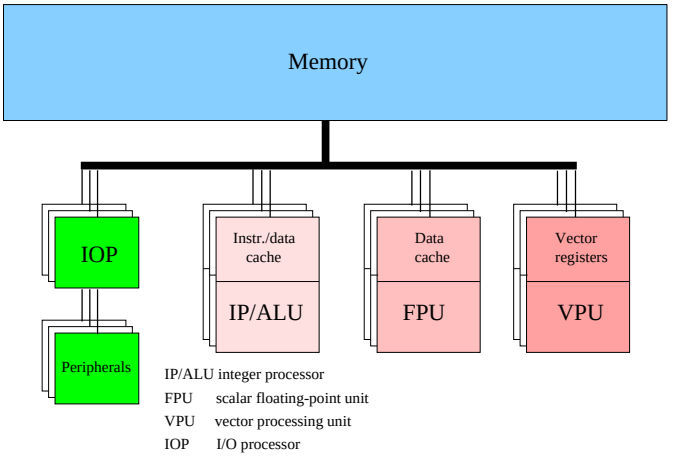


Figure 3.1. Block diagram of a vector processor.

speed between the memory and the processors: the Cray X2 has a cache, and NEC's SX-9 vector system has a facility that is somewhat like a cache.

Although vector processors have existed that loaded their operands directly from memory, and stored the results again immediately in memory (CDC Cyber 205, ETA-10), present-day vector processors use vector registers. This impairs the speed of operations while providing much more flexibility in gathering operands and manipulation with intermediate results.

Because of the generic nature of Figure 3.1 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 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 3.2. Because of the high costs of implementing these data paths between memory and the VPU, compromises are often sought and the full required bandwidth (*i.e.*, two load operations and one store operation at the *same* time) is seldom realized. Only Cray Inc., in its former Y-MP, C-series and T-series, has employed this very high bandwidth. Vendors now rely on additional caches and other tricks to hide the lack of bandwidth.

The VPUs are shown as a single block in Figure 3.1. However, there is considerable diversity in the structure of VPUs. Every VPU consists of a number of vector functional units, or 'pipes', that fulfil one or several functions in the VPU. Every VPU will have pipes that are designated

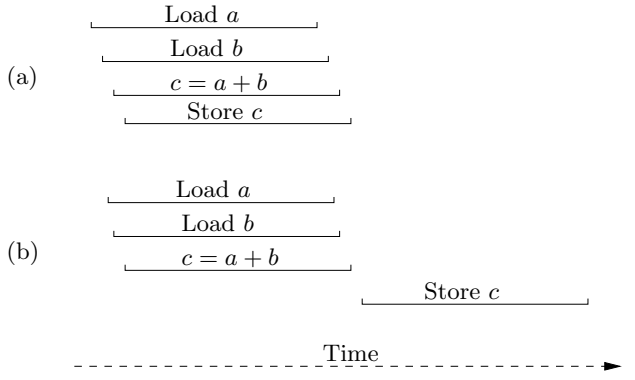


Figure 3.2. Schematic diagram of a vector addition. Case (a) when two load pipes and one store pipe are available; case (b) when two load/store pipes are available.

to perform memory access functions, thus ensuring the timely delivery of operands to the arithmetic pipes and storage of results in memory. 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 to 16-fold replication occurs). Ideally, this will increase performance per VPU by the same factor, provided the bandwidth to memory is adequate.

Lastly, it must be remarked that vector processors as described here are no longer considered a viable economic option, and both the Cray X2 and the NEC SX-9 will disappear in the near future: vector units within standard processor cores and computational accelerators have invaded the vector processing area. Although they are less efficient and have bandwidth limitations, they are so much cheaper that they out-compete classical vector processors.

4. Distributed-memory SIMD machines

Machines of the DM-SIMD type are sometimes also known as *processor array* machines (Hockney and Jesshope 1988). Because the processors of these machines operate in lockstep, *i.e.*, all processors execute the same instruction at the same time (but on different data items), no synchronization 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. At present, no commercially avail-

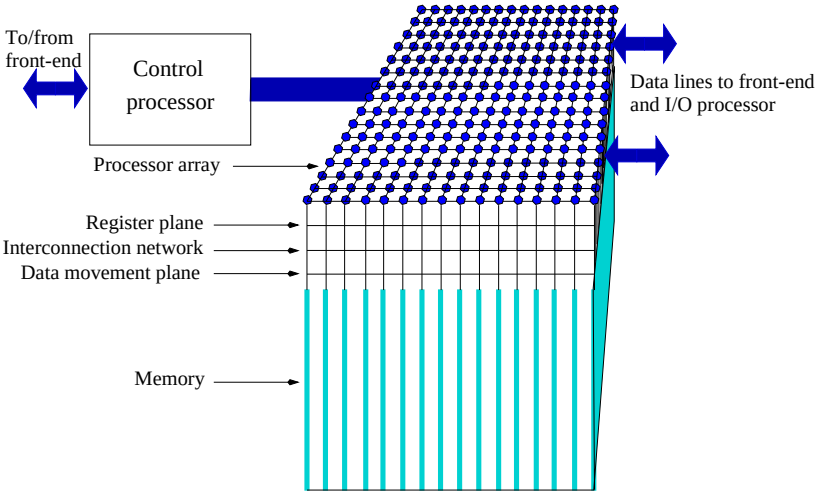


Figure 4.1. A generic block diagram of a distributed-memory SIMD machine.

able machines of the processor array type are marketed. However, because of the shrinking size of devices on a chip, it may be worthwhile locating 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 why 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 4.1 shows a generic model of a DM-SIMD machine, from which actual models will deviate to some degree. Figure 4.1 might suggest that all processors in such systems are connected in a 2D grid, and indeed the interconnection topology of this type of machine always includes the 2D 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 3D, 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 during 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 the processors. To access the data in processor j, the data will have

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

The control processor as depicted in Figure 4.1 may be more or less intelligent. It issues the instruction sequence that will be executed by the processor array. In the worst case (*i.e.*, 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 the case of a more autonomous control processor this can be avoided, thus saving processing interrupts both on the front-end and on the control processor. Most DM-SIMD systems have the ability to handle I/O independently from the front-end processors. This is favourable because the communication between the front-end and back-end systems is avoided. A (specialized) 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. These I/O systems are especially important for very data-intensive applications such as radar and image processing.

A feature that is peculiar to this type of machine is that the processors are sometimes of a very simple bit-serial type, *i.e.*, the processors operate on the data items bit-wise, irrespective of their type. Therefore, operations on integers, for example, 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 twice as fast as a 64-bit result. For floating-point operations a similar situation holds, although the number of cycles required is a multiple of that needed for an integer operation. As the number of processors in this type of system is mostly large (1024 or larger, though the Quadrics Apemille was a notable exception), the slower operation on floating-point numbers can be often compensated for by their number, while the cost per processor is quite low 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, since in both cases operands of only 1–8 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 speed-ups.

At present there are no DM-SIMD systems on the market, but some types of computational accelerator (see Section 10) share many characteristics with DM-SIMD systems that existed until recently. We will briefly discuss properties of these accelerators later.

5. Shared-memory MIMD machines

One subclass of this type of machine was shown in Figure 3.1. 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.

The main problem one faces in shared-memory systems is that of the connection of the CPUs to each other and to the memory. As more CPUs are added, the collective bandwidth to the memory should ideally 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 $O(n^2)$ while increasing the

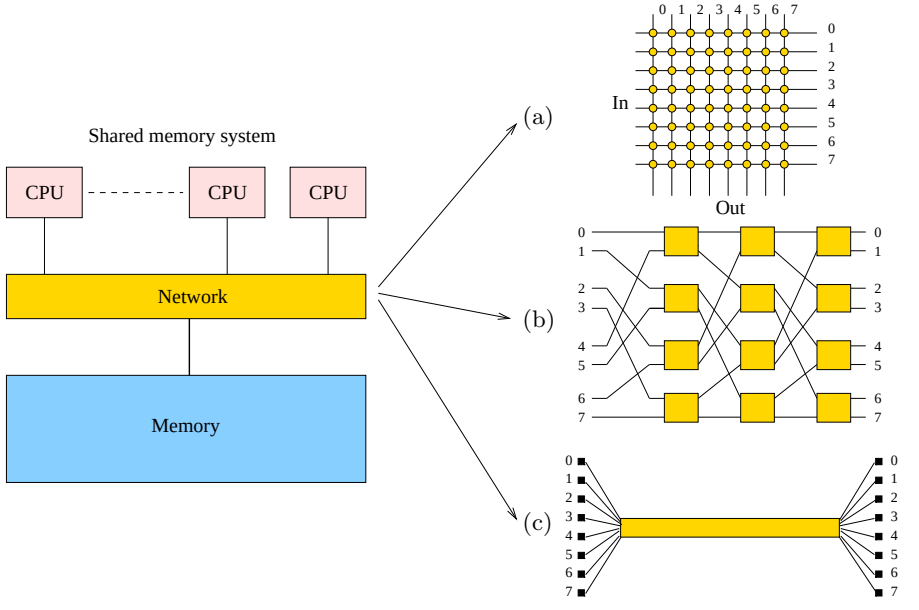


Figure 5.1. Some examples of interconnection structures used in shared-memory MIMD systems. (a) Crossbar, (b) Ω-network, (c) central data bus.

number of processors with $O(n)$. So, various alternatives have been tried. Figure 5.1 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 the 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 computations. Various intricate strategies have been devised using caches associated with the CPUs to minimize the bus traffic. This leads, however, to a more complicated bus structure, which raises the costs. In practice it has proved very hard to design buses that are fast enough, especially where processor speed has been increasing very quickly, imposing an upper bound on the number of processors thus connected, which in practice appears not to exceed 10–20 processors. In 1992, a new standard (IEEE P896) for a fast bus was defined, to connect either internal system components or to connect external systems. This bus, called the Scalable Coherent Interface (SCI), provides a point-to-point bandwidth of 200–1000 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 James, Laundrie, Gjessing and Sohi (1990). However, it has now been effectively superseded by InfiniBand (see Section 11).

A multi-stage crossbar is a network with a logarithmic complexity, and has a structure which is situated somewhere between a bus and a crossbar with respect to potential capacity and costs. The Ω-network, as depicted in Figure 5.1, is an example. Commercially available machines such as the IBM eServer p575, the SGI Altix UV, and many others use(d) a network structure, but a number of experimental machines have also used this or a similar kind of interconnection. The BBN TC2000, which 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 also 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.

Until recently multiprocessor vector processors used crossbars. This was feasible because the maximum number of processors within a system node was small (16 at most). In the late Cray X2 the number of processors had increased so much, however, that it had to change to a logarithmic network topology (see Section 11). Not only does it become harder to build a crossbar of sufficient speed for the larger numbers of processors, but 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 could in principle be arbitrary for any topology. In practice, however, bus-structured machines cannot support vector processors as their speeds would grossly mismatch with any bus that could be constructed at reasonable cost. 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 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 synchronization between tasks is required, and again the interconnection structure is very important here. Some Cray vector processors in the past employed 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 have to synchronize with. This is, however, no longer practised as it is viewed as too costly a feature. The systems may also synchronize via the shared memory. Generally, this is much slower but it can still be acceptable when the synchronization 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 synchronization.

6. 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 6.1. 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 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 characterization of the system

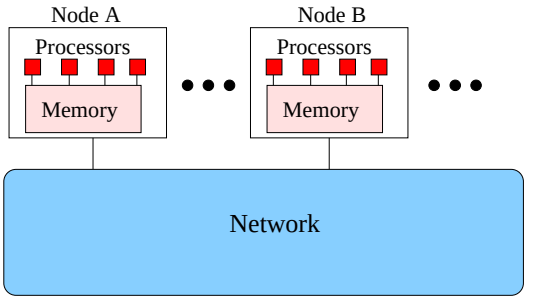


Figure 6.1. Generic diagram of a DM-MIMD machine.

as being of the distributed memory type. The vast majority of all HPC systems today are a variation of the model shown in Figure 6.1.

This type of machine is more difficult to deal with than shared-memory machines and DM-SIMD machines. The latter machines are processor array systems, in which the data structures that are candidates for parallelization are vectors and multi-dimensional arrays laid out automatically on the processor array by 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 the data exchange between processors has to be performed explicitly when using the so-called message-passing parallelization model (which is the case in the vast majority of programs). The initial reluctance to use DM-MIMD machines seems to have decreased. This is partly due to the now existing standard for communication software (Geist *et al.* 1994, Snir *et al.* 1998, Gropp *et al.* 1998) and partly because, at least theoretically, this class of system can outperform all other types of machines.

Alternatively, instead of message passing, a partitioned global address space parallelization model may be used with a programming language such as Unified Parallel C (UPC) or Co-Array Fortran.[3] In this case one still has to know where the relevant data are, 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 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

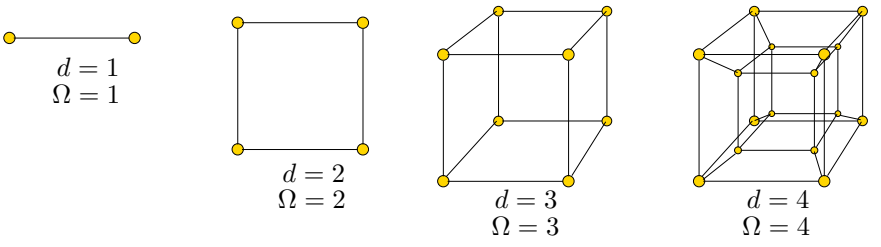
[3] Unified Parallel C home page: upc.gwu.edu/.
Co-Array Fortran home page: www.co-array.org/.

and the speed of the memory should match it), is less important for DM-MIMD machines, because more processors can be configured without the aforementioned bandwidth problems.

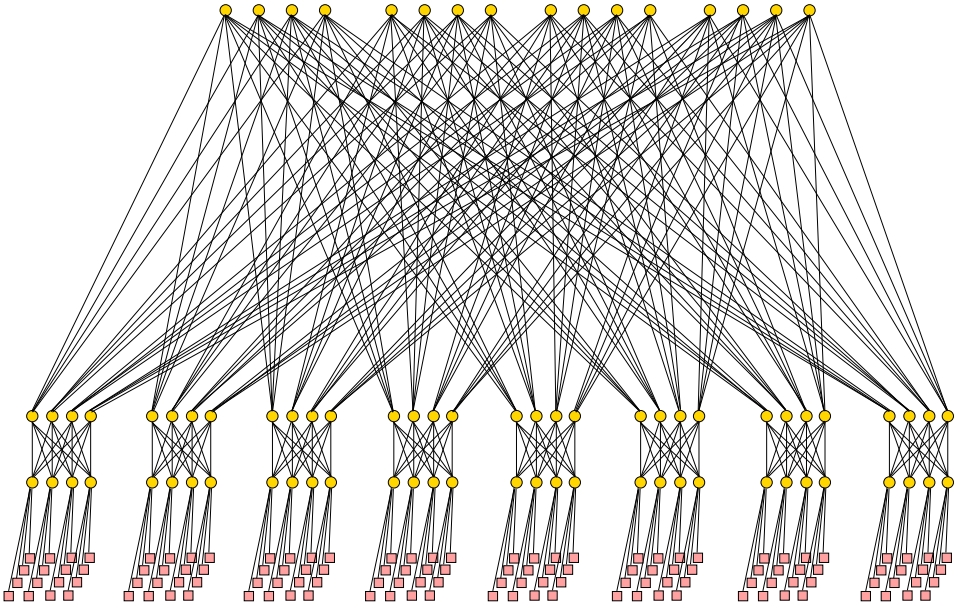
Of course, DM-MIMD systems also have their disadvantages. The communication between processors is slower than in SM-MIMD systems, and so the synchronization overhead, in the case of communicating tasks, is generally orders of magnitude higher than in shared-memory machines. Moreover, the access to data that are not in the local memory belonging to a particular processor have to be obtained from non-local memory (or memories). This again is very slow compared to local data access. When the structure of a problem dictates a frequent exchange of data between processors and/or requires many processor synchronizations, 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 both the topology and the speed of the data paths are also crucial for the practical usefulness of DM-MIMD machines. Again, as in Section 5 on SM-MIMD systems, the richness of the connection structure has to be balanced against cost. 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 6.2(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. Thus, the dimension of the network grows only logarithmically with the number of nodes. In addition, it is theoretically possible to simulate any other topology on a hypercube: trees, rings, 2D and 3D meshes, *etc.* In practice, the exact topology for hypercubes is of secondary importance, because all systems in the market today employ what is called 'wormhole routing', or variants thereof. This means that when a message is to be sent 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 proper data are 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. Further, the network nodes themselves have become quite powerful and, depending on the type of network hardware, may send and reroute message packages in a way that minimizes contention.



(a)



(b)

Figure 6.2. Some frequently used networks for DM machine types. (a) Hypercubes, dimension 1–4. (b) A 128-way fat tree.

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 an N-ary fat tree when the levels towards the roots are N times the number of connections in the level below it. Figure 6.2(b) shows an example of a

quaternary fat tree with a bandwidth in the highest level that is four times that of the lower levels.

A number of massively parallel DM-MIMD systems seem to favour a 2D or 3D 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 (Horie *et al.* 1991). This philosophy has also been followed on IBM's BlueGene systems.

A large fraction of systems in the DM-MIMD class employ crossbars. For relatively small numbers 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 such as 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. Today, the node processors are mostly off-the-shelf RISC processors, sometimes enhanced by vector processors. A problem that is peculiar to DM-MIMD systems is the mismatch of communication versus 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 computation-limited problems into communication-limited problems.

7. ccNUMA machines

As already mentioned in the Introduction, there is a trend to build systems that have a rather small (up to 16) number of processors, tightly integrated in a cluster: a symmetric multiprocessing (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 7.1. Note that in Figure 7.1 all CPUs in a cluster are connected to a common part of the memory. (Figure 7.1 looks functionally identical to Figure 6.1, but 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 explicitly.)

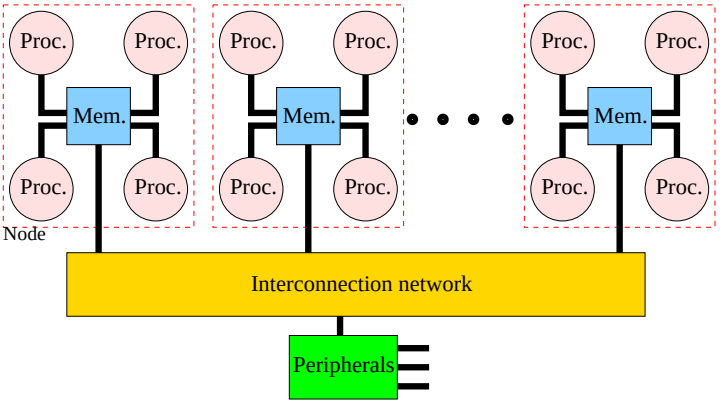


Figure 7.1. Block diagram of a system with a 'hybrid' network: clusters of four CPUs are connected by a crossbar. The clusters are connected by a less expensive network, *e.g.*, a butterfly network.

The most important ways to let the SMP nodes share their memory are S-COMA, which stands for simple cache-only memory architecture, and ccNUMA, or cache coherent non-uniform memory access. Therefore, such systems can be considered as SM-MIMD machines. On the other hand, because the memory is 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 are required that do not reside in the local node's memory, they are retrieved from the memory of the node where they are 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, it must be ensured 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 they are requested by a local CPU. Another way is the *directory memory*, a special part of memory which enables the caches to keep track of all the copies of variables and of their validity.

At present, no commercially available machine uses the S-COMA scheme. In contrast, there are several popular ccNUMA systems (Bull's bullx R422 series, HP Superdome, and SGI Ultraviolet) that are commercially available.

An important characteristic of 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, for which a number of stages of a crossbar might possibly 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 multi-core processors, the ccNUMA phenomenon has also manifested itself in processors with multiple cores: the first-level and second-level caches belong to a particular core and therefore, when another core needs data residing in another core's cache, retrieval is via the complete memory hierarchy of the processor chip. This is typically orders of magnitude slower than when retrieval is from the 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 establish a single system image although the memory is physically distributed.

8. Clusters

The adoption of clusters, *i.e.*, collections of workstations/PCs connected by a local network, has metaphorically 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 in clusters is shown by the creation of the IEEE Task Force on Cluster Computing (TFCC),[4] which reviews the field of cluster computing on a regular basis. Further, books describing how to build and maintain clusters have greatly added to their popularity (Sterling, Salmon, Becker and Savaresse 1999, Spector 2000). As the cluster scene has become a mature and attractive market, large HPC vendors and many start-up companies have entered the field, and offer more-or-less out-of-the-box cluster solutions for those groups who do not want to build their cluster from scratch (hardly anyone builds from scratch these days).

The number of vendors selling cluster configurations has become so large that it is not possible to include all their products in this report. In addition, there is generally great variation in the usage of clusters and they are more often used for *capability computing* (*i.e.*, the system is employed for one or a few programs for which no alternative is readily available in terms of computational capability), while the integrated machines are primarily used for

[4] TFCC home page: www.clustercomputing.org

capacity computing (*i.e.*, the system is employed to the full by use of most of its available cycles by many, often very demanding, applications and users). Traditionally, vendors of large supercomputer systems have learned to provide for capacity computing, as the precious resources of their systems were required to be used as effectively as possible. In contrast, Beowulf clusters typically use the Linux operating system, although a small minority use Microsoft Windows. These operating systems either lack the tools to make use of clusters for capacity computing, or the tools are immature. However, as clusters become on average both larger and more stable, there is a trend to use them as computational capacity servers too, particularly because there is now a plethora of cluster management and monitoring tools. The article by van der Steen (2000) considers some of the aspects that are necessary conditions for this kind of use, such as available cluster management tools and batch systems. The systems assessed then are now quite obsolete, but many of the conclusions are still valid. An important but not very surprising conclusion was that the speed of the network is crucial in all but the most computation-limited applications. Another notable observation was that using compute nodes with more than one CPU may be attractive from the point of view of compactness and (possibly) energy and cooling aspects, but that 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 may also have up to 12 processor cores, this issue has become all the more important, and one might have to choose between capacity-optimized nodes with more processors but less bandwidth per processor core, or capability-optimized nodes with fewer processors per node but 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 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 no longer possible 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 more quickly than for a core in another processor).

Fortunately, there is now 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 those of the networks in integrated systems. These will be discussed in Section 11.

9. Processors

In comparison to 10 years ago, the processor scene has become drastically different. In the period 1980–1990, proprietary processors and, in particular, vector processors were the driving force of supercomputers, but today that role has been taken by common off-the-shelf processors. In fact there are only two companies left that produce vector systems, while all other systems on offer are based on RISC CPUs or x86-like ones. Therefore it is useful to give a brief description of the main processors that populate present supercomputers, and look ahead to the processors that will follow in the coming year. Nonetheless, we will be more conservative in this section than in the description of systems in general, because processors are being turned out at a tremendous pace whereas planning ahead for new generations takes years. We will therefore stick to actually existing components or β versions of a processor.

The RISC processor scene has shrunk 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, which appeared and then disappeared, as they were used in the highly interesting SiCortex systems. Unfortunately SiCortex has had to close down recently, and with it the MIPS processors. In addition, the Itanium processor is no longer used in HPC.

The disappearance of RISC processor families demonstrates a trend that is both worrying and interesting: worrying because diversity in the processor field is decreasing severely and, with it, the choice of systems in this sector. On the other hand there is a trend to enhance systems based on run-of-the-mill processors with special-purpose add-on processors in the form of FPGAs (field programmable gate arrays), or other computational accelerators, because their performance, price level, power consumption, and ease of use have improved to such a degree that they offer attractive alternatives for certain applications.

The very notion of a 'RISC processor' has been eroded somewhat in the sense that 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) has become comparable to those of RISC/EPIC processors. A distinguishing factor is still the mostly much larger set of registers in RISC processors.

