Fortran faces the future at 50

This year marks the 50th anniversary of the computer programming language Fortran. Since its release in 1957 it has allowed physicists to tackle a huge variety of problems, from forecasting the weather to looking for new particles in data from accelerators, explain **Peter Crouch, Clive Page** and **John Pelan**

NASA

Crown copyright, the Met Office

Then and now
In 1957 the IBM 704 (left) was the first computer to run Fortran, while today the latest version of the language can be found powering climate simulations on the UK Met Office's NEC SX-8 supercomputer.

It may come as a surprise to find out that computational physics – the discipline that today uses powerful supercomputers to crunch through vast amounts of data on, say, the Earth's climate – actually began in the late 1920s, some three decades before the first electronic computers were built. One of the people responsible for kick-starting this field was the physicist and mathematician Douglas Hartree, who at the time was trying to calculate atomic wavefunctions in order to determine the structural properties of atoms. While the Schrödinger equation can easily be solved for the hydrogen atom, repeating the feat for multi-electron atoms involves calculations that at the time were generally thought to be intractable. Hartree, however, developed a technique known as the self-consistent field method, which allowed these problems to be solved numerically. Unfortunately, his method was so laborious using the facilities available at the time – mechanical calculators operated by humans – that very little use was made of it until electronic computers became available.

It was not until the Second World War, however, that the development of such computers took off, when they proved invaluable for code-breaking and generating artillery firing tables. After the war, many of the scientists who had taken part in these projects were keen to use these devices for their peace-time research. Hartree himself was involved in these early applications, including advising the US military on the use of ENIAC (the first large-scale reprogrammable electronic computer) for calculating the ballistic properties of different types of ammunition.

At the time, programming computers was an esoteric art. They had to be fed instructions in "machine code" – a language that describes every single addition, subtraction and so on in the precise order required – which was tedious to write, error-prone and required very specialist knowledge (see box on page 32). If computers were to be applied – and hence sold – more widely, then programming them had to become considerably easier, and this required a language that bore a much closer resemblance to the mathematical problems being tackled.

With this in mind, in 1954 a team of researchers at IBM led by John Backus, who died in March this year aged 82, embarked on the creation of Fortran (Formula Translation). This was the first successful "high-level" language – i.e. it used a program known as a "compiler" to translate commands describing the mathematical operations to be performed into instructions in machine code. Three years later, the first Fortran compiler became commercially available, and it was not long before physicists realized the opportunities that it offered. Since then it has evolved through many versions, each more powerful than the last, and even now Fortran is still the language of choice in many areas of physics.

A perfect fit

Fortran is well suited to physics research for several reasons. For one, it is easy to learn and use. It also has excellent facilities for handling large amounts of data, which is important in physics where many problems involve large, multidimensional datasets – for example there are often three spatial dimensions, with time as a fourth, and wavelength or frequency as a fifth. Such datasets are easy to manipulate in Fortran because almost any operation that can be performed on a single number can also be performed on an array of numbers without any extra work from the programmer.

Peter Crouch, Clive Page and **John Pelan** are members of the British Computer Society Fortran Specialist Group, e-mail pccrouch@bcs.org.uk, www.fortran.bcs.org

What's in a language?

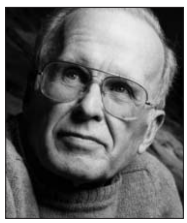
At their fundamental level, computers perform very primitive operations, such as adding two variables together, comparing the relative values of two variables, storing and fetching variables to and from main memory and so on. By arranging sequences of these simple operations, more complex tasks can be built up, like solving differential equations or predicting tomorrow's weather. It would be impractical, if not impossible, to program these complicated tasks manually, i.e. explicitly describing every single addition, subtraction and so on in the precise order required. But that is exactly what had do be done with early electronic computers. This process is called programming in "machine code" or "assembly language".

