# 50 & 25 YEARS AGO

EDITOR **ERICH NEUHOLD**
University of Vienna
erich.neuhold@univie.ac.at

## JUNE 1968

In its early years, *Computer* was published bimonthly. Stay tuned for more interesting historical highlights in the upcoming July 2018 issue.

## JUNE 1993

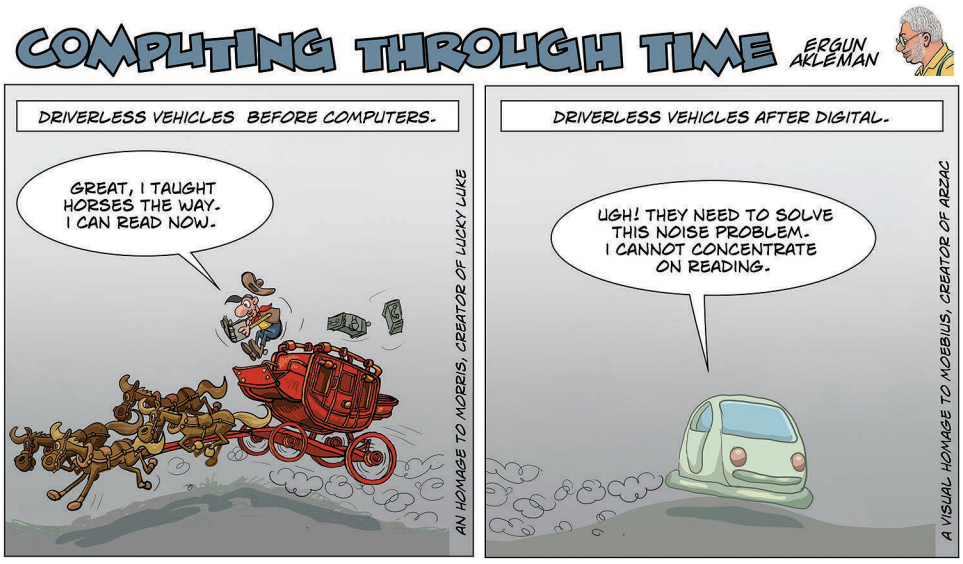
*www.computer.org/csdl/mags/co/1993/06/index.html*

**High-Performance Computing (HPC):** Lighting the Way (p. 6) "HPC refers to numerically intensive computations, the kind of use that is always reaching forward to the next generation in system architecture. What HPC users want is simple: blinding performance and high-speed system interconnects, with minimal power consumption and easy manufacture (read, 'inexpensive'). Technology is responding to those demands. Consider processor speed. Just 10 years ago the fastest CPUs were vector processors capable of 30–70 million floating-point operations per second: vector and highly parallel machines now achieve 20–40 gigaflops. The technological improvements have trickled down to less expensive computer strata with the result that workstations now surpass the top-of-the-line supercomputers of a decade ago. ... Demands for performance are also changing the way we think about computation, moving us toward parallel processing. ... Ever-increasing computational capabilities expand the possibilities for research to a startling degree. The demand appears to be limitless." *[Editor's note: This issue mostly concerns scientific computing and technological applications. Apparently, the time was not right to think about the Web with its rather unlimited pool of data that requires massive computing power for analysis. HPC technologies have continued to grow, following, to a large degree, Moore's law.]*

**Heterogeneous Computing (HC)** (p. 18) "[HC] is the well-orchestrated and coordinated effective use of a suite of diverse high-performance machines (including parallel machines) to provide superspeed processing for computationally demanding tasks with diverse computing needs. ... An HC environment must contain the following components: a set of heterogeneous machines, an intelligent high-speed network connecting all machines, and a (user-friendly) programming environment. ... Moreover, many applications need to process information at more than one level concurrently, with different types of parallelism at each level. ... HC requires machine-independent and portable parallel programming languages and tools. This requirement creates the need for designing cross-parallel compilers for all machines in the environment, and parallel debuggers for debugging cross-machine code."