Another notable development of the last few years is the placement of multiple processor cores on a processor chip and the introduction of various

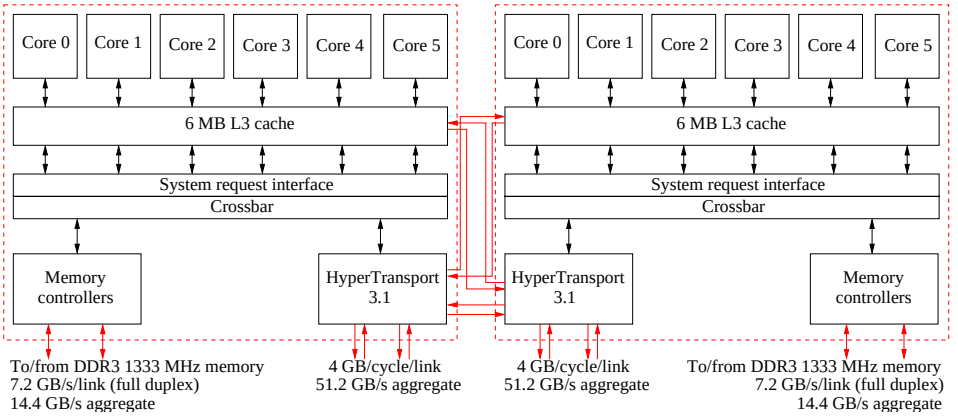


Figure 9.1. Block diagram of an AMD Opteron Magny-Cours processor.

forms of multi-threading. We will discuss these developments for each of the processors separately.

There are two processors one would perhaps expect in this section but are nevertheless not discussed: the Godson 3A and the Itanium Tukwila processors. The first processor, a Chinese one, based on the MIPS architecture, is not available in any machine marketed now or to be marketed in the near future (it is to be succeeded by the Godson 3B early next year). The newest Itanium processor no longer plays a role in the HPC scene and is therefore also omitted.

9.1. AMD Magny-Cours

All AMD processors are clones with respect to Intel's x86 instruction set architecture. The 12-core Opteron variant called 'Magny-Cours', made available in March 2010, is no exception. It is built with a feature size of 45 nm, and in fact the chip is a package containing two modified 6-core Istanbul chips running at a maximum of 2.3 GHz in the 6176 SE variant. The two chips are connected through 16-bit HyperTransport 3.1 links to each other's L3 caches with a single-channel speed of 12.8 GB/s, as shown in Figure 9.1.

The clock frequencies of the various parts of the chip are independent and different: while the processor operates at a speed of of 2.3 GHz, the HyperTransport links run at 3.2 GHz and the four memory buses (two per 6-core chip) run at only 1.8 GHz, thus limiting the maximum bandwidth between memory and the chip to only 28.8 GB/s. AMD has made this choice to limit the power consumption, although the new chips accommodate DDR3 memory at a speed of 1333 MHz, which means that the bandwidth could potentially have been 42.7 GB/s. As in the Istanbul processor, the Magny-Cours processor exploits the 'HT Assist' function. HT Assist sets aside 1 MB in the L3 cache, which contains the position and status of the cache

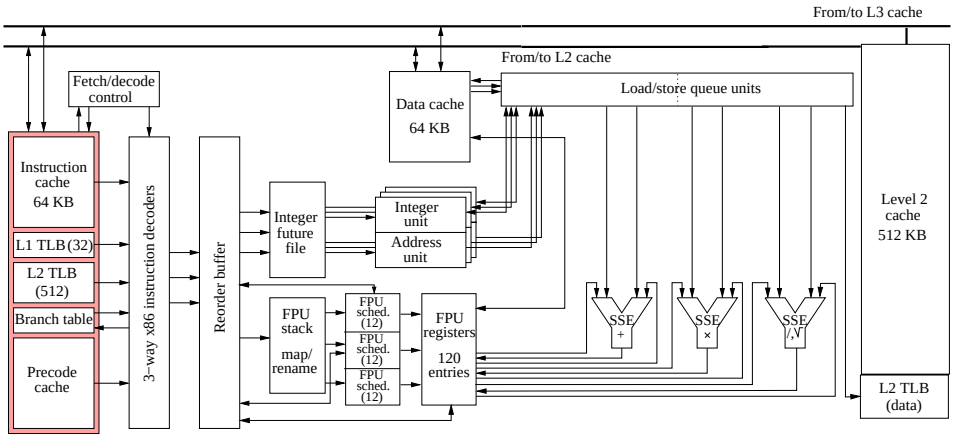


Figure 9.2. Block diagram of an AMD Magny-Cours processor core.

lines in use on the chip. In this way the change in status of cache variables does not have to be broadcast to all cores, but can simply be read from this part of the L3 cache, thus lowering the traffic in the interconnection fabric significantly. This set-up is in fact an implementation of cache coherence via directory memory, as explained in Section 7. Comparison experiments with the earlier Shanghai processor have shown that HT Assist can be highly beneficial thanks to more bandwidth being available for operand transfer. Because the number of cores has doubled with regard to the Istanbul processor, the HT Assist function has become all the more important.

Although they use the x86 instruction set, the AMD processors can be regarded as fully fledged RISC processors: they support out-of-order execution, have multiple floating-point units, and can issue up to nine instructions simultaneously. A block diagram of a processor core is shown in Figure 9.2. It is in effect identical to the Istanbul processor core. The six cores on the chip are connected by an on-chip crossbar. It also connects to the memory controller and, as stated earlier, to its companion chip and other processors on the board via HyperTransport.

The figure shows that a core has three pairs of integer execution units and address generation units that, via an 32-entry integer scheduler, take care of the integer computations and address calculations. Both the integer future file and the floating-point scheduler are fed by the 72-entry reorder buffer, which receives the decoded instructions from the instruction decoders. The decoding in the Opteron core has become more efficient than in the earlier processors: SSE instructions now decode into 1 micro-operation (μ-op) as do most integer and floating-point instructions. In addition, a piece of hardware called the 'sideband stack optimizer' has been added (not shown in the figure), that takes care of the stack manipulations in the instruction stream,

thus making instruction reordering more efficient, and thereby increasing the effective number of instructions per cycle.

The floating-point units allow out-of-order execution of instructions via the FPU stack map and rename unit. It receives floating-point instructions from the reorder buffer and reorders them if necessary, before handing them over to the FPU scheduler. The floating-point register file is 120 elements deep, on a par with the number of registers available in RISC processors.[5]

The floating-point part of the processor contains three units: floating add and multiply units that can work in superscalar mode, resulting in two floating-point results per clock cycle and a unit handling 'miscellaneous' operations, such as division and square-root. Because of the compatibility with Intel's processors, the floating-point units are also able to execute Intel SSE2/3 instructions and AMD's own 3DNow! instructions. However, there is the general problem that such instructions are not directly accessible from higher-level languages such as Fortran 90 or C/C++. Both instruction sets were originally meant for massive processing of visualization data, but are increasingly used for standard dense linear algebra operations.

Due to the shrinkage of technology to 45 nm, each core can harbour a secondary cache of 512 KB. Because of the accommodation of DDR3 memory at a bus speed of 1333 MHz, the total bandwidth (but with the limitation of the 1.8 GHz memory interface) a channel transports is 7.2 GB/s or 14.4 GB/s per 6-core chip.

AMD's HyperTransport is derived from licensed Compaq technology and similar to that employed in HP/Compaq's former EV7 processors. It allows for 'glueless' connection of several processors to form multiprocessor systems with very low memory latencies. The Magny-Cours processor uses fourth-generation HyperTransport 3.1, which transfers 12.8 GB/s at 16-bit wide per unidirectional link. The HyperTransport interconnection possibility makes it highly attractive for building SMP-type clusters or to couple computational accelerators (see Section 10) directly to the same memory as the standard processor.

9.2. IBM POWER6

In the systems that feature IBM's supercomputer line, the p575 series, the nodes contain the POWER6 chip as the computational engine. This will change shortly and therefore we will also discuss the POWER7 processor in Section 9.3, but at the time of writing, the POWER6 is still the processor for IBM's high-end HPC systems. As compared to its predecessor, the POWER5+, there are quite some differences, both in the chip lay-out and in the two cores that reside on a chip. Figure 9.3 shows the layout of the

[5] For x86 instructions, 16 registers in a flat register file are present instead of the register stack typical of Intel architectures.

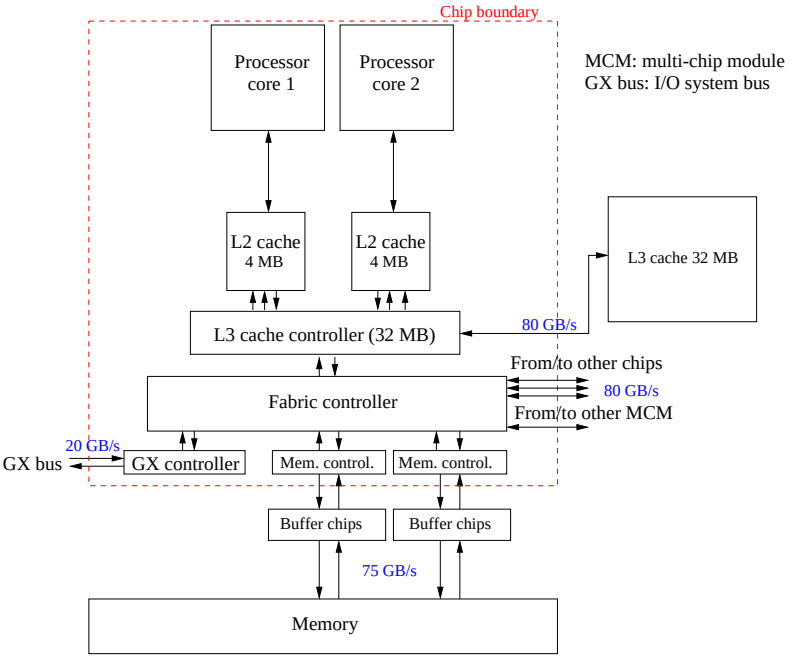


Figure 9.3. Diagram of the IBM POWER6 chip layout.

cores, caches, and controllers on the chip. Already there are significant changes: instead of a 1.875 MB shared L2 cache, each core now has its own 4 MB 8-way set-associative L2 cache that operates at half the core frequency. In addition, there are two memory controllers that connect via buffer chips to the memory and, depending on the number of buffer chips and data widths (both are variable), can have a data read speed ≤ 51.7 GB/s and a write speed of ≤ 25.85 GB/s, *i.e.*, with a core clock cycle of 4.7 GHz up to 11 B/cycle for a memory read and half of that for a memory write. Furthermore, the separate buses for data and coherence control between chips are now unified with a choice of both kinds of traffic, occupying 50 % of the bandwidth, or 67 % for data and 33 % for coherence control. The off-chip L3 cache has shrunk from 36 to 32 MB. It is a 16-way set-associative victim cache that operates at 1/4 of the clock speed.

please check

Further, the core has changed considerably: it is depicted in Figure 9.4. The clock frequency has increased from 1.9 GHz in the POWER5+ to 4.7 GHz for the POWER6 (water-cooled version), an increase of almost a factor of 2.5, while the power consumption has stayed in the same range as that of the POWER5+. This has partly come about by a technology shrink from a 90 nm to 65 nm feature size. It also means that some features of the POWER5+ have disappeared. For instance, the POWER6 largely employs static instruction scheduling, except for a limited amount

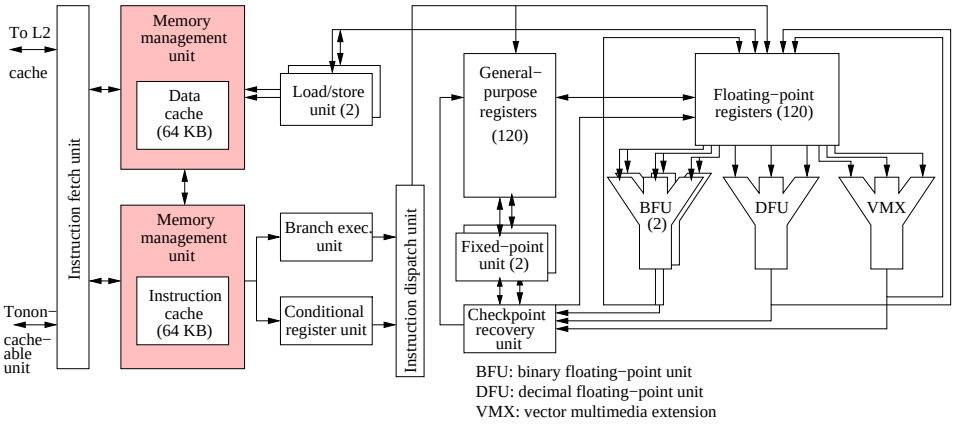


Figure 9.4. Block diagram of the IBM POWER6 core.

of floating-point instruction scheduling, because some of these can be fitted into empty slots left by division and square-root operations. The circuitry required for dynamic instruction scheduling that could thus be removed has, however, been replaced by new units. Besides the two fixed-point units (FXUs) and the two binary floating-point units (BFUs) that were already present in the POWER5+, there is now a decimal floating-point unit (DFU) and a vector multimedia extension (VMX) unit, akin to Intel's SSE units for handling multimedia instructions. In fact, the VMX unit is inherited from the IBM PowerPC's AltiVec unit. The DFU is IEEE 754R compliant. It is obviously for financial calculations and is hardly of consequence for HPC use. Counting only the operations of the BPUs both executing fused multiply–adds (FMAs), the theoretical peak performance in 64-bit precision is 4 flop/cycle or 18.8 Gflop/s/core. A checkpoint recovery unit has been added that is able to catch faulty FXU and FPU (both binary and decimal) instruction executions and reschedule them for retrial. Because of the large variety of functional units, a separate instruction dispatch unit ships the instructions that are ready for execution to the appropriate units, while a significant part of instruction decoding has been pushed into the instruction fetch unit, including updating the branch history tables.

The BFUs not only execute the usual floating-point instructions such as add, multiply and FMA, but also take care of division and square-root operations. A new phenomenon is that integer divide and multiply operations are also executed by the BFUs, again saving on circuitry and therefore power consumption. In addition, these operations can be pipelined in this way and yield a result every two clock cycles.

The L1 data cache has been doubled in comparison to the POWER5+ and is now 64 KB like the L1 instruction cache. Both caches are 4-way set-associative.

The simultaneous multi-threading (SMT) that was already present in the POWER5+ has been retained in the POWER6 processor, and has been improved by a higher associativity of the L1 I and D caches and a larger dedicated L2 cache. Also, instruction decoding and dispatch are dedicated for each thread. By using SMT the cores are able to keep two process threads at work at the same time. The functional units get instructions for the functional units from any of the two threads, whichever is able to fill a slot in an instruction word that will be issued to the functional units. In this way a larger fraction of the functional units can be kept busy, improving the overall efficiency. For very regular computations a single thread (ST) mode may be better, because in SMT mode the two threads compete for entries in the caches, which may lead to trashing in the case of regular data access. Note that SMT is somewhat different from the 'normal' way of multi-threading. In this case a thread that stalls for some reason is stopped and replaced by another process thread that is awoken at that time. Of course this takes some time, which must be compensated for by the thread that has taken over. This means that the second thread must be active for a fair number of cycles (preferably a few hundred cycles at least). SMT does not have this drawback but scheduling the instructions of both threads is quite complicated, especially where only very limited dynamic scheduling is possible.

Because of the much higher clock cycle, and the fact that the memory DIMMs are attached to each chip, it is no longer possible to maintain perfect SMP behaviour within a 4-chip node, *i.e.*, it matters whether data are accessed from a chip's own memory or from the memory of a neighbouring chip. Although the data are only one hop away, there is a ccNUMA effect that one has to be aware of in multi-threaded applications.

9.3. IBM POWER7

As remarked earlier, at this moment IBM is not yet offering HPC systems with the POWER7 inside. Hitachi is already offering a variant of its SR16000 system with the POWER7 processor and IBM is expected to follow shortly. So, it is appropriate to discuss this chip. Figure 9.5 shows the layout of the cores, caches, and memory controllers on the chip. The technology from which the chips are built is identical to that of the POWER6 45 nm Silicon-On-Insulator, but in all other respects the differences from the former generation are large. Firstly, the number of cores has quadrupled. Further, the memory speed has increased from DDR2 to DDR3 via two on-chip memory controllers. As in earlier POWER versions, the inbound and outbound bandwidths 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.

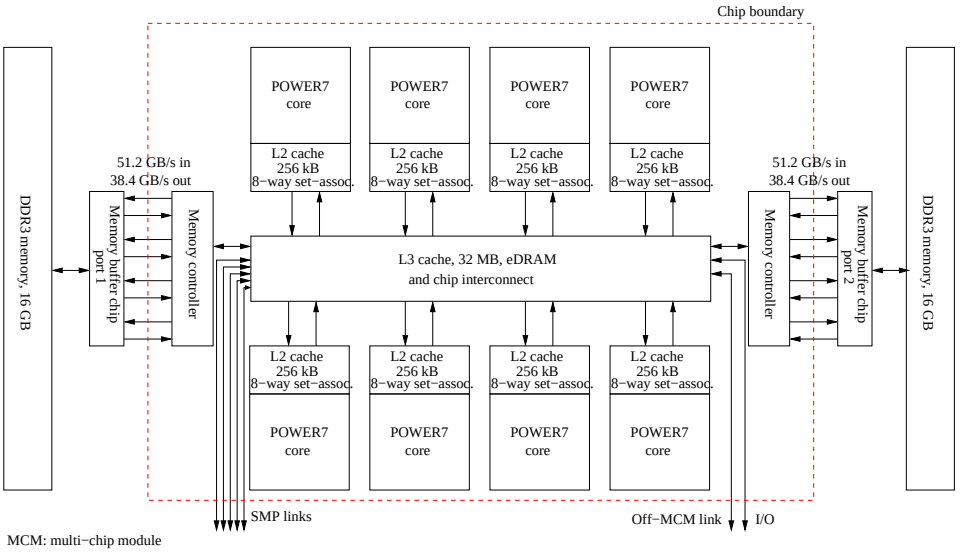


Figure 9.5. Diagram of the IBM POWER7 chip layout.

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 3.5–3.86 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 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 is now on-chip, the latency is considerably lower (about a factor of 6). The L3 cache communicates with the L2 caches, which 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 support SMP operations of up to 32 sockets.

Further, at the core level there are many differences from its predecessor. A single core is depicted in Figure 9.6. To begin with, the number of floating-point units is doubled to four, each capable of a fused multiply–add

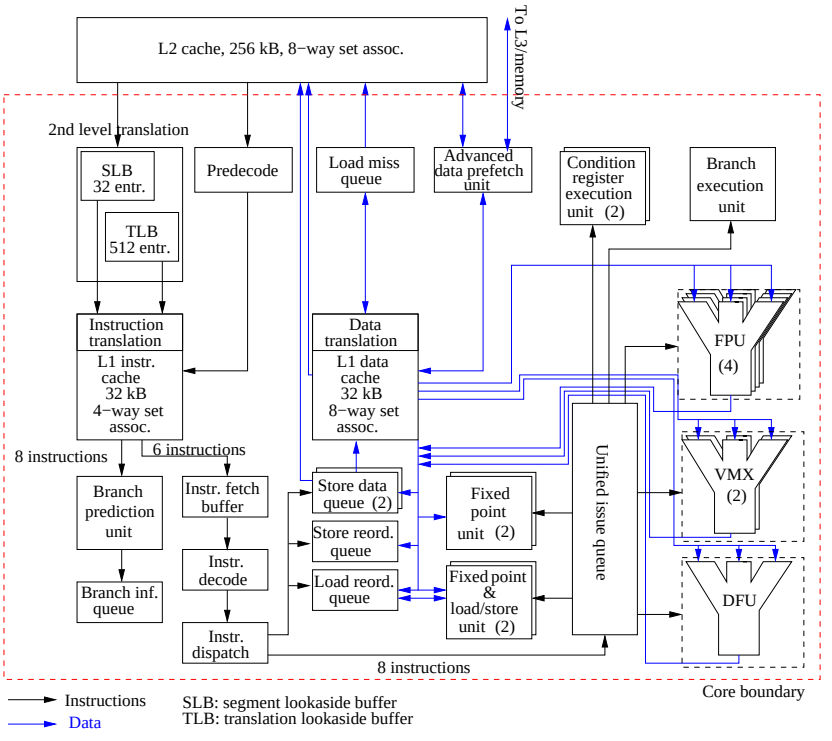


Figure 9.6. Block diagram of the IBM POWER7 core.

operation per cycle. Assuming a clock frequency of 3.86 GHz, this means that a peak speed of 30.88 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 9.6, 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, which is also employed in the PowerPC processors. There are also similarities to the POWER6 processor: the core contains a 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 four SMT threads instead of two. This will be very helpful for the large numbers of functional units to be kept busy. Eight instructions can be taken from the L1 instruction cache. The instruction decode unit can handle six instructions simultaneously while eight instructions can be dispatched every cycle to the various functional units.

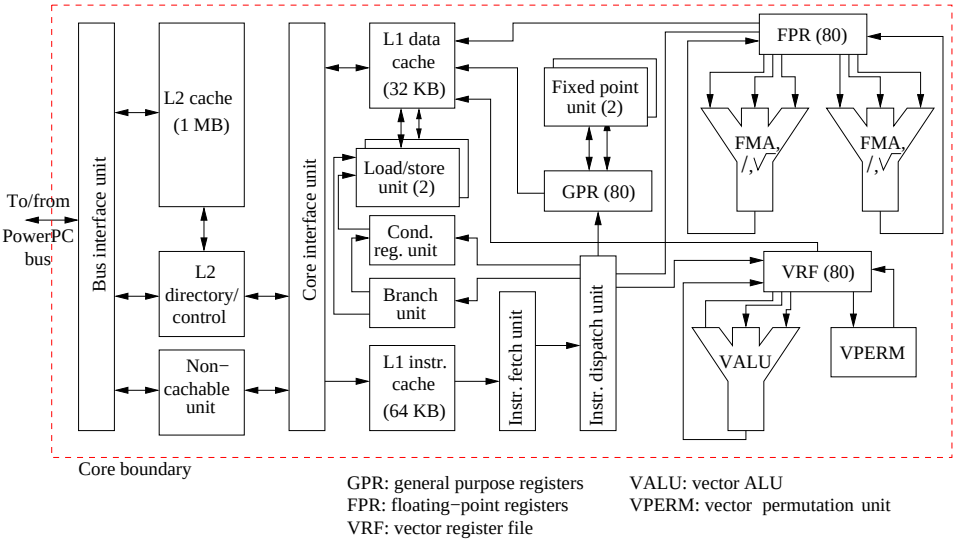


Figure 9.7. Block diagram of the IBM PowerPC 970MP core.

The POWER7 core has elaborate power management features that reduce the power usage for parts that are idle for some time. There are two power-saving modes: *nap* mode and *sleep* mode. In the former the caches and translation lookaside buffers (TLBs) stay coherent to re-activate quickly. In sleep mode, however, the caches are purged and the clock turned off. Only the minimum 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.

9.4. IBM PowerPC 970MP processor

A number of IBM systems are built from JS21 blades, the largest being the Mare Nostrum system at the Barcelona Supercomputing Centre. On these blades a variant of the large IBM PowerPC processor family is used, the dual core PowerPC 970MP. It is a series of dual-core processors, the fastest of which has a clock cycle of 2.2 GHz. A block diagram of a processor core is given in Figure 9.7.

A peculiar trait of the processor is that the L1 instruction cache is twice as large as the L1 data cache: 64 KB versus 32 KB. This is explained partly by the fact that up to 10 instructions can be issued every cycle to the various execution units in the core. Apart from two floating-point units that perform the usual dyadic operations, there is an AltiVec vector facility with a separate 80-entry vector register file, a vector ALU that performs (fused) multiply–add operations, and a vector permutation unit that attempts to order operands such that the vector ALU is used optimally. The vector

unit was designed for graphics-like operations, but works quite nicely on data for other purposes as long as access is regular and the operand type agrees. Theoretically, the speed of a core can be 13.2 Gflop/s/core when both FPUs turn out the results of a fused multiply–add and the vector ALU does the same. One PowerPC 970MP should therefore have a theoretical peak performance of 26.4 Gflop/s. The floating-point units also perform square-root and division operations.

Apart from the floating-point and vector functional units, two integer fixed-point units and two load/store units are present in addition to a conditional register unit and a branch unit. The latter uses two algorithms for branch prediction that are applied according to the type of branch to be taken (or not). The success rate of the algorithms is constantly monitored. Correct branch prediction is very important for this processor as the pipelines of the functional units are quite deep: from 16 for the simplest integer operations to 25 stages in the vector ALU. So, a branch miss can be very costly. The L2 cache is integrated and has a size of 1 MB. To keep the load/store latency low, hardware-initiated prefetching from the L2 cache is possible and 8 outstanding L1 cache misses can be tolerated. The operations are dynamically scheduled and may be out of order. In total 215 operations may be in flight simultaneously in the various functional units, also due to the deep pipelines.

The two cores on a chip have common arbitration logic to regulate the data traffic from and to the chip. There is no third-level cache between the memory and the chip on the board housing them. This is possible because of the moderate clock cycle and the rather large L2 cache.

9.5. IBM BlueGene processors

In the last few years two BlueGene types of systems have become available: the BlueGene/L and the BlueGene/P, the successor of the former. Both feature processors based on the PowerPC 400 processor family.

BlueGene/L processor

This processor is in fact a modified PowerPC 440 processor, which is made especially for the IBM BlueGene family. It runs at a speed of 700 MHz. The modification lies in tacking on floating-point units (FPUs) that are not part of the standard processor but can be connected to the 440's APU bus. Each FPU contains two floating-point functional units capable of performing 64-bit multiply–adds, divisions and square-roots. Consequently, the theoretical peak performance of a processor core is 2.8 Gflop/s. Figure 9.8 shows the embedding of two processor cores on a chip. As can be seen from the figure, the L2 cache is very small: only 2 KB divided in a read and a write part. In fact it is a prefetch and store buffer for the rather large L3 cache.

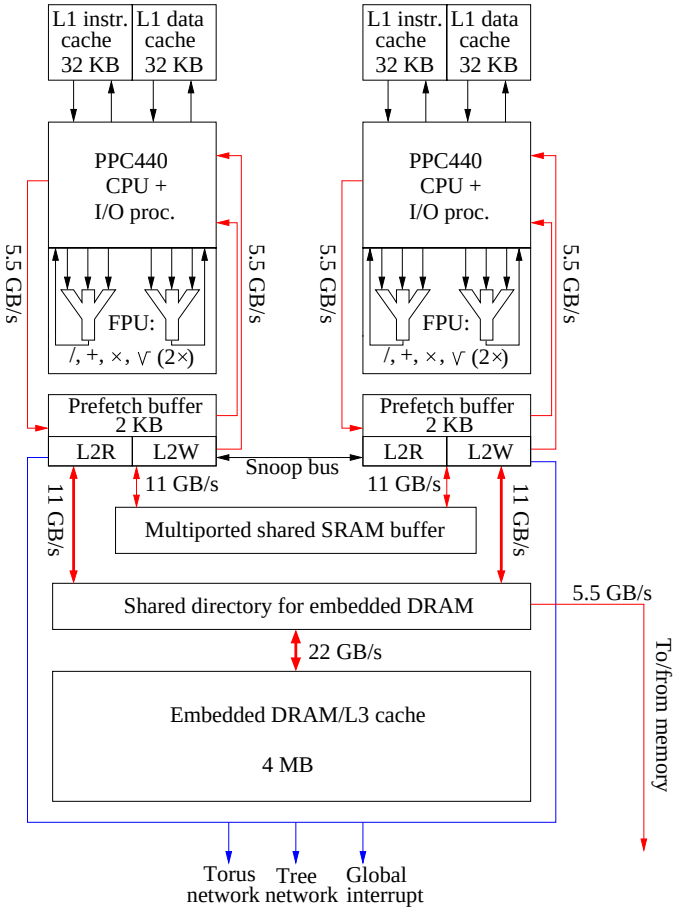


Figure 9.8. Block diagram of an IBM BlueGene/L processor chip.

The bandwidth to and from the prefetch buffer is high, 16 B/cycle to the CPU and 8 B/cycle to the L2 buffer. The memory resides off-chip with a maximum size of 512 MB. The data from other nodes are transported through the L2 buffer, bypassing the L3 cache in the first instance.

BlueGene/P processor

Like the BlueGene/L processor, the BlueGene/P processor is based on the PowerPC core – the PowerPC 450 in this case – at a clock frequency of 850 MHz and with floating-point enhancements similar to those applied to the PPC 440 in the BlueGene/L. The BlueGene/P node contains 4 processor cores, which brings the peak speed to 13.6 Gflop/s/node. The block diagram in Figure 9.9 shows some details. As can be seen from the figure, the structure of the core has not changed much with respect to the BlueGene/L. The relative bandwidth from the L2 cache has been maintained: 16 B/cycle

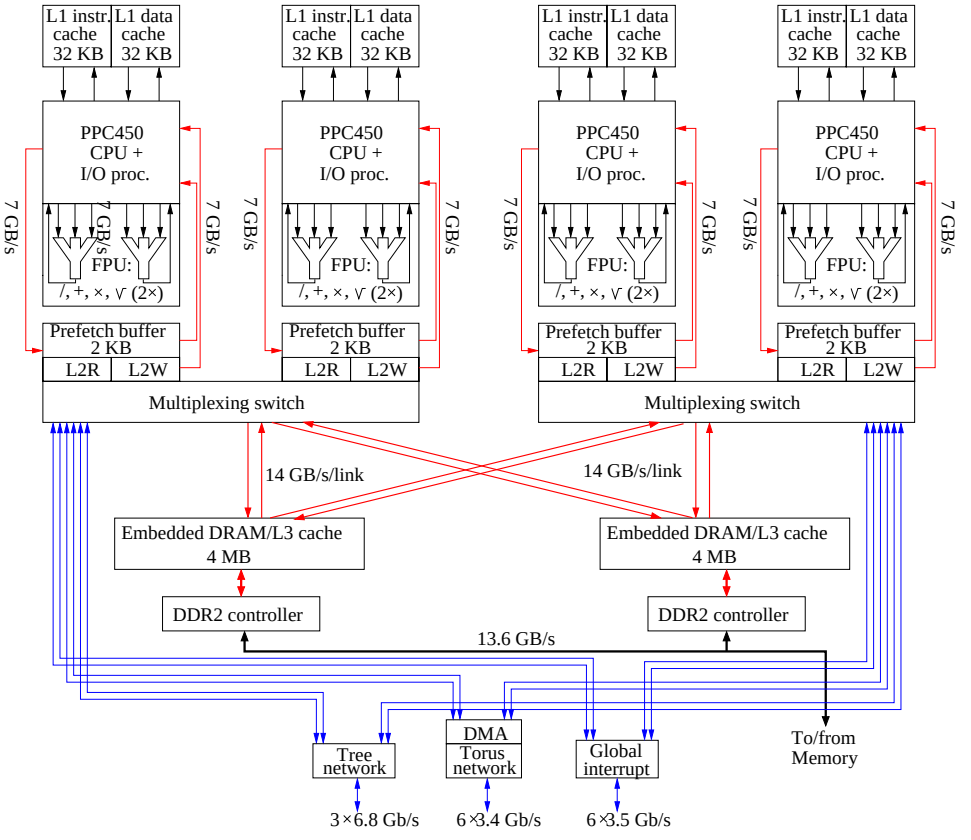


Figure 9.9. Block diagram of an IBM BlueGene/P processor chip.

for reading and 8 B/cycle for writing. In contrast to the BlueGene/L, the cores operate in SMP mode through multiplexing switches that connect pairs of cores to the two 4 MB L3 embedded DRAM chips. So, the L3 size has doubled. Also, the memory per node has increased to 2 GB from 512 MB.

9.6. Intel Xeon

Two variants of Intel's Xeon processors are employed at present in HPC systems (clusters as well as integrated systems): the Nehalem EX, officially the X7500 chip series, and the Westmere EP, officially the X5600 series. Although there is a great deal of common ground, they are sufficiently different that we can discuss each processor separately.

Nehalem EX

The Nehalem EX became available in March 2010 or, more formally, the X7500 series of processors can be regarded as a heavy-duty server extension

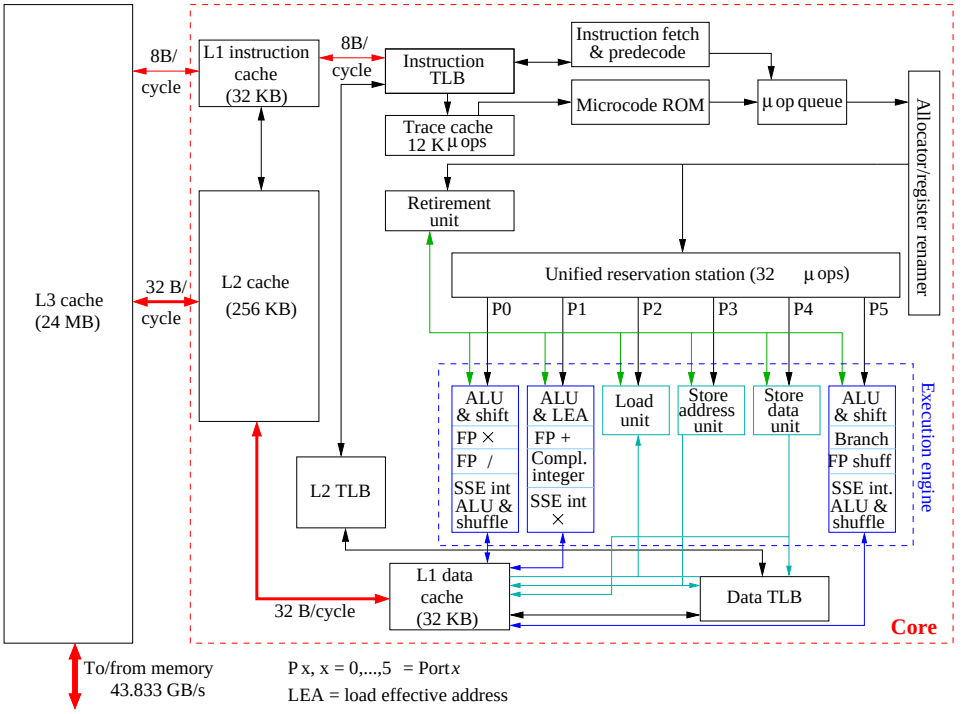


Figure 9.10. Block diagram of an Intel Nehalem processor core.

of the earlier Nehalem EP (X5500) processor. As such, it has twice the number of cores – eight – and the number of QPI links has also doubled to four, to enable the connection to other processors on a server board. In addition the bandwidth is raised, because the number of memory interfaces has increased from three to four. As the bandwidth per memory channel is 10.833 GB/s, the aggregate bandwidth of the chip is 43.3 MB/s. The Nehalem EX is employed in SGI's Altix UV systems and a variety of products from cluster vendors. Below we show the block diagram of the processor core, which is in fact identical to that of the Nehalem EP and built in the same 45 nm technology. A block diagram of the core is given in Figure 9.10, while the layout of an 8-core chip is shown in Figure 9.11.

To stay backwards-compatible with the x86 (IA-32) instruction set architecture, which comprises a CISC instruction set, Intel developed a mode in which these instructions are split into 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 paid for this much more efficient instruction set is an extra decoding stage. Branch prediction has been improved and a second-level TLB cache has been added.

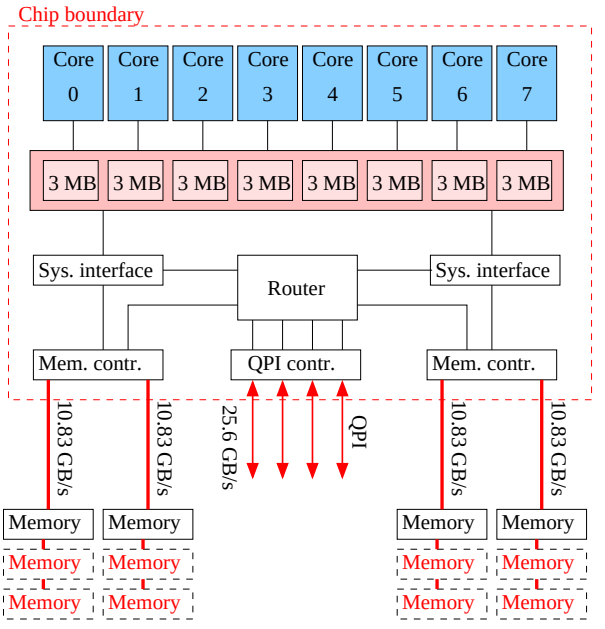


Figure 9.11. Diagram of a Nehalem EX processor.

As in the earlier core architecture, four 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. In the Nehalem, two additional μ-ops can be fused in comparison to the core architecture.

As can be seen in Figure 9.10, the processor cores have an execution trace cache which holds partly decoded instructions of former execution traces that can be drawn upon, thus forgoing the instruction decode phase, which 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 9.10 shows that ports 0 and 5 drive two identical integer ALUs as well as integer SSE units. Ports 0, 1 and 5 take care of the various floating-point operations.

The two integer arithmetic logic units (ALUs) at ports 0 and 5 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 floating-point units also contain additional units that execute the Streaming SIMD Extensions 4 (SSE4) repertoire of instructions, an instruction set of more than 190 instructions, that was initially meant for vector-oriented operations such as those in multimedia, and 3D visualization applications, but is also an advantage for regular vector

operations as occur in dense linear algebra. The length of the operands for these units is 128 bits. The Intel compilers have the ability to address the SSE4 units. This enables in principle much higher floating-point performance. Ports 2, 3 and 4 serve the load unit, the store address unit, and the store data unit, respectively.

A notable enhancement that cannot be shown in the figures is that the Nehalem (again) supports multi-threading, much in the style of IBM's simultaneous multi-threading, and is called 'hyperthreading' by Intel. 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, speed-ups of more than 30 % for some codes have been observed with hyperthreading. Another feature that cannot be shown is the so-called Turbo mode. This means that the clock cycle can be raised from its nominal speed (2.91 GHz for the fastest variant) by steps of 133 MHz to over 3 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 L1 caches have the same size as in the Nehalem's predecessor, but the L2 cache is much smaller: 256 KB instead of 6 MB. It is much faster, however, and able to deliver requested data in 10 cycles or less. The Nehalems feature a common L3 cache that is used by all eight cores in the EX version. Each core has its own section of 3 MB, but when data are not found in the section of a core the other sections can be searched for the missing data item(s). 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 therefore one need not search them.

In Figure 9.11 it can be noticed that, as well as the first bank of memory of $\leq$ 32 GB, a second and third bank are also depicted, represented by dashed boxes. This means that it is indeed possible to have up to 96 GB of memory/processor. However, this can only be done at the expense of the memory bus speed: for one bank it is 1333 MB/s, for two banks 1066 MB/s, and for three banks only 800 MB/s. So, the latter two options may be chosen, for instance, for database systems that benefit from a large memory that need not be at the very highest speed. For HPC purposes, however, configurations with only one memory bank/processor will usually be offered.

Westmere EP

The Westmere EP (X5600 series) is a 32 nm technology shrink of the Nehalem EP chip. The smaller feature size is used to place 6 cores on a die. The fastest variant, the X5690, has a clock cycle of 3.46 GHz at 130 W. The structure of the core is the same as in the Nehalem processors (see Figure 9.10) but there are slight differences in the instruction set for the

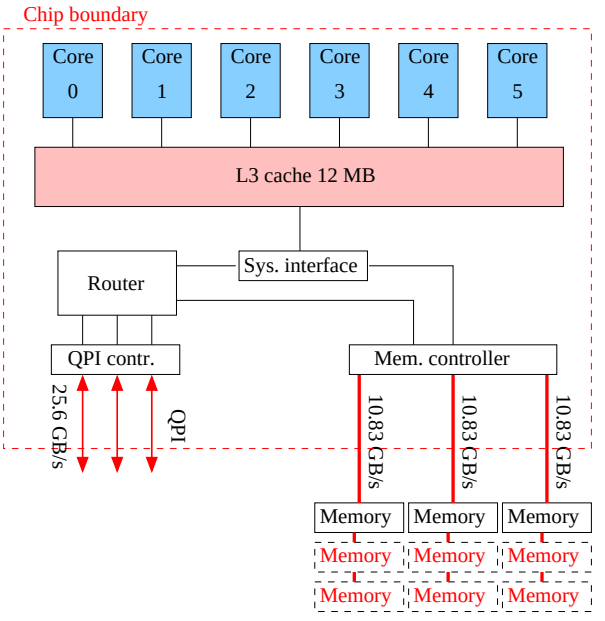


Figure 9.12. Diagram of a Westmere EP processor.

Advanced Encryption Standard (AES). The new instructions, among which is a carry-less multiplication, are said to speed up the en-/decryption rate by a factor of three. Also, the Westmere EP supports the use of 1 GB pages. The packaging on the chip is, apart from the number of cores, identical to that of the Nehalem EP chip with exception of the shared L3 cache. The size of this is halved from 24 MB to 12 MB. The chip layout is depicted in Figure 9.12.

9.7. The SPARC processors

Since Sun was taken over by Oracle all processor development has been shelved. The development of the SPARC processor architecture is now in the hands of Fujitsu, now proceeding with its own SPARC64 implementation. Fujitsu/Siemens markets its HPC servers based on the latter processor. Below we discuss the current SPARC chip, which is commercially available in the Fujitsu machines. Although a follow-on processor, the SPARC64 VIII, seems ready for incorporation in Japan's 10 petaflop/s system, which is currently being built, we only discuss the SPARC64 VII here as this is the one that is commercially available. At present it is not known when its successor will appear on the open market.

The SPARC64 VII is, obviously, Fujitsu's seventh generation of the processor. Of course, the processor must be able to execute the SPARC instruction set but the processor internals are rather different from Sun's late

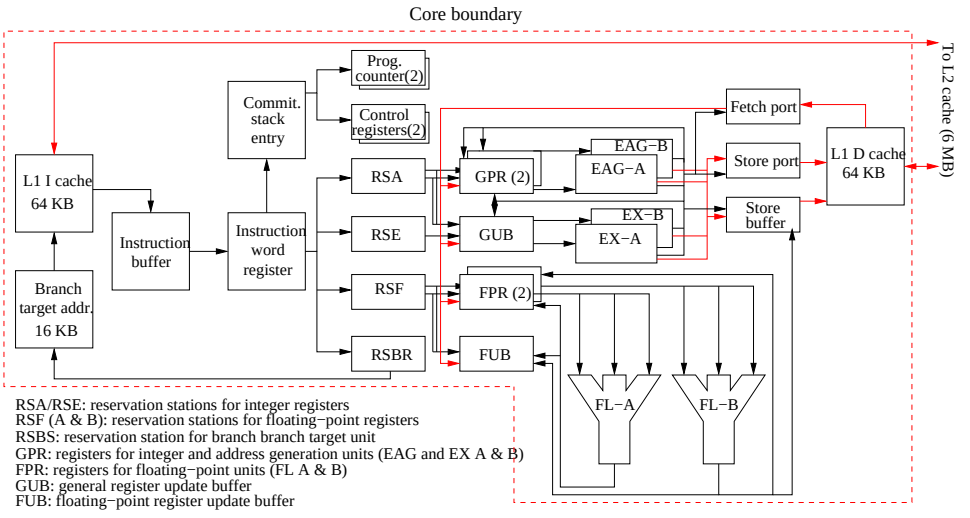


Figure 9.13. Block diagram of a Fujitsu SPARC64 VII processor core.

implementations. Figure 9.13 shows a block diagram of one core of the quad-core SPARC64 VII.

In fact, the core architecture has not changed from the SPARC64 VI, but thanks to the decrease in the feature size from 90 nm to 65 nm, four cores can now be placed on a chip while the highest available clock frequency is raised from 2.4 GHz to 2.52 GHz.

The L1 instruction and data caches are 64 KB, twice as small as in the SPARC64 VI core, and both 2-way set-associative. This decrease in size is somewhat surprising and probably due to the technology shrink to 65 nm feature size. There is also an instruction buffer (IBF), that contains up to 48 4-byte instructions and continues to feed 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 two general register files serve both the two address generation units EAG-A, and EAG-B, and the integer execution units EX-A and EX-B. The latter two are not equivalent: only EX-A can execute multiply-and-divide instructions. There are also two floating-point register files (FPR), that feed the two floating-point units FL-A and FL-B. These units are different from those of Sun in that they are able to execute fused multiply–add instructions, as is also the case in the POWER and Itanium processors. Consequently, a maximum of four floating-point results/cycle can be generated. In addition, FL-A and FL-B also perform divide and square-root operations, in contrast to the SPARC4+, which has a separate unit for these operations. Because of their

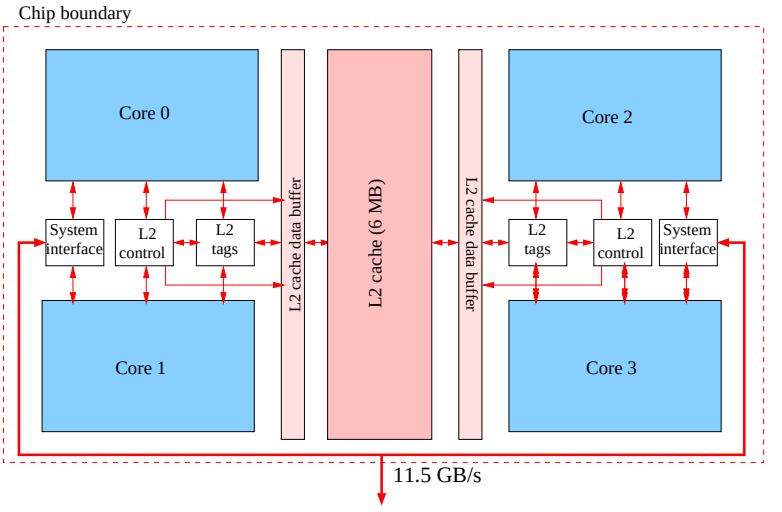


Figure 9.14. Block diagram of a Fujitsu SPARC64 VII processor chip. Four cores share the L2 cache.

iterative nature, the divide and square-root operations are not pipelined. 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.

The dispatch of instructions via reservation stations, each of which can hold 10 instructions, gives the opportunity for speculative dispatch: *i.e.*, dispatching instructions for which the operands are not yet ready at the moment of dispatch but will be by the time the instruction is actually executed. The assumption is that it results in a more even flow of instructions to the execution units.

The SPARC64 VII does not have a third-level cache, but on chip there is a large (6 MB) unified L2 12-way set-associative write-through cache that is shared by the 4 cores in a processor, as can be seen in Figure 9.14. Note that the system bandwidth is the highest available. For the lower-end systems this bandwidth is about 8 GB/s.

The memory management unit (not shown in Figure 9.13) contains separate sets of translation lookaside buffers (TLBs) 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.

What cannot be shown in the diagrams is that, like the IBM and Intel processors, the SPARC VII is dual-threaded per core. The type of multi-threading is similar to that found in the Intel processors, and is called 'simultaneous multi-threading', differing from the type of multi-threading present

in the IBM processors but with the same name. At this moment the highest clock frequency SPARC64 available is 2.52 GHz. As already remarked, the floating-point units are capable of a fused multiply–add operation, like the POWER and Itanium processors, and so the theoretical peak performance is at present 10.08 Gflop/s/core and consequently 40.3 Gflop/s/processor.

10. Computational accelerators

In the last few years computational accelerators have emerged, and now have a firm foothold. They come in various forms, of which we will discuss some general characteristics. Accelerators are not a new phenomenon: in the 1980s, for instance, Floating Point Systems sold attached processors such as the AP120-B with a peak performance of 12 Mflop/s, easily ten times faster than the general-purpose systems they were connected to. Further, the processor array machines described in Section 4 could be regarded as accelerators for matrix-oriented computations in their time. A similar phenomenon is recurring now. HPC users are never 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 with this wish, and at present there is a fair number of products that, when properly deployed, can deliver significant performance gains.

The scene is roughly divided into 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 have set a trend in high-performance computing, namely that the processing units should be diversified according to their abilities – not unlike the occurrence of different functional units within a CPU core.[6] In a few years this will lead to hybrid systems incorporating 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 uncertain if or how this will happen.

When speaking of special-purpose processors, *i.e.*, computational accelerators, one should realize that they are indeed good at some specialized computations but totally unable to perform others. So, not all applications can benefit from them, and of those that can, not all can benefit to the

[6] 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.

same degree. Furthermore, using accelerators effectively is not at all trivial. Although the software development kits (SDKs) for accelerators have recently improved enormously, for many applications it is still a challenge to obtain significant speed-up. An important factor in this 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 recognized by several software companies, such as 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 not used optimally), and, in the case of RapidMind, of Cell processors.