**Jade: A High-Level, Machine-Independent Language for Parallel Programming** (p. 28) "Jade's abstractions of sequential execution and shared memory simplify programming. The system automatically manages parallelism and communication of data, and Jade programs run on a variety of parallel platforms. ... Every Jade program is a serial program augmented with constructs that declare how parts of the program access data. Because Jade preserves the semantics of the serial program, Jade programs execute deterministically. This property considerably simplifies the debugging of parallel applications. Jade programs access data using a single, global address space." *[Editor's note: At the time, Jade raised enthusiastic support because the theoretical soundness of its concepts had been shown. However, enthusiasm soon fizzled.]*

**An Architectural Framework for Supporting Heterogeneous Instruction-Set Architectures** (p. 39) "We are particularly interested in examining the potential for improving performance of existing software written for CISC (complex instruction-set computers) architectures (which we call the base system) by running it on RISC (reduced instruction-set computers), ... (which we call the native system). ... One way to migrate base software is to have the native architecture interpret the instruction set for which the application was originally developed, but this solution suffers from performance drawbacks. Another possibility is to recompile the application and produce code for the native architecture, but this is feasible only if the application is written wholly in a portable high-level language and the source code is available. The solution




we advocate is based on translating base-application object code into native object code. We propose hardware that can execute both types of code and a mechanism that can switch between the code types at certain points in the application execution. Thus, the proposed machine contains a migrant engine that supports execution of base code and a native engine that executes native code." *[Editor's note: The problem discussed is also known as binary translation, where a source code of one architecture is translated into the source code of another architecture. Many translations have succeeded but also encountered problems due to architectural mismatches. Interpreters have been more successful but at the cost of serious performance losses. Here, the solution is to have both architectures available, although this isn't suitable if the old architecture is going to be replaced.]*

**Object Orientation in Heterogeneous Distributed Computing Systems** (p. 57) "Because computers are becoming commonplace in routine business functions, future information processing environments will likely consist of vast networks of heterogeneous, autonomous, and distributed computing resources, including computers (from mainframe to personal), information-intensive applications, and data (files and databases). The telecommunications infrastructure provides a clear example of such an environment. The first phase in creating integrated, large-scale, distributed information systems is interconnectivity. Until recently, most computers were stand-alone systems unable to communicate with other systems. Two or more computing resources are interconnected if they can exchange messages. The second phase is the more ambitious goal of interoperability. Interoperability at this level supports the integration of heterogeneous information in, for example, advanced multimedia applications, and supports information storage in integrated repositories." *[Editor's note: Interoperability and/or information integration is still an intensively researched subject that has moved from distributed processing to grids, clouds, the Web and, more recently, the Internet of Things. Solutions beyond research are still slow in coming.]*

**Event Reconstruction in High-Energy Physics** (p. 68) "Cooperative Processes Software, a parallel programming toolkit developed at [the Fermi National Accelerator Laboratory (Fermilab)], runs as a collection of processes distributed over a network of more than 350 heterogeneous Unix-based workstations. ... The first pass, called event reconstruction, would take about 100 years to compute on a VAX 11/780. Because the problem maps so obviously onto a parallel architecture, Fermilab formed an Advanced Computer Program group in 1984 to find a cost-effective solution to this problem. The ACP Parallel Processing System was established as a result of the program's efforts."

**The Open Channel: A Distributed Heterogeneous Supercomputing Management System** (p. 78) "This framework provides a technique for efficiently managing execution of applications in a distributed heterogeneous supercomputing system. The technique is based on code profiling and machine benchmarking. ... We need new code-profiling methods that incorporate these [architectural] details and thus support more accurate scheduling and mapping decisions with regard to application execution. ... Analytical benchmarking for a [Distributed Heterogeneous Supercomputing System] must therefore estimate machine performance on each part of the application as well as performance of the I/O subsystem."

**The Open Channel: Death or Glory Programming (DOG-P)** (p. 136) "For years, programmers have been told it's wrong to write programs without structure. ... Let's look at the problems with structured methods and see how DOG-P addresses them: First, programmers trained in different structured methods have difficulty understanding each other's programs. In DOG-P, no one can understand other programs, but nobody cares. The first rule is to avoid amending the programs; write them again from scratch. More lines of code means higher productivity. Second, structured programming results in programs with so many modules that it's difficult to know where to look for errors. The DOG-P solution is to have only one module. That way you always know which module is wrong. Third, it's very difficult to record the purpose and location of reusable modules. How do you know if there's a bit of code out there that is exactly what you need? And reusable code needs a vast amount of documentation to describe what inputs and outputs it uses. My solution: Don't reuse code." C