There is one other important consideration that makes accelerators popular: in comparison to general-purpose CPUs they are all very power-efficient, sometimes by orders of magnitude when expressed in flop/Watt. Of course, they will do only part of the work in a complete system, but the power savings can still be considerable, which is very attractive today.

We will now proceed to discuss the three classes of accelerators mentioned above. It must be realized, though, that developments in this field are extremely rapid, and therefore the information given here will become obsolete very fast and hence could be of an approximate nature.

10.1. Graphical processing units

Graphics processing is characterized by doing the same (floating-point) operation on massive amounts of data. To accommodate this way of processing, graphical processing units (GPUs) consist of a large number 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 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 computational demands similar to 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 speed-ups. Another advantage is that GPUs are relatively cheap because of the enormous numbers sold for graphical use in virtually every PC. One drawback is the 32-bit precision of the typical GPU and, in some cases more importantly, there is 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 satisfactory, however. GPU vendors have been quick to focus on the HPC community. They have tended to rename their graphics cards GPGPUs, *i.e.*, general-purpose GPUs, although the product is largely identical to the graphics cards sold in every shop. But there have also 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 usual for general-purpose CPUs, is becoming available (see p. 43). There are C-like languages and runtime environments available that make the life of a developer of GPUs much easier: for NVIDIA this is CUDA, which has become quite popular with users of these systems. AMD/ATI is concentrating on the newly defined standard OpenCL (see below), which 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 considerable effort in rewriting the code. This drawback is taken up by the GPU vendors (and not only them). Recently OpenCL has become available, which in principle is platform-independent, thus protecting the development effort put into the acceleration of a program. At present, Apple, ATI/AMD, Intel, NVIDIA, and PetaPath are members of the consortium that are willing to provide an OpenCL language interface. First experiences with OpenCL version 1.0, as provided by the Khronos Group, showed generally low performance, but with the new enhanced release of OpenCL 1.1 as of June 2010 this situation is improving.

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

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

ATI/AMD

In June 2010 the latest product from ATI (now wholly owned by AMD) was announced: the ATI FireStream 9370 card. The actual delivery was scheduled by AMD in the third quarter of 2010. Information about the

Table 10.1. Some specifications for the ATI/AMD FireStream 9370 GPU.

Number of processors	1600
Memory (GDDR5)	4 GB
Clock cycle	825 MHz
Internal memory bandwidth	$\leq$ 147.2 GB/s
Peak performance (32-bit)	2.64 Tflop/s
Peak performance (64-bit)	528 Gflop/s
Power requirement, typical	170 W
Power requirement, peak	225 W
Interconnect (PCIe Gen2)	$\times$16, 8 GB/s
ECC, error correction	No
Floating-point support	Partial (32/64-bit)

card, however, is still scant. There is not enough information available for a block diagram but we list in Table 10.1 some of the most important features of the processor. The specifications given indicate that, per core, two floating-point results per cycle can be generated, presumably the result of an add and a multiply operation. Whether these results can be produced independently or result from linked operations is not known, because of the lack of information. Unlike NVIDIA's Fermi card, discussed below, the FireStream 9370 does not support error correction yet. So, one has to be careful in assessing the outcomes of numerically unstable calculations.

Like its direct competitor, NVIDIA, ATI offers a free software development kit, SDK v.2.01, which supports OpenCL 1.1, Direct X11 and ComputeX. The earlier languages, such as BROOK+ and the very low-level Close-To-Metal software development vehicles, are no longer supported.

NVIDIA

NVIDIA is the other big player in the GPU field with regard to HPC. Its latest product is the Tesla C2050/C2070, also known as the 'Fermi' card. A simplified block diagram is shown in Figure 10.1. The GigaThread Engine is able to schedule different tasks in the streaming multiprocessors (SMs) in parallel. This greatly improves the occupation rate of the SMs and thus the throughput. As shown in Figure 10.1, three (or four) SMs per graphics processor cluster (GPC) are present. At the moment no more than a total of 14 SMs are available, although 16 were planned. When the 40 nm production process has sufficiently improved, the number of SMs may increase from 14 to the originally planned 16. A newly introduced feature is the L2 cache, shared by all SMs. Also, there is DMA support to get data from the host's memory without having to interfere with the host CPU. The

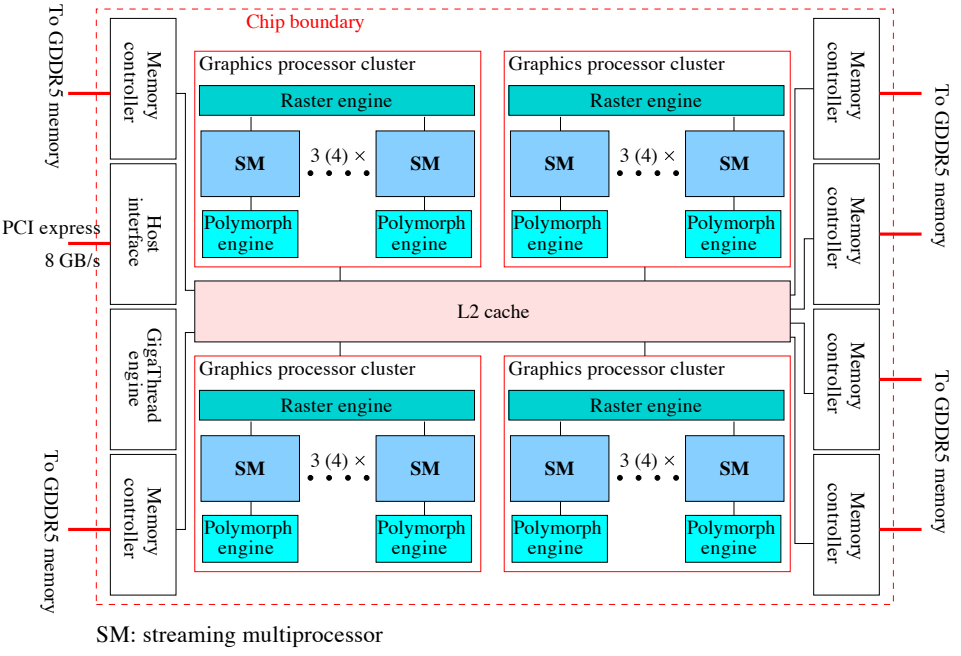


Figure 10.1. Simplified block diagram of the NVIDIA Tesla C2050/C2070 GPU.

GPU memory is GDDR5 and is connected to the card via six 64-bit wide memory interfaces for a bandwidth of about 150 GB/s.

Each SM in turn harbours 32 cores, which used to be named streaming processors (SPs) but now are called CUDA cores by NVIDIA. A diagram of an SM with some internals is given in Figure 10.2. Via the instruction cache's two warp schedulers (a warp is a bundle of 32 threads), the program threads are pushed onto the SPs. In addition each SM has four special function units, which take care of the evaluation of functions that are more complicated than can be profitably computed by the simple floating-point units in the SPs. Lastly, we list some properties of the Tesla C2050/70 in Table 10.2.

From these specifications, it can be inferred that two 32-bit floating-point results per core per cycle can be delivered. 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, provided by NVIDIA, or more specifically, its dense matrix–matrix multiplication. This operation occupies any computational core to the fullest and will therefore consume close to the maximum of the power. As can be seen from the table, the only difference between the C2050 and the C2070 is the amount

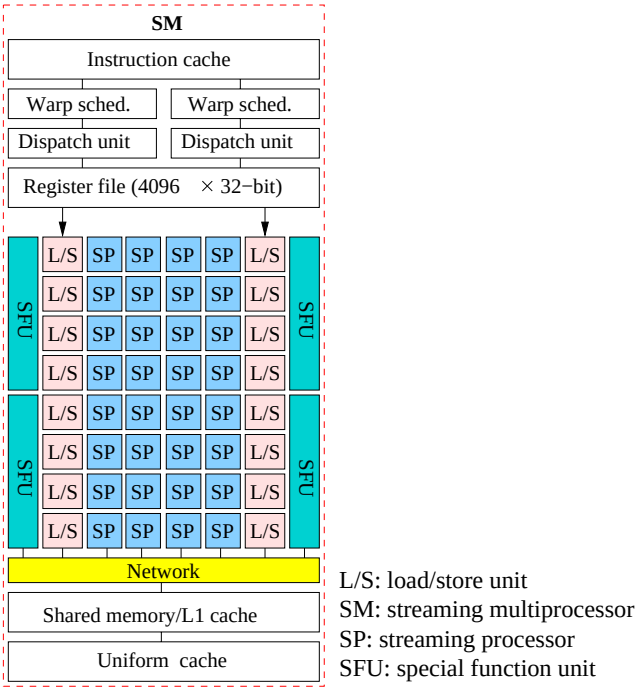


Figure 10.2. Diagram of a streaming processor of the NVIDIA Tesla C2050/C2070.

of memory: the C2050 features 3 GB of GDDR5 memory while the C2070 has double that amount.

Like ATI, NVIDIA provides an SDK comprising a compiler named CUDA, libraries that include BLAS and FFT routines, and a runtime system that accommodates both Linux (Red Hat 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 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 accordingly. This is not unique to the C2050 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. For Windows users the NVIDIA Parallel Nsight for Visual Studio is available, which should ease the optimization of the program parts run on the cards.

Table 10.2. Some specifications for the NVIDIA Tesla C2050/70 GPU.

Number of processors	448
Memory (GDDR5), C2050	3 GB
Memory (GDDR5), C2070	6 GB
Internal bandwidth	$\leq$ 153 GB/s
Clock cycle	1.15 GHz
Peak performance (32-bit)	1.03 Tflop/s
Peak performance (64-bit)	515 Gflop/s
Power requirement, peak	238 W
Interconnect (PCIe Gen2)	$\times$8, 4 GB/s; $\times$16, 8 GB/s
ECC, error correction	Yes
Floating-point support	Full (32/64-bit)

10.2. General computational accelerators

Although we have so far looked at the GPUs primarily from the perspective of computational accelerators, they are of course originally full-blown high-end graphical processors. Several vendors have developed accelerators that did not have graphical processing in mind as the foremost application to be served (although they might not be bad in this respect when compared to general CPUs). The future of general computational accelerators is problematic: in principle it is entirely possible to make accelerators that can compete with GPUs, or with the FPGA-based accelerators discussed in Section 10.3, but the volume will always be much lower than that of the other two accelerator variants, which is reflected in the production cost.

Below we discuss two of these general accelerators for the sake of completeness, but it is doubtful that they will survive as marketable products.

PetaPath

PetaPath is a spin-off of ClearSpeed, to position ClearSpeed products in the HPC market. ClearSpeed works in the embedded processor sector but a main product, namely the CSX700 processor, is well equipped for HPC work. We discuss this processor in some detail below.

The ClearSpeed products are in their third generation. Unlike GPUs, ClearSpeed processors were made to operate on 64-bit floating-point data from the start and have full error correction incorporated. The latest processor is the CSX700 chip, which is packaged in a number of products. The most common is the e710 card, which fits into a PCIe slot on any PC or server unit. A variant with a different form factor but with the same functionality is the e720 card, which can be put into blade servers. PetaPath also markets, apart from the separate cards, its Feynman e740 and e780

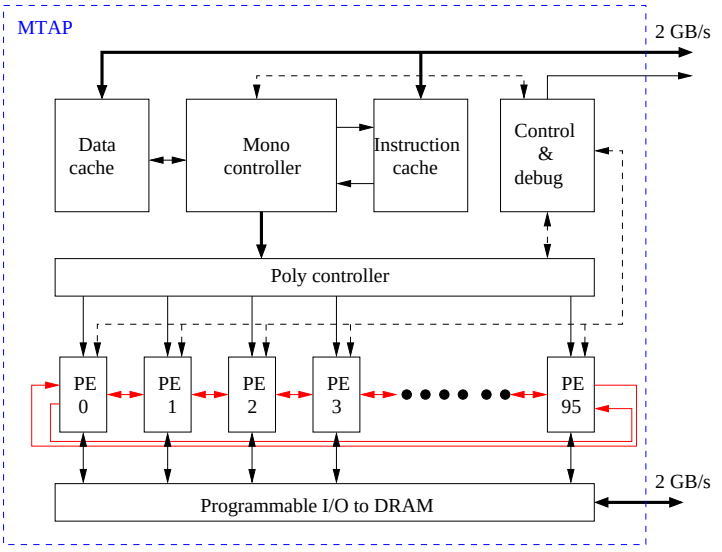


Figure 10.3. Block diagram of a ClearSpeed MTAP unit. Two of these units reside on a CSX700 chip.

units, which house 4 and 8 e720 cards and which connect to a host server by PCIe Gen. 2, 16×, *i.e.*, at 8 GB/s. The bandwidth for the individual cards is 2 GB/s, however. As the peak performance of a single e720 card is 96 Gflop/s, the peak performances of the Feynman e740 and e780 are 384 and 768 Gflop/s, respectively.

The power consumption of the e710/e720 card is extremely low: 25 W maximal, 15 W typical. This is partly due to the low clock frequency of 250 MHz. The e710 card contains, aside from the CSX700 processor, 2 GB DDR2 SDRAM, and an FPGA that manages the data traffic to and from the card. As stated earlier, the interconnect to the host system is compliant with PCIe 8×, amounting to a bandwidth of 2 GB/s. ClearSpeed is quite complete in giving technical details. So, we are able to show a block diagram of the CSX processor in Figure 10.3. Two so-called multi-threaded array processor (MTAP) units are located on one CSX700 chip. As can be seen, an MTAP contains 96 processors (with 4 redundant ones per MTAP). They are controlled via the poly controller, 'poly' being the indication for the data types that can be processed in parallel. The processing elements themselves are able to communicate fast between themselves via a dedicated ring network. Every cycle, a 64-bit data item can be shifted to the right or to the left through the ring. In Figure 10.4 we show the details of a processing element. A maximum of two 64-bit floating-point results can be generated per cycle. As one MTAP contains 96 PEs and there are 2 MTAPs

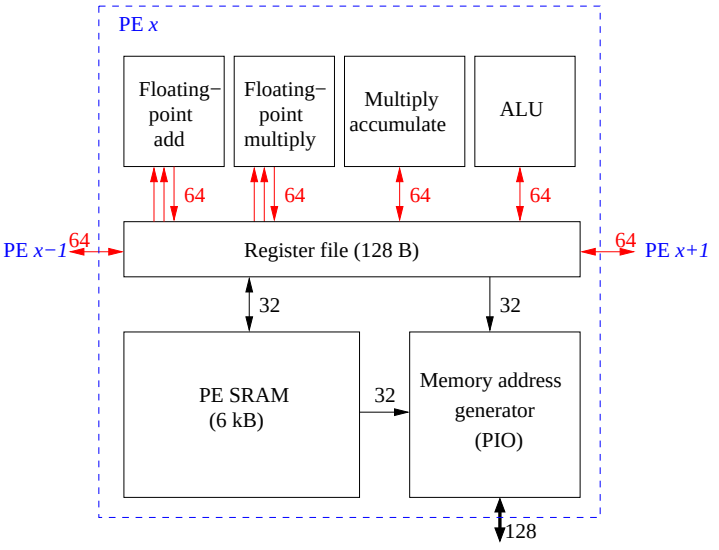


Figure 10.4. Block diagram of a PE in an MTAP of a CSX700 chip. The numbers near the arrows indicate the number of bits that can be transferred per cycle.

on a chip, the peak performance of a CSX700 chip is 96 Gflop/s at a clock frequency of 250 MHz.

Note the control and debug unit present in an MTAP. It enables debugging within the accelerator on the PE level. This is a facility that is missing in the GPUs and the FPGA accelerators we will discuss later.

Further, ClearSpeed employs an extended form of C, called C^n, for program development on the card. The extension is very slight, however. The keywords mono and poly are added to indicate data that should be processed serially or in parallel, respectively. Because ClearSpeed has been in the accelerator trade for quite some time, the SDK is very mature. Apart from the C^n compiler already mentioned, it contains a library with a large set of BLAS/LAPACK routines, FFTs, and random number generators. For dense linear algebra there is an interface that enables calling the routines from a host program in Fortran. Furthermore, a graphical debugging and optimization tool is present that may or may not be embedded in IBM's Eclipse integrated development environment (IDE) as a plug-in.

The IBM/Sony/Toshiba Cell processor

The Cell processor, officially called the Cell Broadband Engine (Cell BE), was designed at least partly with the gaming industry in mind. Sony uses it for its PS3 gaming platform, and to be successful it has to deliver high performance for the graphical part, as well as do a large amount of floating-point computation to sustain the rapidly changing scenes that occur during

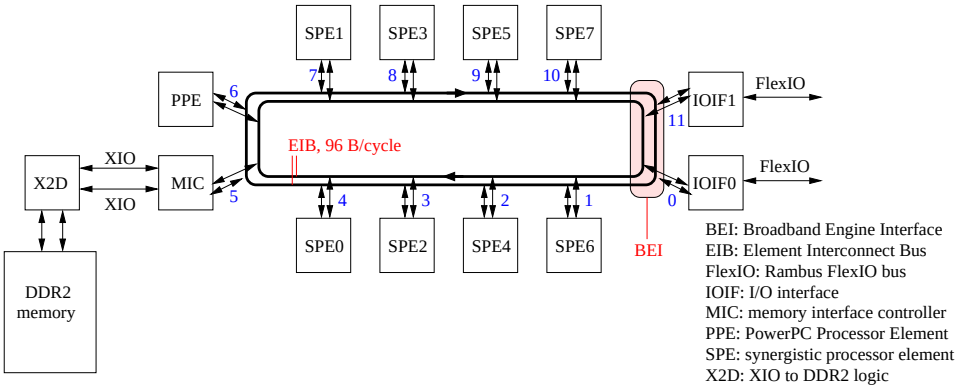


Figure 10.5. Block diagram of an IBM PowerXCell processor. The numbers near the arrows indicate the device numbering used for delivering data via the Element Interconnect Bus.

a game. The Cell processor is therefore not a pure graphics processor but considerably more versatile than a GPU. The testament to this is that Mercury computers, specializing in systems for radar detection, *etc.*, markets a product with two Cell processors, instead of dedicated DSPs (*i.e.*, digital signal processors), while Toshiba has incorporated the Cell in HDTV sets and is considering bringing out notebooks with a Cell processor. The Cell processor is able to operate in 32-bit as well as in 64-bit floating-point mode, though there is a large performance difference: in single precision the peak speed is 204.8 Gflop/s while in double precision it is about 14 Gflop/s. From the start there was keen interest in the HPC community. It also restarted discussion of whether it is necessary to use 64-bit precision calculation all the way through an application or, by reformulating some key algorithms, whether it would be possible to get results with acceptable accuracy when parts are carried out in single precision (Langou *et al.* 2006). At least for the Cell processor this discussion has become of less importance, as at present the variant is available under the name of PowerXCell 8i, which is developed by IBM, probably expressly targeted at the HPC area. In the PowerXCell the speed for 64-bit precision has increased considerably to 102.4 Gflop/s, half the speed of the single precision computations. Also, it is produced in 65 nm instead of 90 nm technology, and it employs DDR2 memory instead of the Rambus memory used in the original Cell processor. Figure 10.5 shows a diagram of this rather complicated processor. As can be seen, the processor is hybrid in the sense that it contains two different kinds of processors: the PPE, which is essentially a PowerPC core as discussed in Section 9.4, and eight synergistic processor elements (SPEs), all running at a clock frequency of 3.2 GHz. The SPEs are meant to do the bulk of the computation, while

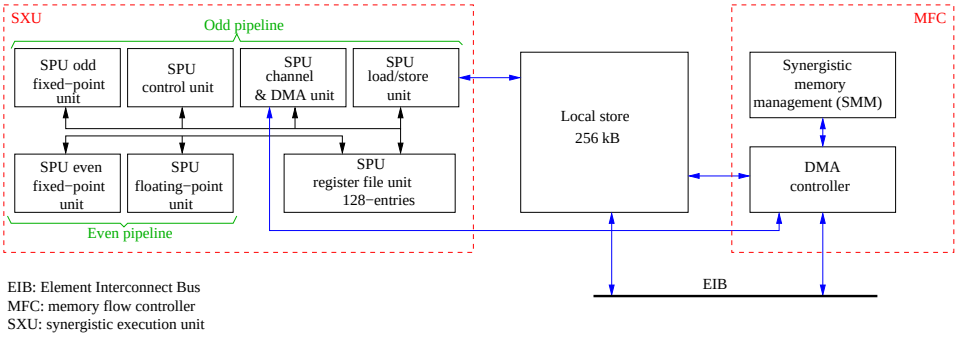


Figure 10.6. Block diagram of an IBM PowerXCell synergistic processing element (SPE).

the PPE takes care of operating system tasks and coordinates the work to be done by the SPEs. All devices in the processor are connected by the Element Interconnect Bus (EIB). The EIB in fact consists of four 16-B-wide rings that transport data in opposite directions to minimize the distance between the devices in the processor. The devices connected to the EIB are numbered, to allow data to be transferred from one device to another. Up to 96 B/cycle can be transferred, amounting to 307.2 GB/s. Although the PowerXCell uses DDR2 memory, the processor proper is designed for use with Rambus memory. This has been taken care of by including the X2D device, which translates the DDR memory requests into Rambus requests and *vice versa*. The two I/O interfaces are controlled through the Broadband Engine Interface (BEI). They have different functions: IOIF1 takes care of the usual external I/O devices via the IOIF protocol, while IOIF0 is able to use the internal I/O protocol, BIF, which is also used on the EIB rings. In this way it is possible to connect to other Cell processors.

The SPEs are the computational workhorses of the Cell processor. We show the internals of an SPE in Figure 10.6. Roughly, there are three important parts in an SPE: the synergistic execution unit (SXU), which contains the functional units for computation, load/store, and DMA control; the local store, which contains the local data to be operated on; and the memory flow controller (MFC), which in turn contains the DMA controller and the memory management unit. As shown in Figure 10.6, in the SXU, the functional units are organized into an odd and an even pipeline. Two instructions can be issued every cycle, one for each of these pipelines. This also implies that one floating-point instruction can be issued per cycle. Depending on the type of operands, this can yield four 32-bit results or two 64-bit results per cycle (in the PowerXCell, in the original Cell processor a 64-bit result can be delivered every 13 cycles, hence the much lower double-precision performance). Note that an SPE does not have any form of cache.

Rather, data are brought in from external memory by DMA instructions via the EIB. This leads to much lower memory latency when a data item is not in the local store. Up to 16 DMA requests can be outstanding for any of the SPEs. As all SPEs are independent, up to 128 DMA requests can be in flight. Of course, this explicit memory management does not make for easy programming. So, one must be careful in managing the data to get (close to) optimal performance.

IBM has put much effort into a software development kit for the Cell processor. It is freely available and, apart from the necessary compilers, there is an extensive library for managing the data transport both from the PPE to the SPEs, between SPEs, initiating the processes on the SPEs, retrieving the results, and managing program overlays. As the local stores in the SPEs are small, the old concept of overlays has been revived. The program is divided into units that depend on each other but do not constitute the whole program. By loading and unloading the units in the correct sequence one can still execute the total program. In addition, there are debugging and performance analysis tools. The total program development can be done using IBM's IDE, Eclipse.

The PowerXCell 8i won its share of fame for its use in the Roadrunner system at Los Alamos National Laboratory. In this system 3240 so-called triblades are connected by InfiniBand. A triblade consists of two QS22 blades, each containing two PowerXCell processors, and an LS21 blade with two Opteron processors. This configuration was the first to break the LINPACK petaflop barrier. This fact certainly helped to increase interest in the Cell processor as an accelerator platform. At present, there are many research projects under way to assess the applicability of Cell BE accelerators and to make their learning curve less steep.

10.3. FPGA-based accelerators

An FPGA (field programmable gate array) is an array of logic gates that can be hardware-programmed to fulfil 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 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 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 means of a hardware description language (HDL), such as VHDL or Verilog. This is very cumbersome for the average programmer as one has to explicitly define not only details such as the placement of the configured devices but also the width of the operands to be be operated on, *etc.* This problem has been recognized 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. Further, for FPGA-based accelerators, as for GPUs, there is an initiative to develop a unified API that will ensure compatibility between platforms. The non-profit OpenFPGA consortium is heading this effort. Various working groups are concentrating on, for instance, a core library, an application library, and an API definition. There is no unified way to program FPGAs platform independently, however, and it may take a long time to get there.

The two big players on the FPGA market are Altera and Xilinx. However, in the accelerator business one will seldom find these names mentioned, because the FPGAs they produce are packaged in a form that makes them unusable 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 from 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 FPGAs that have, for instance, a PowerPC405 embedded. Therefore we present FPGA accelerators only in the most global way, and our treatment is necessarily incomplete.

In the following we will discuss products of vendors that have gone to great lengths to not expose their users to the use of HDLs, although for the highest benefits this cannot always be avoided. Necessarily, our choice of topics is again somewhat arbitrary, because this area is changing extremely rapidly.

Convey

The Convey HC-1 was announced in November 2008. It is an example of the hybrid solutions that have arisen to avoid the unwieldy HDL programming of FPGAs, while still benefiting from their potential acceleration capabilities. The HC-1 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 four Xilinx V5 FPGAs that can be configured into a variety of 'personalities' that would accommodate users from different application areas. The personalities offered include the oil and gas industry, the financial analytics market, and the life sciences.

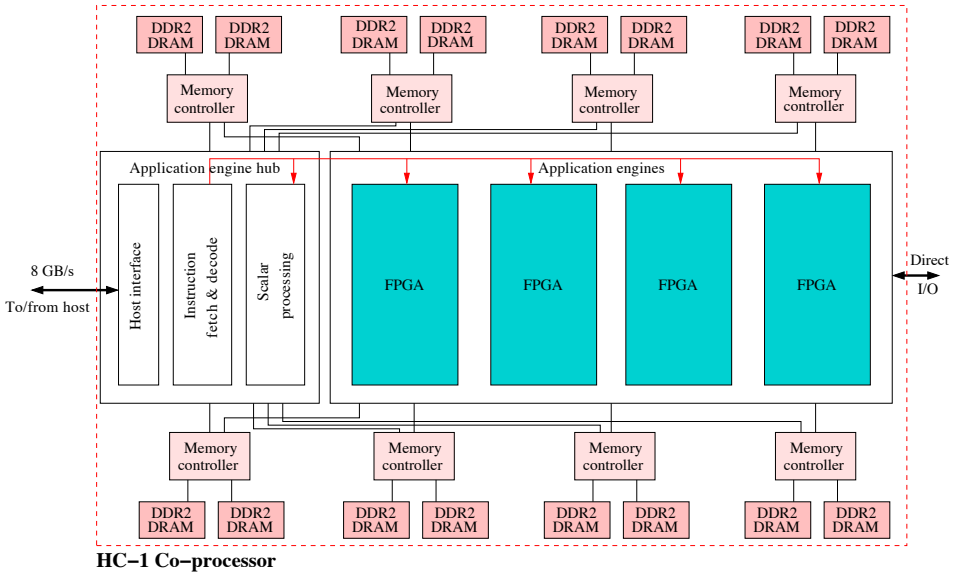


Figure 10.7. Block diagram of the Convey HC-1.

In Figure 10.7 we give a diagram of the HC-1 co-processor's structure. A personality that will be used often for scientific and technical work is the vector personality. Thanks to the compilers provided by Convey standard, code in Fortran and C/C++ can be automatically vectorized and execute the vector units configured in the four 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, a pipe 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 10.7 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. As yet there is too little experience with the HC-1 to compare it one-to-one with other systems in terms of performance. However, it is clear that the potential speed-up for many applications can be large.

Kuberre

Since May 2009 Kuberre has marketed 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. Although Kuberre is naturally oriented to the financial analytics market, the little material that is accessible shows that libraries such as ScaLAPACK, Monte Carlo algorithms, FFTs and wavelet transforms are available. For the life sciences, standard applications such as BLAST and Smith–Waterman are present. The standard GNU C libraries can also be linked seamlessly.

The processors are organized in a grid fashion and use a 256 GB distributed shared cache to combat data access latency. The system comes configured with 768 RISC CPUs for what are called 'generic C/C++ programs', or as 1536 double-precision cores for heavy numerical work. It is possible to split the system to run up to 16 different 'contexts' (reminiscent of Convey's personalities: see p. 52). Part of the machine may be dedicated to a life sciences application, while other parts work on encryption and numerical applications.

As for the Convey HC-1, 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.

The material publicly available does not allow us to show a reliable block diagram, but this may come about later when the system is installed at sites that want to evaluate it.

SRC

Until two 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. In addition the so-called SRC-7 MAP station is now marketed, MAP being the processing unit that contains 2 Altera Stratix II FPGAs. Furthermore, SRC has the IMAP card as a product that can be plugged into a PCIe slot on any PC.

SRC has gone to great lengths to ban the term 'FPGA' from its documentation. Instead, it talks about implicit versus 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 as well, because shared extended memory can be put into the system that is equally accessible by

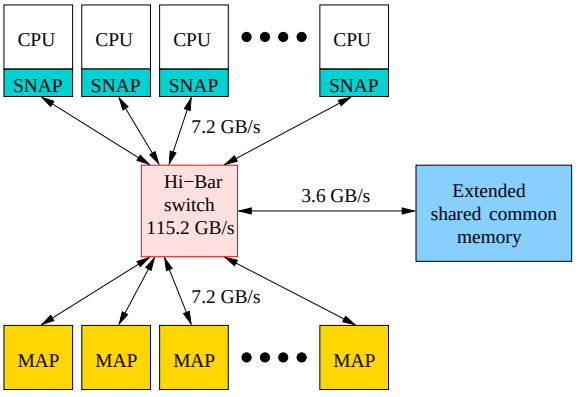


Figure 10.8. Approximate machine structure of the SRC-7.

both the CPUs and the MAP processors. We show a sketch of the machine structure in Figure 10.8. 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 is 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 SRC's 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, common extended memory can also be configured. This allows for shared-memory parallelism in the system across CPUs and MAP processors.

The MAP station is a shrunken version of the SRC-7: it contains an 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. As with Convey and Kuberre, C/C++ is also available. The parallelization and acceleration are largely done by putting comment directives in Fortran code and 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 configure them and execute the target code. Furthermore, there is an extensive library of functions, a debugger and a performance analyser. When one wants to employ specific non-standard functionality, *e.g.*, computing with arithmetic of non-standard length, one can create a so-called 'application-specific functional unit'. In fact, one then configures one or more of the FPGAs directly and one has to fall back on VHDL or Verilog for this configuration.

11. Networks

Fast interprocessor networks are, together with fast processors, decisive factors for 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 greatly, and to give a balanced view of the possibilities opened by the improved networks, a discussion of some of these networks is in order. Networks have been employed as an important component of 'integrated' parallel systems.

Of course, Gigabit Ethernet (GbE) is now widely available, and with a maximum theoretical bandwidth of 125 MB/s would be able to fulfil a useful role for some applications that are not latency-bound. Furthermore, 10 Gigabit Ethernet (10 GbE) is increasingly available. 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 10 GbE. 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 10 GbE. As such it is not quite on a par with the ubiquitous InfiniBand interconnects with regard to latency and bandwidth. However, the costs are lower and may compensate for a somewhat lower performance in many cases. Various vendors, such as Myrinet and SCS, have circumvented the problem with TCP/IP by implementing their own protocol, thus using standard 10 GbE equipment but with their own network interface cards (NICs) to handle the proprietary protocol. In this way

Table 11.1. Some bandwidths and latencies for various networks as measured with an MPI ping-pong test.

Network	Bandwidth GB/s	Latency μs
Arista 10 GbE (stated)	1.2	4.0
BLADE 10 GbE (measured)	1.0	4.0
Cray SeaStar2+ (measured)	6.0	4.5
Cray Gemini (measured)	6.1	1.0
IBM (InfiniBand) (measured)	1.2	4.5
SGI NumaLink 5 (measured)	5.9	0.4
InfiniBand (measured)	1.3	4.0
InfiniPath (measured)	0.9	1.5
Myrinet 10-G (measured)	1.2	2.1

latencies of 2–4 μs can be achieved: well within the range of other network solutions. Very recently Mellanox came out with 40 GbE on an InfiniBand fabric. It is too early, however, to give characteristics of this new medium.

We restrict ourselves here to networks that are independently marketed as the proprietary networks for systems such as those of Cray and SGI, and 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 the main developments which one is likely to meet when one scans the high-performance computing arena. Unfortunately, the spectrum of network types is narrowed by the demise of Quadrics. Quadrics' QsNet$^{\text{II}}$ was rather expensive but it had excellent characteristics. The next generation, QsNet$^{\text{III}}$, was on the brink of deployment when the Italian mother company Alinea terminated Quadrics – much to the regret of HPC users and vendors.

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 customized 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 of the node, and therefore the maximum bandwidth that can be attained in internode communication. In practice, the available bandwidths are in the range 110–480 MB/s. Since 1999 PCI-X has been available, initially at 1 GB/s, in PCI-X 2.0 also at 2 and 4 GB/s. Coupling with PCI-X is at present mostly superseded by its successor PCI-Express 1.1 (PCIe). This provides a 200 MB/s bandwidth per data lane where 1×, 2×, 4×, 8×, 12×, 16×, and 32× multiple data lanes are supported: this makes it fast enough for the host bus adapters of any communication network vendor so far. So, for the networks discussed below, often different bandwidths are quoted, depending on the PCI bus type and the supporting chipset. Therefore, when speeds are quoted, it is always with the proviso that the PCI bus of the host node is sufficiently wide/fast.

Lately, PCIe 2, commonly known as PCIe Gen2, has emerged, with twice the bandwidth. Currently PCIe Gen2 is mostly used within servers to connect to high-end graphics cards (including GPUs used as computational accelerators) at speeds of 4–8 GB/s, but evidently it could also be used to connect to either other computational accelerators or network interface cards that are designed to work at these speeds.

An idea of network bandwidths and latencies for some networks, both proprietary and vendor-independent, is given in Table 11.1. 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 which these values were obtained was very diverse. So, there is no guarantee that these are the optimum attainable results.

11.1. InfiniBand

InfiniBand has rapidly become a widely accepted medium for internode networks. The specification was finished in June 2001. Since 2002, a number of vendors have started to offer their products based on the InfiniBand standard. A very complete description (1200 pages) can be found in Shanley (2002). 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 Gigabit 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 due to the high volume of sales at present. The characteristics of InfiniBand are rather nice. There are product definitions both for copper and glass fibre connections, switch and router properties are defined, and multiple connections can be employed for high bandwidth. Also, the way messages are broken up into packets and reassembled, as well as routing, prioritizing, and error handling, are all described in the standard. This makes InfiniBand independent of one particular technology and, because of its completeness, it is a good basis upon which to implement a communication library (such as MPI).

Conceptually, InfiniBand knows of two types of connectors to the system components: host channel adapters (HCAs), already mentioned, and target channel adapters (TCAs). The latter are typically used to connect to I/O subsystems, while HCAs concern us more as they are the connectors used in interprocessor communication. InfiniBand defines a basic link speed of 2.5 Gb/s (312.5 MB/s) but also a 4× and 12× speed of 1.25 GB/s and 3.75 GB/s, respectively. Moreover, HCAs and TCAs can have multiple ports that are independent and allow for higher reliability and speed.

Messages can be sent on the basis of remote direct memory access (RDMA) from one HCA/TCA to another: an 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, provides a message with a 128-bit header which is IPv6 compliant and which 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 sent on. Short messages of 32 B can be embedded in control messages, which cuts down on the negotiation time for control messages.

InfiniBand switches for HPC are normally offered with 8–864 ports and now mostly at a speed of 1.25 GB/s. However, Sun is now providing a

3456-port switch for its Constellation cluster systems. Switches and HCAs accommodating twice this speed (double data rate, DDR) are now common, but are being replaced more and more by quad data rate (QDR), which became available in late 2008. Obviously, to take advantage of this speed at least PCI Express must be present at the nodes to which the HCAs are connected. The switches can be configured in any desired topology, but in practice a fat tree topology is almost always preferred (see Figure 6.2(b)). How much of the raw speed can be realized depends, of course, on the quality of the MPI implementation imposed on the InfiniBand specifications. A ping-pong experiment on InfiniBand-based clusters with different MPI implementations has shown bandwidths of 1.3 GB/s, and an MPI latency of 4 μs for small messages is quoted by Mellanox, one of the large InfiniBand vendors. The in-switch latency is typically about 200 ns. For the QDR 2.5 GB/s products, the MPI bandwidth indeed nearly doubles while the latency stays approximately the same. At the time of writing, QDR InfiniBand products are available from Mellanox and QLogic. A nice feature of QDR InfiniBand is that it provides dynamic routing, which is not possible with earlier generations. In complicated communication schemes this feature should alleviate contention on some data paths by letting the message take an alternative route.

Because of the recent profusion of InfiniBand vendors, the price is now on a par with, or lower than, those of other fast network vendors such as Myrinet (Section 11.3) and 10 GbE.

11.2. InfiniPath

InfiniPath only provides HCAs with a 4-wide (1.25 GB/s) InfiniBand link on the network side and connecting to a HyperTransport bus or PCI-Express on the computer side. For systems with AMD processors on board, the HyperTransport option is particularly attractive because of the direct connection to the host's processors. This results in very low latencies for small messages. PathScale, the vendor of the InfiniPath HCAs, quotes latencies as low as 1.29 μs. Obviously, this type of HCA cannot be used with systems based on non-AMD processors. For these systems the HCAs with PCI-Express can be used. They have a slightly higher, but still low, latency of 1.6 μs. The effective bandwidth is also high: a uni-directional bandwidth of $\approx$ 950 MB/s can be obtained using MPI for both types of HCA.

The InfiniPath HBAs do not contain processing power themselves. Any processing associated with the communication is done by the host processor. According to PathScale this is an advantage because the host processor is usually much faster than the processors employed in switches. An evaluation report from Sandia National Lab (Doerfler 2005) seems to corroborate this assertion.

PathScale only offers HCAs (and the software stack coming with it) and these can be used by any InfiniBand switch vendor that adheres to the OpenIB protocol standard, which includes pretty much all of them.

11.3. Myrinet

Until recently Myrinet was the market leader in fast cluster networks and it is still one of the largest. The Myricom company which sells Myrinet started in 1994 with its first Myrinet implementation (Boden *et al.* 1995), as an alternative to Ethernet for connecting the nodes in a cluster. Apart from the higher bandwidth, around 100 MB/s at that time, the main advantage was that it entirely operated in user space, thus avoiding operating system interference and the delays that come with it. This meant that the latency for small messages was around 10–15 μs. Latency and bandwidth compared nicely with the proprietary networks of integrated parallel systems of Convex, IBM, and SGI at the time. Although such a network came at a non-negligible cost, in many cases it proved a valuable alternative to either an Ethernet-connected system or an even costlier integrated parallel system.

Since then hardware upgrades and software improvements have made Myrinet the network of choice for many cluster builders, and until a few years ago there was hardly an alternative when a fast, low-latency network was required.

Like InfiniBand, Myrinet uses cut-through routing for an efficient utilization of the network. Also, RDMA is used to write to/read from the remote memory of other host adapter cards, called Lanai cards. These cards interface with the PCI-X of PCI Express bus of the host they are attached to. Myrinet allows copper cables or fibres as signal carriers. The latter form gives a high flexibility in the connection and much headroom in the speed of signals, but the fibre cables and connectors are rather delicate, which can lead to damage when cluster nodes have to be serviced.

Myrinet offers ready-made 8–256 port switches (8–128 for its newest product: see below). The 8 and 16 port switches are full crossbars. In principle all larger networks are built from these using a Clos network topology. An example for a 64-port systems is shown in Figure 11.1. A Clos network is another example of a logarithmic network with the maximum bi-sectional bandwidth of the endpoints. Note that 4 ports of the 16×16 crossbar switches are unused, but other configurations need either more switches or connections or both.

Since the start of 2006 Myricom, like many InfiniBand switch vendors, has provided a multi-protocol switch (and adapters), the Myri-10G. Apart from Myricom's own MX protocol it also supports 10 Gigabit Ethernet, which makes it easy to connect to external nodes/clusters – an ideal start-

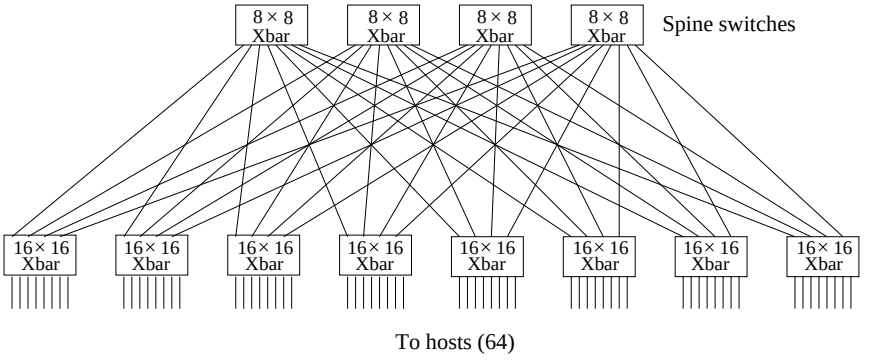


Figure 11.1. An 8×16 Clos network using 8 and 16 port crossbar switches to connect 64 processors.

ing point for building grids from a variety of systems. The specifications as given by Myricom are quite good: ≈ 1.2 GB/s for the uni-directional theoretical bandwidth for both its MX protocol and about the same for the MX emulation of TCP/IP on Gigabit Ethernet. According to Myricom, there is no difference in bandwidth between MX and MPI, and the latencies are claimed to be the same: just over 2 μs.

12. Recent trends in high-performance computing

In this section we analyse major recent trends and changes in high-performance computing. Massively parallel processors (MPPs) became successful in the early 1990s due to their better price/performance ratios, which was enabled by advances in microprocessors. The success of microprocessor-based symmetric multiprocessor (SMP) concepts, even for very high-end systems, was the basis for the emergence of cluster concepts in the early 2000s. During the first half of the decade clusters of PCs and workstations became the prevalent architecture for many HPC application areas on all ranges of performance. However, the Japanese Earth Simulator vector system demonstrated that many scientific applications could benefit greatly from other computer architectures. At the same time there has been renewed broad interest within the scientific HPC community concerning new hardware architectures and new programming paradigms. The IBM BlueGene system is one early example of a shifting design focus for large-scale systems.

12.1. Introduction

Looking back on the last four decades, the HPC market has always been characterized by rapid change in vendors, architectures, technologies and system usage. Despite all these changes, the evolution of performance on a

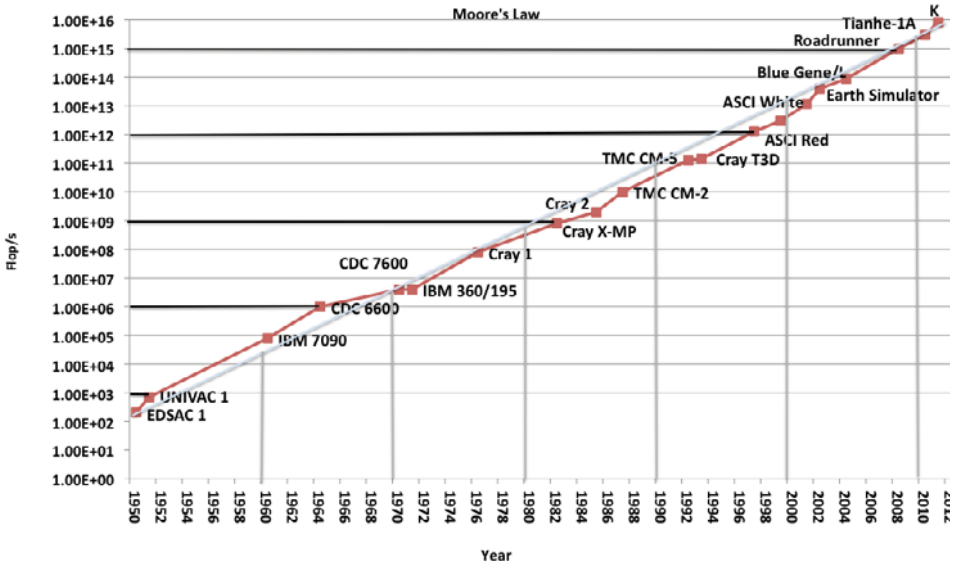


Figure 12.1. Performance of the fastest computer systems for the last six decades.

large scale seems to be a very steady and continuous process. Moore's Law is often cited in this context. Figure 12.1 plots the peak performance of various computers of the last six decades, all 'supercomputers' of their time (Hockney and Jesshope 1988, Meuer 1994), and demonstrates how well this law holds for nearly the entire lifespan of modern computing. On average we see an increase in performance of two orders of magnitude every decade. In this section we analyse recent major trends and changes in the HPC market. For this, we focus on systems which had at least some commercial relevance. This paper extends a previous analysis of the HPC market in Meuer, Strohmaier, Dongarra and Simon (2011). Historical overviews with different focuses can be found in Wilson (1994) and Woodward (1996). Section 12.3 analyses the trend in the first half of the 2000s, and Section 12.4 looks to the future.

The initial success of vector computers in the 1970s was driven by raw performance. The introduction of this type of computer system started the era of 'supercomputing'. In the 1980s the availability of standard development environments and application software packages became more important. Next to performance, these criteria determined the success of MP vector systems, especially with industrial customers. MPPs became successful in the early 1990s due to their better price/performance ratios, enabled by the attack of the 'killer micros'. In the lower and medium market segments, the MPPs were replaced by microprocessor-based SMP systems in the middle of the 1990s. Towards the end of the 1990s, only the companies which

had entered the emerging markets for massively parallel database servers and financial applications attracted enough business volume to be able to support the hardware development for the numerical high-end computing market as well. Success in the traditional floating-point intensive engineering applications was no longer sufficient for survival in the market. The success of microprocessor-based SMP concepts, even for the very high-end systems, was the basis for the emergence of cluster concepts in the early 2000s. During the first half of the decade clusters of PCs and workstations became the prevalent architecture for many application areas in the Top500 on all ranges of performance. However, the Earth Simulator vector system demonstrated that many scientific applications can benefit greatly from other computer architectures. At the same time there has been renewed broad interest within the scientific HPC community concerning new hardware architectures and new programming paradigms. The IBM BlueGene/L system is one early example of a shifting design focus for large-scale systems. The IBM Roadrunner system at Los Alamos National Laboratory broke the petaflops threshold in June 2008. And in November 2011 the Japanese K computer reached the 10 petaflop mark using over a half a million cores of conventional design.

please check

12.2. A short history of supercomputers

In the second half of the 1970s the introduction of vector computer systems marked the beginning of modern supercomputing. These systems offered a performance advantage of at least one order of magnitude over conventional systems of that time. Raw performance was the main if not the only selling argument. In the first half of the 1980s the integration of vector systems in conventional computing environments became more important. Only those manufacturers which provided standard programming environments, operating systems and key applications were successful in getting industrial customers, and survived. Performance was mainly increased by improved chip technologies and by producing shared-memory multiprocessor systems.

Fostered by several US government programmes, massively parallel computing with scalable systems using distributed memory became the centre of interest at the end of the 1980s. The main goal for their development was overcoming the hardware scalability limitations of shared-memory systems. The increase in performance of standard microprocessors after the RISC revolution, together with the cost advantage of large-scale productions, formed the basis for the 'attack of the killer micros'. The consequence was a transition from ECL to CMOS chip technology and the use of 'off-the-shelf' microprocessors instead of custom-designed processors for MPPs.

The traditional design focus for MPP systems was the very high end of performance. In the early 1990s the SMP systems of various workstation

manufacturers, as well as the IBM SP series, which targeted the lower and medium market segments, gained great popularity. Their price/performance ratios were better due to not including support for very large configurations and due to economies of scale. Due to the vertical integration of production, it was no longer economically feasible to produce and focus on the highest end of computing power alone. The design focus for new systems shifted to the market of medium-performance systems.

please check

The acceptance of MPP systems not only for engineering applications but also for new commercial applications, especially for database applications, emphasized different criteria for market success, such as stability of systems, continuity of manufacturer and price/performance. Success in commercial environments became a new important requirement for a successful supercomputer business towards the end of the 1990s. Due to these factors and the consolidation in the number of vendors in the market, hierarchical systems built with components designed for the broader commercial market replaced homogeneous systems at the very high end of scientific computing. The marketplace readily adopted clusters of SMPs, while academic research focused on clusters of workstations and PCs.

12.3. 2000–2005: clusters, Intel processors, and the Earth Simulator

In the early 2000s, clusters built with off-the-shelf components gained more and more attention, not only as academic research objects, but also as computing platforms with end-users of HPC systems. By 2004, this group of clusters represented the majority of new systems on the TOP500 in a broad range of application areas. One major consequence of this trend was the rapid rise in the use of Intel processors in HPC systems. Though virtually absent at the high end at the beginning of the decade, Intel processors are now used in the majority of HPC systems. Clusters in the 1990s were mostly self-made systems designed and built by small groups of dedicated scientist or application experts. This changed rapidly as soon as the market for clusters based on PC technology matured. Now the large majority of TOP500-class clusters are manufactured and integrated by either a few traditional large HPC manufacturers such as IBM or Hewlett-Packard, or numerous small, specialized integrators of such systems.

In 2002 a system with a different architecture, the Earth Simulator, entered the spotlight as the new number one system on the TOP500, and it managed to take the US HPC community by surprise, even though it had been announced four years earlier. The Earth Simulator, built by NEC, is based on NEC vector technology and showed unusually high efficiency on many scientific applications. This fact invigorated discussions about future architectures for high-end scientific computing systems. The first system built with a different design focus, but still with mostly conventional off-

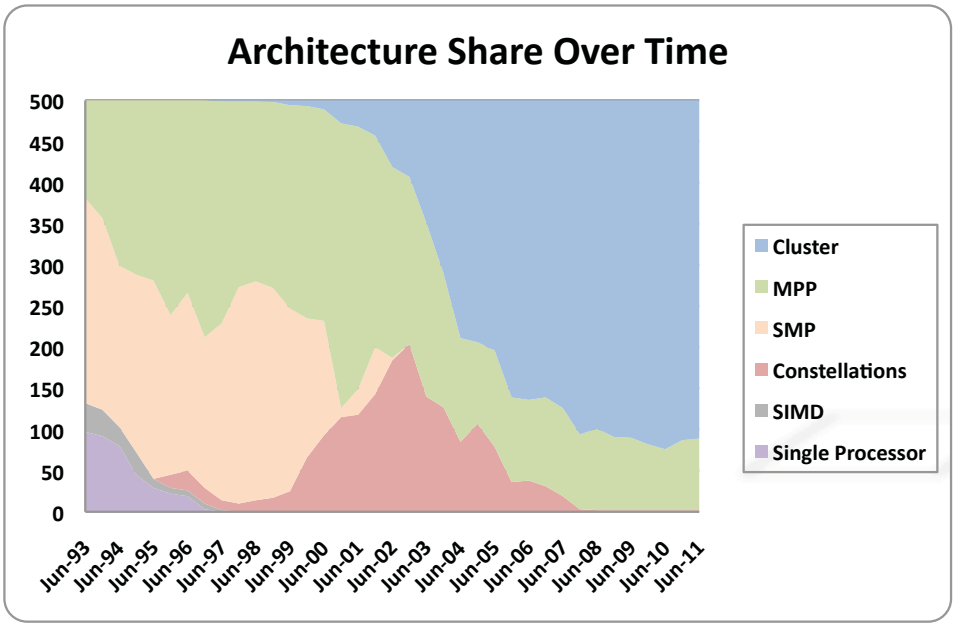


Figure 12.2. Main architectural categories seen in the Top500. (The term 'constellations' refers to clusters of SMPs.)

the-shelf components, is the IBM BlueGene/L system. Its design focuses on a system with an unprecedented number of processors using a power-efficient design while sacrificing main memory size.

Explosion of cluster-based systems

By the end of the 1990s clusters were common in academia, but mostly as research objects, and not so much as computing platforms for applications. Most of these clusters were of comparable small scale and as a result the November 1999 edition of the Top500 listed only seven cluster systems. This changed dramatically, as industrial and commercial customers started to deploy clusters as soon as their applications permitted them to take advantage of the better price/performance ratio of commodity-based clusters. At the same time, all major vendors in the HPC market started selling this type of cluster, fully integrated into their customer base. In November 2004 clusters became the dominant architecture in the Top500, with 294 systems at all levels of performance (see Figure 12.2). Companies such as IBM and Hewlett-Packard sold the majority of these clusters, and a large number of them were installed at commercial and industrial sites. To some extent, the reasons for the dominance of commodity-processor systems are economic. Contemporary distributed-memory supercomputer systems based on commodity processors (such as Linux clusters) appear to be

substantially more cost-effective – by roughly an order of magnitude – in delivering computing power to applications that do not have stringent communication requirements. On the other hand, there has been little progress, and perhaps regress, in making scalable systems easy to program. Software directions that were started in the early 1980s (such as CM-Fortran and High-Performance Fortran) were largely abandoned. The pay-off to finding better ways to program such systems, and thus expand the domains in which these systems can be applied, would appear to be large.

The move to distributed memory has forced changes in the programming paradigm of supercomputing. The high cost of processor-to-processor synchronization and communication requires new algorithms that minimize those operations. The structuring of an application for vectorization is seldom the best structure for parallelization on these systems. Moreover, despite some research successes in this area, without some guidance from the programmer, compilers are generally able neither to detect enough of the necessary parallelism, nor to reduce sufficiently the inter-processor overheads. The use of distributed-memory systems has led to the introduction of new programming models, particularly the message-passing paradigm, as realized in MPI, and the use of parallel loops in shared-memory subsystems, as supported by OpenMP. It has also forced significant reprogramming of libraries and applications to port onto the new architectures. Debuggers and performance tools for scalable systems have developed slowly, however, and even today most users consider the programming tools on parallel supercomputers to be inadequate.

Fortunately, there are a number of choices of communication networks available. There is generally great variation in the use of clusters and their more integrated counterparts: clusters are mostly used for capacity computing while the integrated machines are primarily used for capability computing. Traditionally, vendors of large supercomputer systems have learned to provide for capacity computing, as the precious resources of their systems were required to be used as effectively as possible. In contrast, Beowulf clusters are mostly operated through the Linux operating system, although a small minority use Microsoft Windows. These operating systems either lack the tools to make use of clusters for capacity computing, or the tools are immature. However, as clusters become on average both larger and more stable, they are being used as computational capacity servers.

Intel-ization of the processor landscape

The HPC community had already started to use commodity parts in large numbers in the 1990s. MPPs and constellations (the term 'constellations' refers to a cluster of SMPs), typically using standard workstation microprocessors, still might use custom interconnect systems. There was, however, one big exception: virtually nobody used Intel microprocessors. Lack of

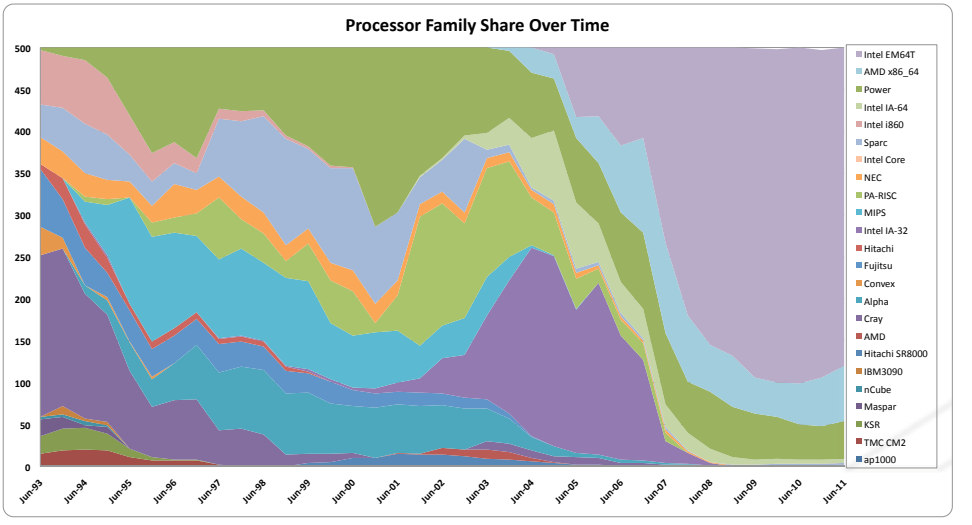


Figure 12.3. Main processor families seen in the TOP500.

performance and the limitations of a 32-bit processor design were the main reasons for this. This changed with the introduction of the Pentium 3 and especially in 2001 with the Pentium 4, which featured greatly improved memory performance due to its front-side bus and full 64-bit floating-point support. The number of systems in the TOP500 with Intel processors exploded from only six in November 2000 to 375 in June 2008 (Figure 12.3)

The Earth Simulator shock

The Earth Simulator (ES) was conceived, developed, and implemented by Dr Hajime Miyoshi, who is regarded as the Seymour Cray of Japan. Unlike his peers, he seldom attended conferences or gave public speeches. However, he was well known within the HPC community in Japan for his involvement in the development of the first Fujitsu supercomputer, and later on the Numerical Wind Tunnel (NWT) at the National Aerospace Laboratory of Japan. In 1997 he took up his post as director of the Earth Simulator Research and Development Center and led the development of the 40 Tflop/s Earth Simulator, which would serve as a powerful computational engine for global environmental simulation.

Prior to the ES, global circulation simulations were made using a 100 km grid width, although ocean-atmospheric interactive analyses were not performed. Obtaining quantitatively good predictions for the evaluation of environmental effects may require a grid width of at most 10 km or 10 times finer meshes in the x, y and z directions, and interactive simulations. Thus a supercomputer 1000 times faster and larger than a 1995 conventional

supercomputer might be required. Miyoshi investigated whether such a machine could be built in the early 2000s. His conclusion was that it could be realized if several thousand of the most advanced vector supercomputers of approximately 10 Gflop/s speed were clustered using a very high-speed network. He forecasted that extremely high-density LSI integration technology, high-speed memory, all packaged into small-size, high-speed network (crossbar) technology, as well as an efficient operating system and Fortran compiler, could all be developed within the next few years. He thought that only a strong initiative project with government financial support could realize this kind of machine.

The machine was completed in February 2002, and the entire system is still being used as an end-user service. Miyoshi, as leader of the NWT project, supervised the development of NWT Fortran and organized the HPF (High Performance Fortran) Japan Extension Forum, which is used on the ES. He knew that a high-level vector/parallel language is critical for such a supercomputer.

The launch of the Earth Simulator created a substantial amount of concern in the USA that it had lost the leadership in high-performance computing. While there was certainly a loss of national pride for the USA in not being first on a list of the world's fastest supercomputers, it is important to understand the set of issues surrounding that loss of leadership. The development of the ES represents a large investment (approximately $500m, including a special facility to house the system) and a large commitment over a long period of time. The USA has made an even larger investment in HPC in the Department of Energy Advanced Strategic Computing (ASC) programme, but the funding has not been spent on a single platform. Other important differences are as follows.

- The ES was developed for basic research and is shared internationally, whereas the ASC programme is driven by national defence and the systems have restricted domestic use.
- A large part of the ES investment supported NEC's development of their SX-6 technology. The ASC program has made only modest investments in industrial R&D.
- The ES uses custom vector processors. The ASC systems use commodity processors.
- The ES software technology largely originates from abroad, although it is often modified and enhanced in Japan. For example, significant ES codes were developed using a Japanese enhanced version of HPF. Virtually all software used in the ASC program has been developed by the USA.

Surprisingly, the Earth Simulator's number one ranking on the TOP500 list was not a matter of national pride in Japan. In fact, there is considerable

resentment of the Earth Simulator in some sectors of the research community in Japan. Some Japanese researchers feel that the ES is too expensive and drains critical resources from other science and technology projects. Due to the continued economic crisis in Japan and large budget deficits, it is getting more difficult to justify government projects of this kind.

New architectures on the horizon

Interest in novel computer architectures has always been large in the HPC community, which comes as no surprise, as this field was engendered by and continues to thrive on technological innovations. Some of the concerns of recent years have been the ever-increasing space and power requirements of modern commodity-based supercomputers. In the BlueGene/L development, IBM addressed these issues by designing a very power- and space-efficient system. BlueGene/L does not use the latest commodity processors available but computationally less powerful and much more power-efficient processor versions developed, not for the PC and workstation market, but mainly for embedded applications. Together with a drastic reduction of available main memory, this provided a very dense system. To achieve the targeted extreme performance level, an unprecedented number of these processors (up to 212 992) are combined using several specialized interconnects.

There was and is considerable doubt whether such a system would be able to deliver the promised performance and would be usable as a general-purpose system. The first results of the beta-System were very encouraging, and the one-quarter size beta-System commissioned by Lawrence Livermore National Laboratory was able to claim the number one spot on the November 2004 TOP500 list.

12.4. 2005 and beyond

Three decades after the introduction of the Cray 1, the HPC market had changed considerably. It used to be a market for systems clearly different from any other computer systems. Today the HPC market is no longer an isolated niche market for specialized systems. Vertically integrated companies produced systems of any size. Components used for these systems are the same as those from an individual desktop PC up to the most powerful supercomputers. Similar software environments are available on all of these systems. This was the basis of a broad acceptance by industrial and commercial customers.

The increasing market share of industrial and commercial installations had several very critical implications for the HPC market. The manufacturers of supercomputers for numerical applications, in the market for small- to medium-size HPC systems, face strong competition from manufacturers selling their systems in the very lucrative commercial market. These systems

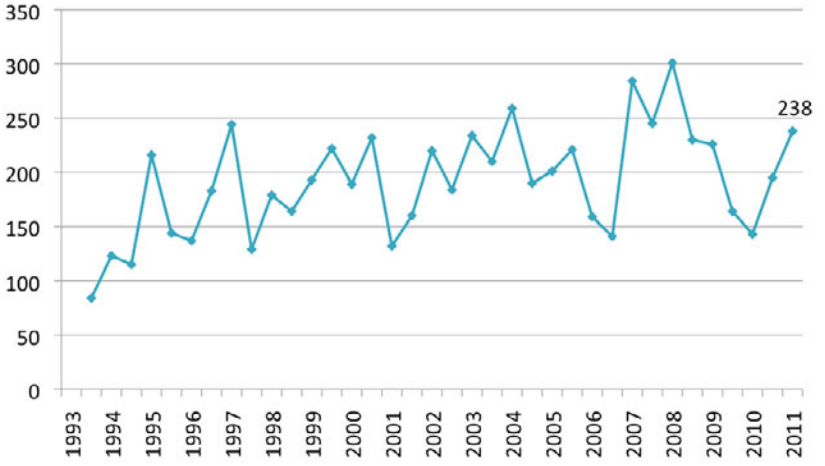


Figure 12.4. The replacement rate in the Top500 defined as the number of systems omitted because of their performance being too small.

tend to have better price/performance ratios due to the larger production numbers of systems sold to commercial customers and the reduced design costs of medium-size systems. The market for very high-end systems itself is relatively small, and hardly grows, if at all. It cannot easily support specialized niche market manufacturers. This forces the remaining manufacturers to change the design for the very high end away from homogeneous large-scale systems towards cluster concepts based on 'off-the-shelf' components.

'Clusters' are the dominating architecture in the Top500. In November 1999 we had only seven clusters in the Top500, whereas in June 2011 the list included 411 cluster systems. At the same time the debate as to whether we need new architectures for very high-end supercomputers has once again increased in intensity.

Novel hybrid architectures have appeared in the Top500 list. The number one machine in June 2008, the IBM Roadrunner, was just such a system. The Roadrunner is a hybrid design built from commodity parts. The system is composed of two processor chip architectures, the IBM PowerXCell and the AMD Opteron, which use InfiniBand interconnect. The system can be characterized as an Opteron-based cluster with Cell accelerators. Each Opteron core has a Cell chip (composed of nine cores). The Cell chip has eight vector cores and a conventional PowerPC core. The vector cores provide the bulk of the computational performance. The other hybrid design that has found some favour is one based on a linking between a commodity CPU and a graphical processing unit (GPU) accelerator. The model for GPU computing is to use a CPU and GPU together in a heterogeneous co-processing computing model. The sequential part of the application runs on the CPU and the computationally intensive part is accelerated by the GPU.

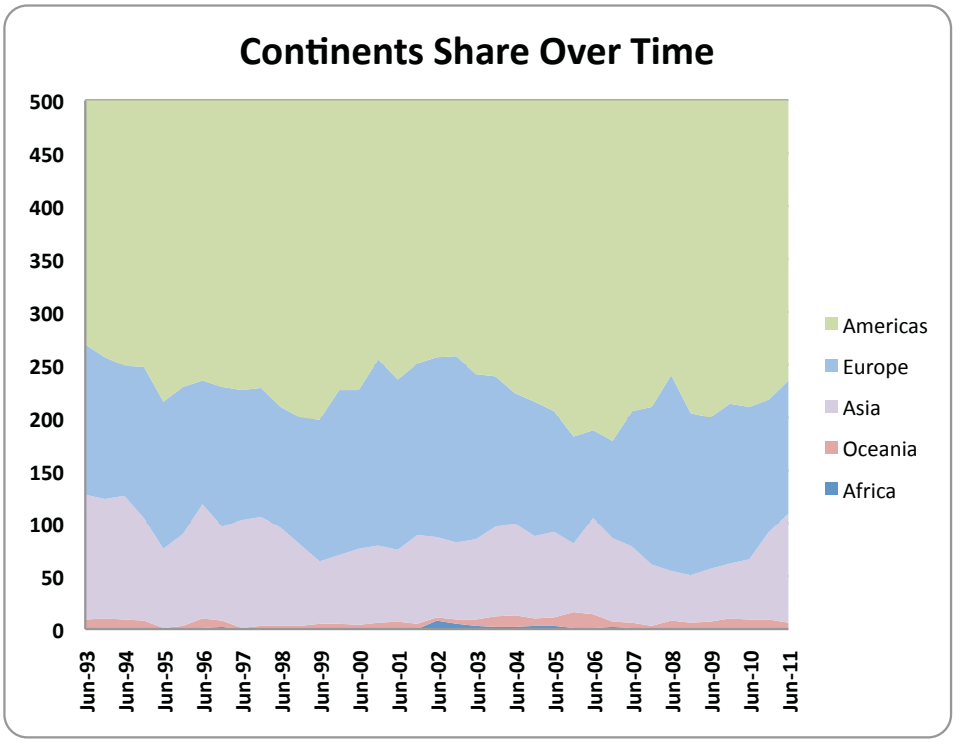


Figure 12.5. The consumers of HPC systems in different geographical regions, as seen in the Top500.

Dynamic of the market

The HPC market is by its very nature dynamic. This is not only reflected by the coming and going of new manufacturers but especially by the need to update and replace systems quite often to keep pace with the general performance increase. This general dynamic of the HPC market is well reflected in the Top500. In Figure 12.4 we show the number of systems that fall off the end of the list within six months due to the increase in entry-level performance. We see an average replacement rate of about 180 systems every six months, or more than half the list every year. This means that a system which is at position 100 at a given time will fall off the Top500 within two to three years. The June 2011 list shows almost one-half replacement, with 238 systems being displaced from the previous list.

Consumer and producer

The dynamic of the HPC market is well reflected in the rapidly changing market shares of the chip or system technologies, of manufacturers, customer types or application areas. However, if we are interested in where these HPC systems are installed or produced, we see a different picture.

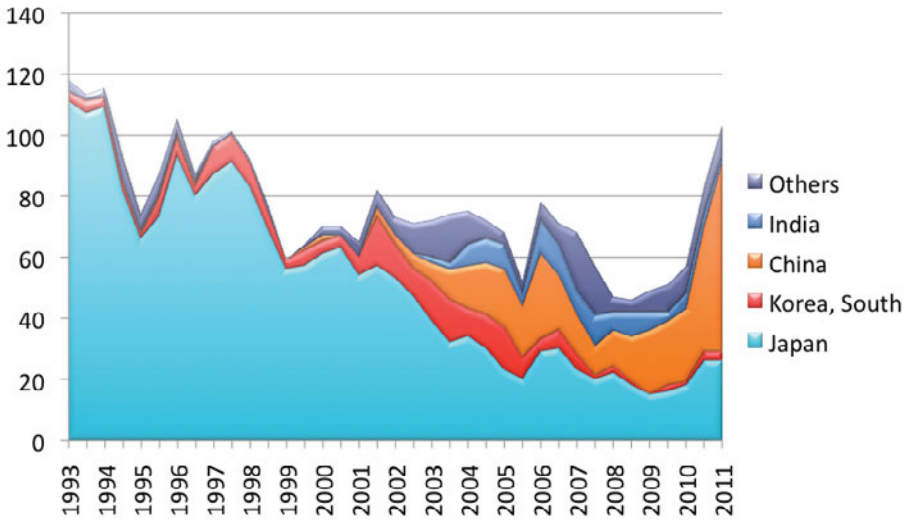


Figure 12.6. The consumers of HPC systems in Asia, as seen in the TOP500.

Plotting the number of systems installed in different geographical areas in Figure 12.5, we see a more-or-less steady distribution. The number of systems installed in the USA is about half of the list, while the number of systems in Asia is slowly increasing. Europe has steadily acquired HPC systems, as shown in Figure 12.5. While this can be interpreted as a reflection of increasing economical strength in these countries, it also highlights the fact that it is becoming easier for such countries to buy or even build cluster-based systems themselves. Figure 12.6 shows the number of HPC systems in Japan, the initial use of such systems in India, and the rapid growth of systems in China.

Performance growth

While many aspects of the HPC market change quite dynamically over time, the evolution of performance seems to follow Moore's Law quite well, as mentioned earlier. The TOP500 provides ideal data to verify an observation like this. Looking at the computing power of the individual machines presented in the TOP500 and the evolution of the total installed performance, we plot the performance of the systems at positions 1 and 500 in the list as well as the total accumulated performance. In Figure 12.7 the curve of position 500 shows on average an increase of a factor of 1.9 within one year. All other curves show a growth rate of 1.8 ± 0.05 per year.

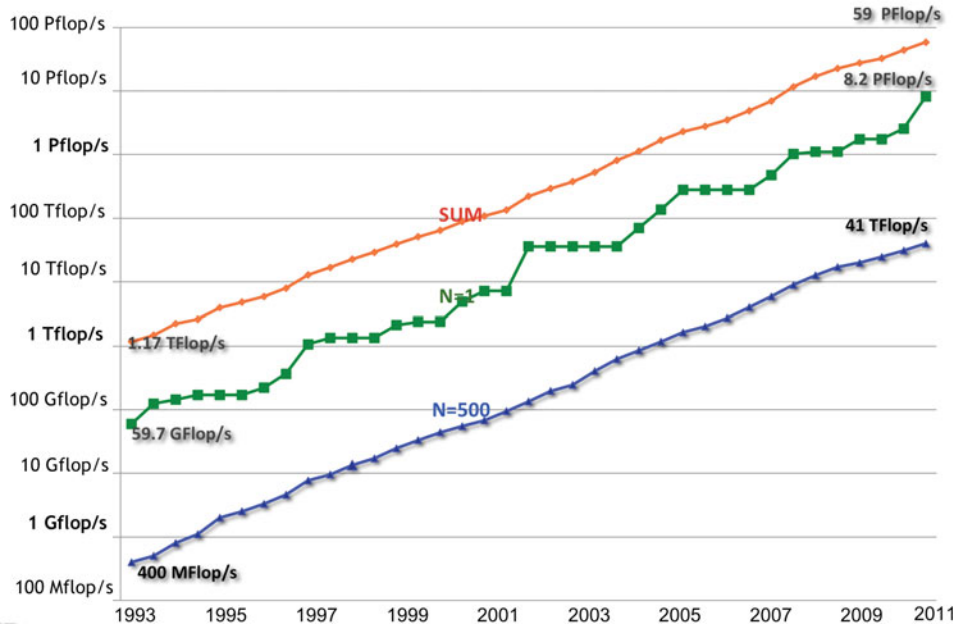


Figure 12.7. Overall growth of accumulated and individual performance, as seen in the TOP500.

Projections

Based on the current TOP500 data, which cover the last fourteen years, and the assumption that the current performance development will continue for some time to come, we can now extrapolate the observed performance and compare these values with the goals of the mentioned government programs. In Figure 12.8, we extrapolate the observed performance values using linear regression on the logarithmic scale. This means that we fit exponential growth to all levels of performance in the TOP500. This simple fitting of the data shows surprisingly consistent results. In 1999, based on a similar extrapolation (Meuer *et al.* 2011), we expected to have the first 100 TFlop/s system by 2005. We also predicted that by 2005 no system smaller than 1 TFlop/s should still be able to make the TOP500. Both of these predictions are basically certain to be fulfilled next year. Looking forward another five years to 2010, we expected to see the first petaflop system at about 2009 (Meuer *et al.* 2011). We hit the petaflop/s mark in 2008 and 10 petaflop/s in 2011. please check

Looking even further into the future, we could speculate that, based on the current doubling of performance every year, the first system exceeding 100 petaflop/s should be available around 2015, and we should expect an exaflop/s system in 2019, as can be seen in Figure 12.8. Indeed we see an eleven-year cycle of achieving a three-orders-of-magnitude increase in perfor-

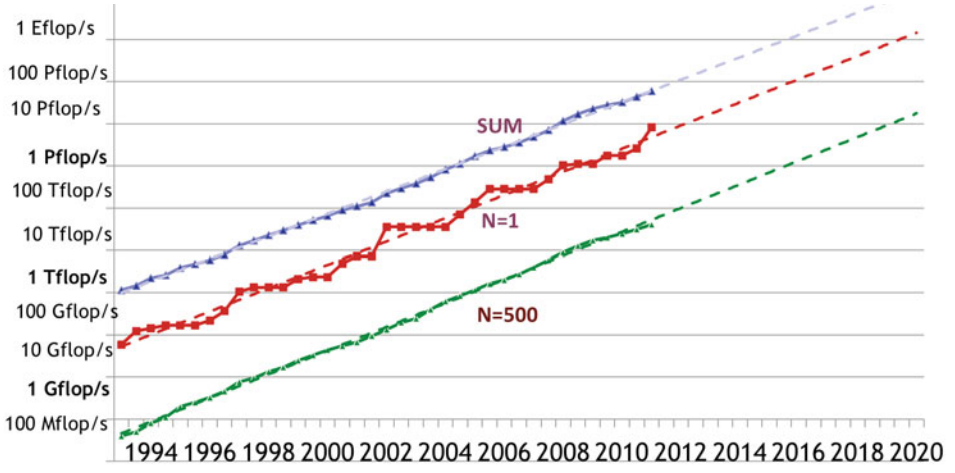


Figure 12.8. Extrapolation of recent growth rates of performance seen in the Top500.

mance. This has been true since 1986 with the first gigaflop system, in 1997 with the first teraflop system, and in 2008 with the first petaflop system. Due to the rapid changes in the technologies used in HPC systems, there is, however, again no reasonable projection possible for the architecture of such a system in ten years. Even though the HPC market has changed quite substantially since the introduction of the Cray 1 four decades ago, there is no end in sight for these rapid cycles of re-definition: the only constant is change.

13. HPC challenges

Supercomputing capability benefits a broad range of industries, including energy, pharmaceuticals, aircraft, automobiles and entertainment. More powerful computing capability will allow these diverse industries to more quickly engineer superior new products that could improve a nation's competitiveness. In addition, there are considerable flow-down benefits that will result from meeting both the hardware and software high-performance computing challenges. These would include enhancements to smaller computer systems and many types of consumer electronics, from smartphones to cameras.

With respect to software, it seems clear that the scope of the effort to develop software for exascale must be truly international. In terms of its rationale, scientists in nearly every field now depend upon the software infrastructure of high-end computing to open up new areas of enquiry (*e.g.*, the very small, very large, very hazardous, very complex), to dramatically increase their research productivity, and to amplify the social and economic

impact of their work. It serves global scientific communities who need to work together on problems of global significance and leverage distributed resources in transnational configurations. In terms of feasibility, the dimensions of the task – totally redesigning and recreating, in the period of just a few years, the massive software foundation of computational science in order to meet the new realities of extreme-scale computing – are simply too large for any one country, or small consortium of countries, to undertake all on its own.

Standardization is also a minimum requirement for broad international collaboration on development of software components. In addition, the international nature of the science will demand further development of global data management tools and standards for shared data.

The development of an exascale computing capability, with machines capable of executing $O(10^{18})$ operations per second in the 2018 time frame, will be characterized by significant and dramatic changes in computing hardware architecture from current (2011) petascale high-performance computers. From the perspective of computational science, this will be at least as disruptive as the transition from vector supercomputing to parallel supercomputing that occurred in the 1990s. Similar to that transition, the achievement of scientific application performance commensurate with the expected improvement in computing capability will require identifying and/or developing mathematical models and numerical algorithms that map efficiently onto exascale architectures, significant re-engineering of scientific application codes supported by the corresponding development of new programming models and system software appropriate for these new architectures. Achieving these increases in capability by 2018 will require a significant acceleration in the development of both hardware and software. This could be accomplished through an intensive 'co-design' effort, where system architects, application software designers, applied mathematicians, and computer scientists work interactively to characterize and produce an environment for computational science discovery that fully leverages these significant advances in computational capability.

The algorithmic challenges

Advancing science in key areas requires development of next-generation physical models to satisfy the accuracy and fidelity needs for targeted simulations. The impact of these simulation fidelity needs on requirements for computational science is twofold. First, more complex physical models must be developed to account for more aspects of the physical phenomena being modelled. Second, for the physical models being used, increases in resolution for key system variables, such as numbers of spatial zones, time steps or chemical species, are needed to improve simulation accuracy, which in turn places higher demands on computational hardware and software.

Application models represent the functional requirements that drive the need for certain numerical algorithms and software implementations. The choice of model is in part motivated by the science objectives, but it is also constrained by the computer hardware characteristics attainable in the relevant time frame. The choice and specification of system attributes (*e.g.*, peak speed or node memory capacity) tend to constrain the functional attributes able to be employed in a given physical model on that system.

Science priorities lead to science models, and models are implemented in the form of algorithms. Algorithm selection is based on various criteria, such as appropriateness, accuracy, verification, convergence, performance, parallelism and scalability.

Models and associated algorithms are not selected in isolation but must be evaluated in the context of the existing computer hardware environment. Algorithms that perform well on one type of computer hardware may become obsolete on newer hardware, so selections must be made carefully and may change over time.

Moving forward to exascale will put heavier demands on algorithms in at least two areas: the need for increasing amounts of data locality in order to perform computations efficiently, and the need to obtain much higher factors of fine-grained parallelism as high-end systems support increasing numbers of compute threads. As a consequence, parallel algorithms must adapt to this environment, and new algorithms and implementations must be developed to extract the computational capabilities of the new hardware.

As with science models, the performance of algorithms can change in two ways as application codes undergo development and new computer hardware is used. First, algorithms themselves can change, motivated by new models or performance optimizations. Second, algorithms can be executed under different specifications, *e.g.*, larger problem sizes or changing accuracy criteria. Both of these factors must be taken into account.

Significant new model development, algorithm redesign, and science application code reimplementation, supported by (an) exascale-appropriate programming model(s), will be required to effectively support the power of exascale architectures. The transition from current sub-petascale and petascale computing to exascale computing will be at least as disruptive as the transition from vector to parallel computing in the 1990s.

Uncertainty quantification will permeate the exascale science workload. The demand for predictive science results will drive the development of improved approaches for establishing levels of confidence in computational predictions. Both statistical techniques involving large ensemble calculations and other statistical analysis tools will have significantly different dynamic resource allocation requirements than in the past, and the significant code redesign required for the exascale will present an opportunity to embed uncertainty quantification techniques in exascale science applications.

New multicore-friendly and multicore-aware algorithms
Scalable multicore systems bring new computation/communication ratios. Within a node, data transfers between cores are relatively inexpensive, but temporal affinity is still important for effective cache use. Across nodes, the relative cost of data transfer is growing very large. The development of new algorithms that take these issues into account can often perform very well, as do communication-avoiding algorithms that increase the computation/communication ratio or algorithms that support simultaneous computation/communication, or algorithms that vectorize well and have a large volume of functional parallelism.

Adaptive response to load imbalance
Adaptive multiscale algorithms are an important part of the US Department of Energy portfolio since they apply computational power precisely where it is needed. However, they introduce challenging computational requirements because they introduce dynamically changing computation that results in load imbalances from a static distribution of tasks. As we move towards systems with billions of processors, even naturally load-balanced algorithms on homogeneous hardware will present many of the same daunting problems with adaptive load balancing that are observed in today's adaptive codes. For example, software-based recovery mechanisms for fault-tolerance or energy-management features will create substantial load imbalances as tasks are delayed, by rollback, to a previous state or correction of detected errors. Scheduling based on a directed acyclic graph also requires new approaches to optimizing for resource utilization without compromising spatial locality. These challenges require development and deployment of sophisticated software approaches to rebalance computation dynamically in response to changing workloads and conditions of the operating environment.

Multiple precision algorithms/software
Algorithms and applications are becoming increasingly adaptive and we have seen that various adaptivity requirements have become an essential, key component of their roadmap to exascale computing. Another aspect of this quest for adaptivity is related to the development of libraries that recognize and exploit the presence of mixed-precision mathematics. A motivation comes from the fact that, on modern architectures, the performance of 32-bit operations is often at least twice as fast as the performance of 64-bit operations. Moreover, by using a combination of 32-bit and 64-bit floating-point arithmetic, the performance of many linear algebra algorithms can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. This can be applied not only to conventional processors but also to other technologies such as GPUs, and thus can spur the creation

of mixed-precision algorithms that more effectively utilize heterogeneous hardware.

Mixed-precision algorithms can easily provide substantial speed-up for very little code effort by mainly taking into account existing hardware properties. Earlier work has shown how to derive mixed-precision versions for various architectures and for a variety of algorithms for solving general sparse or dense linear systems of equations. Typically, a direct method is first applied in single precision in order to achieve a significant speed-up compared to double precision. Then an iterative refinement procedure aims at retrieving the lost digits. Iterative refinement can also be applied for eigenvalue and singular-value computations.

Current topics of interest include the extension and incorporation of this approach in applications that do not necessarily originate from linear algebra, and studying the robustness of mixed-precision algorithms on large-scale platforms. Indeed, the convergence of the mixed-precision iterative refinement solvers strongly depends on the condition number of the matrix at hand. The conditioning can be determined at runtime and proper precision can be selected. Ideally, the user could specify the required precision for the result and the algorithm would choose the best combination of precision on the local hardware in order to achieve it. The actual mechanics would be hidden from the user.

Fast implicit solvers

Carefully analysing complex problems, and adapting preconditioners to the underlying problem physics, is how most of the progress in this area is being made. However, it is typically the case that advanced preconditioners are composed of standard algebraic components such as advanced multigrid/multilevel methods, incomplete factorizations and basic smoothers. Furthermore, we need to renew our focus on basic iterative methods in an attempt to address bottlenecks due to collective operations (*e.g.*, dot-products) and poor kernel performance. Emphasis on block methods, recycling methods, s-step-like methods and mixed-precision formulations will be necessary to address the next generation of problems.

Communication avoiding and asynchronous algorithms

Algorithmic complexity is usually expressed in terms of the number of operations performed rather than the quantity of data movement to memory. This is antithetical to the true costs of computation, where memory movement is very expensive and operations are nearly free. To address the critical issue of communication costs, there is a need to investigate algorithms that reduce communication to a minimum. One needs to derive bandwidth and latency lower bounds for various dense and sparse linear algebra algorithms on parallel and sequential machines, *e.g.*, by extending the well-known lower

bounds for the usual $O(n^3)$ matrix multiplication algorithm. Then one needs to discover new algorithms that attain these lower bounds in many cases. Second, for Krylov subspace methods such as GMRES, CG and Lanczos, one should focus on taking k steps of these methods for the same communication costs as a single step.

In a seminal paper, Chazan and Miranker (1969) studied chaotic relaxation, now usually called asynchronous relaxation, for the solution of linear systems. In chaotic relaxation, the order in which components of the solution are updated is arbitrary and the past values of components that are used in the updates are also selected arbitrarily. This is a model for parallel computation in which different processors work independently and have access to data values in local memory.

When this and subsequent research was undertaken in the late 1960s and 1970s, it was largely theoretical: the existing computers did not have the capability for massively parallel processing. Today we are at the extreme, with the next generation of machines having $O(10^9)$ program threads. We are being challenged to devise algorithms and software that can effectively exploit the parallel hardware systems that are being developed. When solving very large problems on parallel architectures the most significant concern becomes the cost per iteration of the method – typically on account of communication and synchronization overheads. This is especially the case for Krylov methods, which are the most popular class of iterative methods for large sparse systems. This means that, for the first time, totally asynchronous iterative algorithms will become competitive for a wide range of application problems. Coping with fault tolerance, load balancing, and communication overheads in a heterogeneous computation environment is a challenging undertaking for software development. In traditional synchronous algorithms each iteration can only be performed as quickly as the slowest processor permits. If a processor fails, or is less capable, or has an unduly heavy load, then this markedly impacts iteration times. The use of asynchronous methods allows one to overcome many of the communication, load-balancing and fault tolerance issues we now face and which limit our ability to scale to the extreme.

Auto-tuning

Libraries need to have the ability to adapt to the possibly heterogeneous environment in which they have to operate. The adaptation has to deal with the complexity of discovering and implementing the best algorithm for diverse and rapidly evolving architectures. This calls out for automating the process, both for the sake of productivity and for correctness. Here, productivity refers both to the development time and the user's time to solution. The objective is to provide a consistent library interface that remains the same for users independent of scale and processor heterogeneity, but which

achieves good performance and efficiency by binding to different underlying code, depending on the configuration. The diversity and rapid evolution of today's platforms means that auto-tuning of libraries such as BLAS will be indispensable to achieving good performance, energy efficiency, load balancing, *etc.*, across this range of systems. In addition, the auto-tuning has to be extended to frameworks that go beyond library limitations, and are able to optimize data layout (such as blocking strategies for sparse matrix/SpMV kernels), stencil auto-tuners (since stencil kernels are diverse and not amenable to library calls), and even tuning of optimization strategy for multigrid solvers (optimizing the transition between the multigrid coarsening cycle and bottom-solver to minimize runtime). Adding heuristic search techniques and combining them with traditional compiler techniques will enhance the ability to address generic problems extending beyond linear algebra.

Scheduling and memory management for heterogeneity and scale
Extracting the desired performance from environments that offer massive parallelism, especially where additional constraints (*e.g.*, limits on memory bandwidth and energy) are in play, requires more sophisticated scheduling and memory management techniques than have heretofore been applied to linear algebra libraries. Another form of heterogeneity comes from confronting the limits of domain-decomposition in the face of massive explicit parallelism. Feed-forward pipeline parallelism can be used to extract additional parallelism without forcing additional domain-decomposition, but exposes the user to dataflow hazards. Ideas relating to a data flow-like model, expressing parallelism explicitly in directed acyclic graphs, so that scheduling tasks dynamically, support massive parallelism, and apply common optimization techniques to increase throughput. Approaches to isolating side-effects include explicit approaches that annotate the input arguments to explicitly identify their scope of reference, or implicit methods such as using language semantics or strongly typed elements to render code easier to analyse for side-effects by compiler technology. New primitives for memory management techniques are needed that enable diverse memory management systems to be managed efficiently and in coordination with the execution schedule.

Fault tolerance and robustness for large-scale systems
Modern PCs may run for weeks without rebooting and most data servers are expected to run for years. However, because of their scale and complexity, today's supercomputers run for only a few days before rebooting. Exascale systems will be even more complex and have millions of processors in them. The major challenge in fault tolerance is that faults in extreme-scale

systems will be continuous rather than an exceptional event. This requires a major shift from today's software infrastructure. Every part of the exascale software ecosystem has to be able to cope with frequent faults without rebooting; otherwise applications will not be able to run to completion. The system software must be designed to detect and adapt to frequent failure of hardware and software components. On today's supercomputers every failure kills the application running on the affected resources. These applications have to be restarted from the beginning or from their last checkpoint. The checkpoint/restart technique will not be an effective way to utilize exascale systems, because checkpointing will not scale to such highly parallel systems. With the potential that exascale systems will be having constant failures somewhere across the system, application software is not going to be able to rely on checkpointing to cope with faults. A new fault will occur before the application could be restarted, causing the application to get stuck in a state of constantly being restarted. For exascale systems, new fault-tolerant paradigms will need to be developed and integrated into both existing and new applications.

Research in the reliability and robustness of exascale systems for running large simulations is critical to the effective use of these systems. New paradigms must be developed for handling faults within both the system software and user applications. Equally important are new approaches for integrating detection algorithms, in both the hardware and software, and new techniques to help simulations adapt to faults.

Building energy efficiency into algorithms foundations

It is widely recognized that emerging constraints on energy consumption will have pervasive effects on HPC. Energy reduction depends on software as well as hardware. Power and energy consumption must now be added to the traditional goals of algorithm design, namely correctness and performance. The emerging metric of merit becomes performance per watt. Consequently, we believe it is essential to build power and energy awareness, control and efficiency into the foundations of our numerical libraries. First and foremost this will require us to develop standardized interfaces and APIs for collecting energy consumption data, just as PAPI has done for hardware performance counter data. Accurate and fine-grained measurement of power consumption underpins all tools that seek to improve such metrics (anything that cannot be measured cannot be improved). Secondly, we must use these tools to better understand the effects that energy-saving hardware features have on the performance of linear algebra codes. Finally, we must identify parameters and alternative execution strategies for each numerical library that can be tuned for energy-efficient executions, and to enhance our schedulers for better low-energy execution.

Sensitivity analysis
Many areas of modelling and simulation are still pushing to reach high-fidelity solutions to a given set of input conditions. However, as performance and fidelity improves, it becomes possible and imperative to study the sensitivity of a model to parameter variability and uncertainty, and to seek an optimal solution over a range of parameter values. The most basic form, the forward method for either local or global sensitivity analysis, simultaneously runs many instances of the model or its linearization, leading to an embarrassingly parallel execution model. The adjoint sensitivity method, with its powerful capabilities for efficiently computing the sensitivity of an output functional with respect to perturbations in a great many parameters, is a workhorse algorithm in weather prediction and in engineering design such as shape optimization. It requires the simulation of the forward and the adjoint problem; hence its parallelization will depend on the capability for highly efficient simulation.

Multiscale/multiphysics modelling
Engineering is increasingly operating at the micro- and nanoscales to achieve objectives at the macroscale. Models of these processes are intrinsically multiscale and multiphysics. For example, electrochemically reactive surfaces play a central role in the fabrication as well as the functional capabilities of an enormous variety of technological systems. Precise control of surface processes during fabrication is required in applications including on-chip interconnections between transistors, decorative and industrial coatings, batteries for electric vehicles, thin-film photovoltaic solar devices, magnetic materials, and patterned deposits for sensors. Surface processes are occurring at the nanoscale and must be modelled by kinetic Monte Carlo methods, whereas reactions and diffusion in the electrolyte can be modelled by deterministic methods (*i.e.*, PDEs). The two computations must be dynamically linked. Such a computation is very demanding and is currently consuming huge numbers of cycles on NCSA's supercomputers, with only modest resolution of the problem domain. Simulation is only the tip of the iceberg of this type of problem, where parameter estimation and optimal design are the ultimate goals and require orders of magnitude more computation time.

Cell biology is another area where processes operating at the microscale yield change at the macroscale (phenotypical change). In microscopic systems formed by living cells, the small numbers of some reactant molecules can result in dynamical behaviour that is discrete and stochastic rather than continuous and deterministic. An analysis tool that respects these dynamical characteristics is the stochastic simulation algorithm (SSA), which applies to well-stirred (spatially homogeneous) chemically reacting systems. Usually, a large ensemble of SSA simulations is used to estimate the probability density functions of important variables in the system. This leads to

an embarrassingly parallel implementation. At the same time, cells are not spatially homogeneous. Spatio-temporal gradients and patterns play an important role in many cellular processes. The modelling of stochastic diffusive transfers between subvolumes is an important challenge for parallelization.

Summary

The move to extreme-scale computing will require tools for understanding complex behaviour and for performance optimization to be based on a knowledge-oriented process. Performance models and expectations will be used to drive knowledge-based investigation and reasoning. It will raise the level at which tools inter-operate and can be integrated with the application development and execution environment. The challenges for performance analysis and tuning will grow, as performance interactions and factor analysis must involve a whole system perspective.

The co-design methodology is iterative, requiring frequent interactions among hardware architects, systems software experts, designers of programming models, and implementers of the science applications that provide the rationale for building extreme-scale systems. As new ideas and approaches are identified and pursued, some will fail. As with past experience, there may be breakthroughs in hardware technologies that result in different micro- and macro-architectures becoming feasible and desirable, but they will require rethinking of certain algorithmic and system software implementations.

13.1. Technology trends and their impact on exascale

The design of the extreme-scale platforms that are expected to become available in 2018 will represent a convergence of technological trends and the boundary conditions imposed by over half a century of algorithm and application software development. Although the precise details of these new designs are not yet known, it is clear that they will embody radical changes along a number of different dimensions as compared to the architectures of today's systems, and that these changes will render obsolete the current software infrastructure for large-scale scientific applications. The first step in developing a plan to ensure that appropriate system software and applications are ready and available when these systems come on line, so that leading-edge research projects can actually use them, is to carefully review the underlying technological trends that are expected to have such a transformative impact on computer architecture in the next decade. These factors and trends, which we summarize in this section, provide essential context for thinking about the looming challenges of tomorrow's scientific software infrastructure; by describing them, therefore, we lay the foundations for our narrative structure.

Technology trends
In developing a roadmap for the X-stack software infrastructure, the IESP has been able to draw on several thoughtful and extensive studies of impacts of the current revolution in computer architecture (Kogge *et al.* 2008, Sarkar *et al.* 2009). As these studies make clear, technology trends over the next decade – broadly speaking, increases of 1000× in capability over today's most massive computing systems, in *multiple* dimensions, as well as increases of similar scale in data volumes – will force a disruptive change in the form, function, and inter-operability of future software infrastructure components and the system architectures incorporating them. The momentous nature of these changes can be illustrated for several critical system-level parameters.

Concurrency. Moore's Law scaling in the number of transistors is expected to continue to the end of the next decade, at which point the minimal VLSI geometries will be as small as five nanometres. Unfortunately, the end of Dennard scaling means that clock rates are no longer keeping pace, and may in fact be reduced in the next few years to reduce power consumption. As a result, the exascale systems on which the X-stack will run will likely be composed of hundreds of millions of arithmetic logic units (ALUs). Assuming there are multiple threads per ALU to cover main-memory and networking latencies, applications may contain ten billion threads.

Reliability. System architecture will be complicated by the increasingly probabilistic nature of transistor behaviour due to reduced operating voltages, gate oxides, and channel widths/lengths resulting in very small noise margins. Given that state-of-the-art chips contain billions of transistors and the multiplicative nature of reliability laws, building resilient computing systems out of such unreliable components will become an increasing challenge. This cannot be cost-effectively addressed with pairing or TMR; rather, it must be addressed by X-stack software and perhaps even scientific applications.

Power consumption. Twenty years ago, HPC systems consumed less than a megawatt. The Earth Simulator was the first such system to exceed 10 MW. Exascale systems could consume over 100 MW, and few of today's computing centres have either adequate infrastructure to deliver such power or the budgets to pay for it. The HPC community may find itself measuring results in terms of power consumed, rather than operations performed. The X-stack and the applications it hosts must be conscious of this situation and act to minimize it.

Similarly dramatic examples could be produced for other key variables, such as *storage capacity*, *efficiency*, and *programmability*.

More importantly, a close examination shows that changes in these parameters are interrelated and not orthogonal. For example, scalability will be limited by efficiency, as are power and programmability. Other cross-correlations can be perceived through analysis. The US Defense Advanced Research Projects Agency (DARPA) Exascale Technology Study (Kogge *et al.* 2008) exposes power as the pace-setting parameter. Although an exact power consumption constraint value is not yet well defined, with upper limits of today's systems of the order of 5 megawatts, increases of an order of magnitude in less than 10 years will extend beyond the practical energy demands of all but a few strategic computing environments. A politico-economic pain threshold of 25 megawatts has been suggested (by DARPA) as a working boundary. With dramatic changes to core architecture design, system integration, and programming control over data movement, best estimates for CMOS-based systems at the 11-nanometre feature size is a factor of three to five times this amount. One consequence is that clock rates are unlikely to increase substantially. Among the controversial questions is how much instruction-level parallelism (ILP) and speculative operation is likely to be incorporated on a per processor core basis and the role of multi-threading in subsuming more of the fine-grained control space. Data movement across the system, through the memory hierarchy, and even for register-to-register operations will likely be the single principal contributor to power consumption, with control adding to this appreciably. Since future systems can ill afford the energy wasted by data movement that does not advance the target computation, alternative ways of hiding latency will be required in order to guarantee, as much as possible, the utility of every data transfer. Even taking into account the wastefulness of today's conventional server-level systems and the energy gains that careful engineering has delivered for systems such as BlueGene/P, an improvement of two orders of magnitude, will still be required.

As a result of these and other observations, exascale system architecture characteristics are beginning to emerge, though the details will become clear only as the systems themselves actually develop. Among the critical aspects of future systems, available by the end of the next decade, which we can predict with some confidence are the following:

- feature size of 22 to 11 nanometres, CMOS in 2018,
- total average of 25 picojoules per floating-point operation,
- approximately 10 billion-way concurrency for simultaneous operation and latency hiding,
- 100 million to 1 billion cores,
- clock rates of 1 to 2 GHz,
- multi-threaded, fine-grained concurrency of 10- to 100-way concurrency per core,

- hundreds of cores per die (varies dramatically depending on core type and other factors),
- global address space without cache coherence, and extensions to PGAS (*e.g.*, AGAS),
- 128-petabyte capacity mix of DRAM and non-volatile memory (most expensive subsystem),
- explicitly managed high-speed buffer caches; part of deep memory hierarchy,
- optical communications for distances >10 centimetres, possibly inter-socket,
- optical bandwidth of 1 terabit per second,
- systemwide latencies of the order of tens of thousands of cycles,
- active power management to eliminate wasted energy by currently unused cores,
- fault tolerance by means of graceful degradation and dynamically reconfigurable structures,
- hardware-supported rapid thread context switching,
- hardware-supported efficient message-to-thread conversion for message-driven computation,
- hardware-supported, lightweight synchronization mechanisms,
- 3D packaging of dies for stacks of four to ten dies, each including DRAM, cores, and networking.

Because of the nature of the development of the underlying technology, most of the predictions above have an error margin of +/–50 % or a factor of two, independent of specific roadblocks that may prevent us reaching the predicted value.

The list quoted above demonstrates the large variety of items that have to change significantly in order to reach the goal of a 1000× acceleration of HPC systems in the 2018–2020 timeframe. It is clear from this list that we cannot expect the current hardware technology to satisfy all these requirements. Two components that were traditionally considered less crucial than the processors are the memory system and the interconnect network. The perspective has changed drastically in this respect: data movement at low energy level has become a prime target and both the memory and the network are involved in this. We look in more detail at two of the most important directions that may help exaflop/s systems to come about: non-volatile memory and optical network components.

Non-volatile memory

The use of non-volatile memory, *i.e.*, memory that retains its contents when no current is applied, is important for two reasons. The first, obvious one

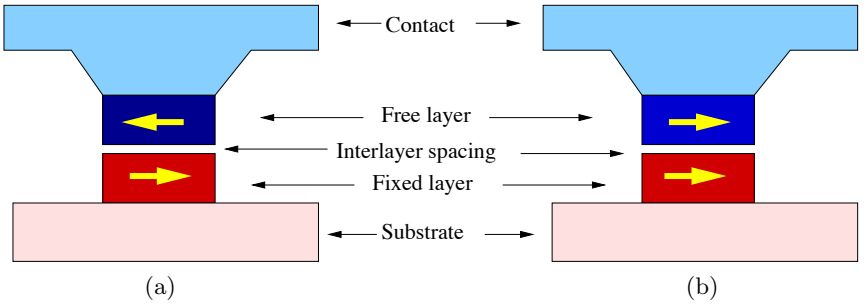


Figure 13.1. An MRAM memory cell:
(a) represents 0, (b) represents 1.

is that it is much more energy-efficient than the currently used DRAM technology, where the contents must be refreshed continuously. A second reason is connected with the present way DRAM is implemented: the feature size is currently in the range of 40 nm and it can still be shrunk somewhat by using 3D techniques instead of the planar technology that is used today (Intel and Samsung will ship 3D memory chips shortly). However, already the leak current occurring within the chips is quite significant, and it will only increase when the memory cells are more densely packed. This increases both energy costs and unreliability. Therefore new memory technologies are urgently needed.

Various interesting alternatives are actively being researched. Among them are magnetic RAM (MRAM), ferro-magnetic RAM (FRAM) and memristors (a special form of FRAM). From these three technologies spin torque transfer MRAM and FRAM are already in production, though still with a density that is not suitable for use in (HPC) memory. Both are, however, used in embedded processors and sensors where low energy consumption is of prime importance. FRAM and, consequently, memristors are based on the magnetic hysteresis effect, while MRAM is based on the giant magneto-resistive effect, as is also employed in present-day spinning disks. The MRAM implementation is, however, static. A memory cell is depicted in Figure 13.1.

When the magnetic field orientations in the fixed and free layers are opposite (Figure 13.1(a)), the total magnetic moment is much lower than when they are aligned (Figure 13.1(b)). This difference can be sensed and interpreted as a 0, resp. 1 value. The magnetic field of the free layer can be changed by a spin-polarized current, thus writing a 0 or a 1 value.

Hewlett-Packard, the first company that was able to demonstrate memristor memory, has teamed up with Hynix, a memory production company, to make commercial memristor products which are scheduled for late 2012 or early 2013.

please check projected dates

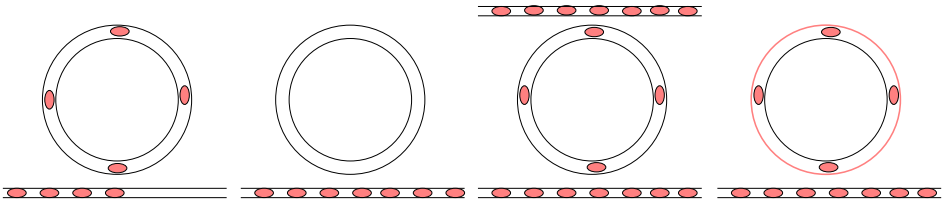


Figure 13.2. Three functions of a ring resonator. (a) It resonates in phase with the light waves and diverts the light signal. (b) It does not resonate, and lets the light signal pass. (c) Through resonation it transfers the signal to another wave guide, thus acting as a switch. (d) A doped resonator picks up a signal of a pre-defined wavelength and so acts as a signal detector.

Optical networks

Just as for memory, interconnect networks already consume a significant amount of energy, irrespective of whether they are used, because the network switches must be ready to pass on incoming messages at any time. In addition, the signalling speed of the wires that are employed start to pose problems with respect to signal integrity when the clock frequency of the communication engines increase. The increase of this frequency therefore has a negative impact both on the energy consumption (as in any electronic device) and on the reliability of the message transfer. This is why there is active research to implement the networks with optical components with as few electronic-optical transitions as possible. A big step forward in this respect is the development of the so-called ring resonator. A ring resonator is a minute glass ring with a size of ≈ 5 μm, and it is possible to make it resonate in phase with the light waves that pass along it in an optical wave guide. Figure 13.2 shows the three functions that the resonator can fulfil. According to its position and properties it can act as a modulator blocking or passing signals as desired (Figure 13.2(a, b)). When it is coupled with another wave guide and made to resonate with the right frequency, it will pass the signal on to this wave guide (Figure 13.2(c)), thus implementing a switching function. Lastly, it can act as a signal detector when the ring is doped to tune it to a signal of a desired wavelength (Figure 13.2(d)). The advantages of this optical switching are many. The bandwidth of the interconnection can be quite high because multiple wavelengths can be sent through the same wave guide. Furthermore, optical signals do not interfere with each other as electrical signals do when they are near each other. In addition, the power consumption is much lower than that of its electronic equivalent. Unfortunately there are also drawbacks. The rings are very sensitive to temperature changes, so for proper operation they must either be

in an extremely well-controlled environment with respect to temperature, or provisions must be made on the rings themselves to keep their temperature constant. This is technologically possible but greatly complicates the design.

At present, optical switches are still in the laboratory phase or moving to a preproduction stage, so it will take another few years for them to emerge in commercial HPC systems. Yet there is little doubt that this path will be taken, as there are virtually no alternatives.

Science trends

A basic driver of the IESP is the fact that the complexity of advanced challenges in science and engineering continues to outpace our ability to adequately address them through available computational power. Many phenomena can be studied only through computational approaches: well-known examples include simulating complex processes in climate and astrophysics. Increasingly, experiments and observational systems are finding that not only are the data they generate exceeding petabytes and rapidly heading toward exabytes, but the computational power needed to process the data is also expected to be in exaflop/s range.

A number of reports and workshops have identified key science challenges and applications of societal interest that require computing at exaflops levels and beyond (Stevens, Zacharia and Simon 2008, US Department of Energy 2008, 2009*a*–*i*, 2010, US National Research Council Committee 2008). Here we summarize some of the significant findings on the scientific necessity of exascale computing; we focus primarily on the need for the software environments needed to support the science activities. The US Department of Energy held eight workshops in the past year that identified science advances and important applications that will be enabled through the use of exascale computing resources. The workshops covered the following topics: climate, high-energy physics, nuclear physics, fusion energy sciences, nuclear energy, biology, materials science and chemistry, and national nuclear security. The US National Academy of Sciences published the results of a study in the report 'The potential impact of high-end capability computing on four illustrative fields of science and engineering' (US National Research Council Committee 2008). The four fields were astrophysics, atmospheric sciences, evolutionary biology, and chemical separations.

Likewise, the US National Science Foundation has embarked on a petascale computing programme that has funded dozens of application teams via its Peta-Apps and PRAC programmes, across all areas of science and engineering, to develop petascale applications, and is deploying petaflops systems. It has commissioned a series of task forces to help plan for the transition from petaflops to exaflops computing facilities, to support the software development necessary, and to understand the specific science and engineering needs beyond petascale.

Similar activities are seen in Europe and Asia, all reaching similar conclusions: significant scientific and engineering challenges in both simulation and data analysis already exceed petaflops and are rapidly approaching exaflop-class computing needs. In Europe, the Partnership for Advanced Computing in Europe (PRACE) involves twenty partner countries, supports access to world-class computers, and has activities aimed at supporting multi-petaflops and eventually exaflop-scale systems for science. The European Union is also planning to launch projects aimed at petascale and exascale computing and simulation. Japan has a project to build a 10-petaflop/s system and has historically supported the development of software for key applications such as climate. As a result, scientific and computing communities, and the agencies that support them in many countries, have been meeting to plan joint activities that will be needed to support these emerging science trends.

To give a specific and timely example, a recent report, 'Science prospects and benefits with exascale computing' (Kothe 2007, p. 9) states that the characterization of abrupt climate change will require sustained exascale computing in addition to new paradigms for climate change modelling. The types of questions that could be tackled with exascale computing (and cannot be tackled adequately without it) include the following:

- How do the carbon, methane, and nitrogen cycles interact with climate change?
- How will local and regional water, ice, and clouds change with global warming?
- How will the distribution of weather events, particularly extreme events, determine regional climate change with global warming?
- What are the future sea-level and ocean circulation changes?

Among the findings of the astrophysics workshop and other studies are that exascale computing will enable cosmology and astrophysics simulations aimed at the following:

- measuring the masses and interactions of dark matter,
- understanding and calibrating supernovae as probes of dark energy,
- determining the equation of state of dark energy,
- understanding the nature of gamma-ray bursts.

Energy security

The search for a path forward in ensuring sufficient energy supplies in the face of a climate-constrained world faces a number of technical challenges, ranging from issues related to novel energy technologies, to issues related to making existing energy technologies more (economically) effective and safer, to issues related to the verification of international agreements regarding the

emission (and possible sequestration) of CO_2 and other greenhouse gases. Among the science challenges are the following:

- verification of 'carbon treaty' compliance,
- improvement in the safety, security, and economics of nuclear fission,
- improvement in the efficiency of carbon-based electricity production and transportation,
- improvement in the reliability and security in the (electric) grid,
- nuclear fusion as a practical energy source.

Computational research will also play an essential role in the development of new approaches to meeting future energy requirements (*e.g.*, wind, solar, biomass, hydrogen, and geothermal), which in many cases will require exascale power.

Industrial applications, such as simulation-enhanced design and production of complex manufactured systems and rapid virtual prototyping, will also be enabled by exascale computing. To characterize materials deformation and failure in extreme conditions will require atomistic simulations on engineering time scales that are out of reach with petascale systems.

A common theme in all of these studies of the important science and engineering applications that are enabled by exaflops computing power is that they have complex structures and present programming challenges beyond just scaling to many millions of processors. For example, many of these applications involve multiple physical phenomena spanning many decades of spatial and temporal scale. As the ratio of computing power to memory grows, the 'weak scaling' that has been exploited for most of the last decade will increasingly give way to 'strong scaling', which will make scientific applications increasingly sensitive to overhead and noise generated by the X-stack. These applications are increasingly constructed from components developed by computational scientists worldwide, and the X-stack must support the integration and performance portability of such software.

Key requirements imposed by trends on the X-stack

The cited trends in technology and applications will impose severe constraints on the design of the X-stack. Below are cross-cutting issues that will affect all aspects of system software and applications at exascale.

Concurrency. A 1000× increase in concurrency for a single job will be necessary to achieve exascale throughput. New programming models will be needed to enable application groups to address concurrency in a more natural way. This capability will likely have to include 'strong scaling' because growth in the volume of main memory will not match that of the processors. This in turn will require minimizing any X-stack overheads that might otherwise become a critical Amdahl fraction.

Energy. Since much of the power in an exascale system will be expended moving data, both locally between processors and memory as well as globally, the X-stack must provide mechanisms and APIs for expressing and managing data locality. These will also help minimize the latency of data accesses. APIs also should be developed to allow applications to suggest other energy saving techniques, such as turning cores on and off dynamically, even though these techniques could result in other problems, such as more faults/errors.

Resiliency. The VLSI devices from which exascale systems will be constructed will not be as reliable as those used today. All software, and therefore all applications, will have to address resiliency in a thorough way if they are to be expected to run at scale. Hence, the X-stack will have to recognize and adapt to errors continuously, as well as provide the support necessary for applications to do the same.

Heterogeneity. Heterogeneous systems offer the opportunity to exploit the extremely high performance of niche market devices such as GPUs and game chips (*e.g.*, STI Cell) while still providing a general-purpose platform. An example of such a system today is Tokyo Tech's Tsubame, which incorporates AMD Opteron CPUs along with ClearSpeed and NVIDIA accelerators. Simultaneously, large-scale scientific applications are also becoming more heterogeneous, addressing multiscale problems spanning multiple disciplines.

I/O and memory. Insufficient I/O capability is a bottleneck today. Ongoing developments in instrument construction and simulation design make it clear that data rates can be expected to increase by several orders of magnitude over the next decade. The memory hierarchy will change based on both new packaging capabilities and new technology. Local RAM and NVRAM will be available either on or very close to the nodes. The change in memory hierarchy will affect programming models and optimization.

Relevant politico-economic trends

The HPC market is growing at approximately 11 % per year. The largest-scale systems, those that will support the first exascale computations at the end of the next decade, will be deployed by government computing laboratories to support the quest for scientific discovery. These capability computations often consume an entire HPC system and pose difficult challenges for concurrent programming, debugging and performance optimization. Thus, publicly funded computational scientists will be the first users of the X-stack and have a tremendous stake in seeing that suitable software exists, which is the *raison d'être* for IESP.

In the late 1980s, the commercial engineering market place, spanning diverse fields such as computer-aided engineering and oil reservoir modelling, used the same computing platforms and often the same software as the scientific community. This is far less the case today. The commercial workload tends to be more capacity-oriented, involving large ensembles of smaller computations. The extreme levels of concurrency necessary for exascale computing suggests that this trend may not change, so the demand for those features of the X-stack is not unique to exascale computing for scientific computing. On the other hand, the HPC vendor community is eager to work with, and leverage the research and development effort of, the IESP software community. To that end, plans for cooperation and coordination between the IESP software and the HPC vendor community are being developed.

REFERENCES[7]

C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu and W. Zwaenepoel (1995), 'TreadMarks: Shared memory computing on networks of workstations', *IEEE Computer* **29**, 18–28. www.cs.rice.edu/~willy/TreadMarks/papers.htm

G. Bell (1999), The next ten years of supercomputing. In *Proc. 14th Supercomputer Conference* (H. W. Meuer, ed.), MaxionMedia (CD-ROM).

N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and Wen-King Su (1995), 'Myrinet: A gigabit-per-second local area network', *IEEE Micro* **15**, 29–36.

R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald and R. Menon (2001), *Parallel Programming in OpenMP*, Morgan Kaufmann.

B. Chapman, G. Jost and R. van der Pas (2007), *Using OpenMP*, MIT Press.

D. Chazan and W. Miranker (1969), 'Chaotic relaxation', *Linear Algebra Appl.* **2**, 199–222.

D. E. Culler, J. P. Singh and A. Gupta (1998), *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann.

D. W. Doerfler (2005), An analysis of the PathScale Inc. InfiniBand host channel adapter, InfiniPath. Sandia report SAND2005-5199, Sandia National Laboratories.

J. Dongarra (2011), Performance of various computers using linear equations software. www.netlib.org/benchmark/performance.ps

J. Dongarra, P. Beckman, *et al.* (2011), 'The International Exascale Software roadmap', *Internat. J. High Performance Computer Applications*, **25**, 3–60.

M. J. Flynn (1972), 'Some computer organizations and their effectiveness', *IEEE Trans. Comput.* **C-21**, 948–960.

[7] The URLs cited in this work were correct at the time of going to press, but the publisher and the authors make no undertaking that the citations remain live or are accurate or appropriate.

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

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

D. B. Gustavson and Q. Li (1995), Local-area multiprocessor: The Scalable Coherent Interface. SCIzzL Report, Department of Computer Engineering, Santa Clara University, available via www.scizzl.com.

High Performance Fortran Forum (1993), 'High Performance Fortran language specification', *Scientific Programming*, **2**, 1–170.

R. W. Hockney and C. R. Jesshope (1988), *Parallel Computers II*, Adam Hilger.

T. Horie, H. Ishihata, T. Shimizu, S. Kato, S. Inano and M. Ikesaka (1991), AP1000 architecture and performance of LU decomposition. In *Proc. Internat. Symposium on Supercomputing*, pp. 46–55.

D. V. James, A. T. Laundrie, S. Gjessing and G. S. Sohi (1990), 'Scalable coherent interface', *IEEE Computer* **23**, 74–77. See also 'Scalable coherent interface': sunrise.scu.edu/

P. M. Kogge *et al.* (2008), ExaScale computing study: Technology challenges in achieving exascale systems. US Defense Advanced Research Projects Agency report, DARPA Information Processing Techniques Office. users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf

D. B. Kothe (2007), Science prospects and benefits with exascale computing. Report ORNL/TM-2007/232, National Center for Computational Sciences, Oak Ridge National Laboratory. www.nccs.gov/wp-content/media/nccs_reports/Science%20Case%20_012808%20v_final.pdf

J. Langou, J. Langou, P. Luszczek, J. Kurzuk, A. Buttari and J. J. Dongarra (2006), Exploiting the performance of 32-bit floating point arithmetic in obtaining 64-bit accuracy. In *Proc. 2006 ACM/IEEE Conference on Supercomputing*.

H. W. Meuer (1994), The Mannheim supercomputer statistics 1986–1992. Top500 report 1993, University of Mannheim.

H. W. Meuer, E. Strohmaier, J. J. Dongarra and H. D. Simon (2011), Top500. Available at www.top500.org.

V. Sarkar *et al.* (2009), ExaScale software study: Software challenges in extreme scale systems. US Defense Advanced Research Projects Agency report, DARPA Information Processing Techniques Office. users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf

C. Schow, F. Doany and J. Kash (2010), 'Get on the optical bus'. *IEEE Spectrum*, September 2010, 31–35.

T. Shanley (2002), *InfiniBand Network Architecture*, Addison-Wesley.

H. D. Simon (1994), High performance computing in the US. Top500 report 1993, University of Mannheim, 116–147.

M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J. Dongarra (1998), *MPI: The Complete Reference*, Vol. 1, *The MPI Core*, MIT Press.

D. H. M. Spector (2000), *Building Unix Clusters*, O'Reilly.

A. J. van der Steen (1990), Exploring VLIW: Benchmark tests on a multiflow TRACE 14/300. Technical Report TR-31, Academic Computing Centre Utrecht.

A. J. van der Steen, ed. (1995), *Aspects of Computational Science*, NCF.

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

A. J. van der Steen (2010), Overview of recent supercomputers. www.euroben.nl, directory reports/.)

T. L. Sterling, J. Salmon, D. J. Becker and D. F. Savaresse (1999), *How to Build a Beowulf*, MIT Press.

R. Stevens, T. Zacharia and H. Simon (2008), Modeling and simulation at the exascale for energy and the environment. Town Hall Meetings Report, US Department of Energy Office of Advance Scientific Computing Research. www.sc.doe.gov/ascr/ProgramDocuments/Docs/TownHall.pdf

E. Strohmaier, J. J. Dongarra, H. W. Meuer and H. D. Simon (1999), 'The marketplace of high-performance computing', *Parallel Computing* **25**, 1517.

US Department of Energy (2008), Challenges in climate change science and the role of computing at the extreme scale. Department of Energy report, *Scientific Grand Challenges Workshop Series*. www.er.doe.gov/ascr/ProgramDocuments/Docs/ClimateReport.pdf.

US Department of Energy (2009*a*), Forefront questions in nuclear science and the role of high performance computing: Summary report. Department of Energy Workshop Report, Washington DC. extremecomputing.labworks.org/nuclearphysics/PNNL_18739_↩ onlineversion_opt.pdf

US Department of Energy (2009*b*), Cross-cutting technologies for computing at the exascale. Department of Energy report, *Scientific Grand Challenges Workshop Series*. extremecomputing.labworks.org/crosscut/CrosscutWSFinalReptDraft02.pdf

US Department of Energy (2009*c*), Fusion energy science and the role of computing at the extreme scale. Department of Energy report, *Scientific Grand Challenges Workshop Series*. extremecomputing.labworks.org/fusion/PNNL_Fusion_final19404.pdf

US Department of Energy (2009*d*), Science based nuclear energy systems enabled by advanced modeling and simulation at the extreme scale. Department of Energy Workshop Report, Washington, DC. www.er.doe.gov/ascr/ProgramDocuments/Docs/SC-NEWorkshopReport.pdf

US Department of Energy (2009*e*), Discovery in basic energy sciences: The role of computing at the extreme scale. Department of Energy report, *Scientific Grand Challenges Workshop Series*.

US Department of Energy (2009*f*), Opportunities in biology at the extreme scale of computing. Department of Energy report, *Scientific Grand Challenges Workshop Series*. www.er.doe.gov/ascr/ProgramDocuments/Docs/BiologyReport.pdf

US Department of Energy (2009*g*), Scientific grand challenges in national security: The role of computing at the extreme scale. Department of Energy report, *Scientific Grand Challenges Workshop Series*.
www.er.doe.gov/ascr/ProgramDocuments/Docs/NNSAGrandChallenges↩
Report.pdf

US Department of Energy (2009*h*), Architectures and technology for extreme scale computing. Department of Energy report, *Scientific Grand Challenges Workshop Series*.

US Department of Energy (2009*i*), Scientific challenges for understanding the quantum universe and the role of computing at extreme scale: Summary report. Department of Energy report, *Scientific Grand Challenges Workshop Series*.
extremecomputing.labworks.org/highenergyphysics/reports/↩
HEPreport101609_final.pdf

US Department of Energy (2010), Exascale workshop panel meeting report. Department of Energy report, *Scientific Grand Challenges Workshop Series*.
www.er.doe.gov/ascr/ProgramDocuments/Docs/Trivelpiece↩
ExascaleWorkshop.pdf

US National Research Council Committee (2008), The potential impact of high-end capability computing on four illustrative fields of science and engineering. US National Academy of Sciences.

G. V. Wilson (1994), Chronology of major developments in parallel computing and supercomputing.
ftp://ftp.cs.toronto.edu/csri-technical-reports/312/csri312.ps.gz

P. R. Woodward (1996), 'Perspectives on Supercomputing', *Computer* **10**, 99–111.

Online resources

Co-Array Fortran: www.co-array.org/
DSM systems: www.cs.umd.edu/~keheler/dsmbiblio/dsmbiblio.html
OpenMP Application Interface, version 2.5: www.openmp.org/
Task Force on Cluster Computing: www.clustercomputing.org
Unified Parallel C: upc.gwu.edu/