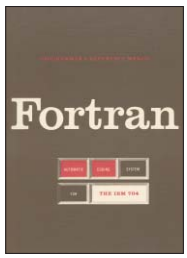
It is clearly desirable to be able to program at a higher level – i.e. where one is less concerned with the precise machinations of the computer and more concerned with the mathematical problems at hand. Such "high-level" languages – like Fortran – enable you to say "multiply these two matrices together" or "find me the lowest number from this set of numbers". The key component of these languages is a program known as a "compiler", which takes the high-level program and applies the rules of the language to generate instructions in machine code, preferably in an optimal manner so that the computer is used efficiently. Aside from reducing the programming effort required, high-level languages make finding faults in a program easy because erroneous lines of code are revealed when the program is compiled. Another advantage is that since using a compiler effectively separates the program from the underlying hardware – and indeed compensates for the differences between different hardware configurations – a high-level program can be run on any machine that supports an appropriate compiler.

Since the 1990s, so-called object-oriented languages have also become popular with physicists. Object-oriented programming was intended to make it easier to write complex but high-quality programs because object-oriented programs are built up using many small modules, each of which constitutes a program in its own right. As such, an object-oriented program may be seen as a collection of co-operating objects, as opposed to a traditional high-level program that is just a list of instructions to the computer. The downside is that the budding object-oriented programmer needs to learn about many arcane computer-science concepts, whereas the Fortran programmer does not. In addition, the problems solved by physicists tend to need more attention to algorithms rather than data structures, thus making object-oriented programming less relevant.

Some modern high-level languages can also be "interpreted" – meaning that the commands are converted to machine code as they are entered without going through the compilation step. Such interpreted languages – such as Java and Matlab – can be easy to use, but they suffer from two main disadvantages compared with compiled languages such as Fortran. The first is that language errors, which would be picked up by a compiler, are only revealed when the program is run. The second is that such languages are slower than a compiled language since the translation into machine code has to be repeated to some extent, and cannot be optimized in the same way. Typical number-crunching programs written in Java, for example, take about twice as long to execute as those written in Fortran, so interpreted languages are impractical for many physics applications.

IBM





Programming pioneer
IBM's John Backus, who led the team that created Fortran in the 1950s, along with the front cover of the original Fortran codebook.

In addition, Fortran produces code that can be executed exceptionally quickly. The reason for this was that if Fortran was to be accepted as an improvement over programming directly in machine code, it had to produce programs that would run as quickly as these "hand-crafted" machine-code programs. The compiler was therefore written to generate programs that used the hardware as efficiently as possible. This optimizing behaviour has been a hallmark of Fortran compilers ever since. It is particularly useful for operations where a quick result is needed, or when there is a very large amount of data to crunch through. Indeed, in many areas of physics the amount of data to be analysed is so huge that recently researchers have been turning to cluster computing (where several processors are linked together and work in parallel). Fortran offers good support for parallel-machine architectures because it has many tools for dealing with data structures such as arrays.

Astronomers and space physicists were among the first researchers to embrace Fortran, since they needed to analyse large amounts of data collected by telescopes. By the late 1970s several projects had been set up to develop dedicated software packages – such as NASA's FTOOLS – for analysing data from many different sources, and these were mostly written in Fortran. Today, cosmologists and theoretical physicists need fast software and clusters of powerful processors to simulate the workings of stars and to study the evolution of galaxies, or even the entire universe. Because of its run-time efficiency and support for parallel hardware, Fortran is still the language of choice in many cases.

These features also made Fortran attractive to high-energy physicists. For example, researchers at the CERN particle-physics lab near Geneva have been using Fortran to analyse the data from its detectors since 1961 and it remained the main language used there until well into the 1990s. In recent years, however, many other languages have been introduced, such as C and its object-oriented descendent C++ (see box above). Indeed, Fortran has now been overtaken by C++ for data processing in particle physics, although there are some areas of accelerator design and tuning that have been newly coded in the latest versions of Fortran.

More down-to-earth areas where Fortran has been invaluable are weather forecasting – where speed is of the essence – and climate modelling, which involves analysing vast amounts of data and performing complex mathematical operations. In fact, the climate model HadSM3, which was developed by the UK Met Office's Hadley Centre, contains over one million lines of Fortran code. Fortran is also widely used by geophysicists and seismologists to analyse seismic waves in order to locate earthquakes or explosions, or to search for oil, gas and minerals. The UK's Atomic Weapons Establishment (AWE), for example, uses Fortran programs to distinguish between earthquakes and clandestine underground nuclear tests, as well as to simulate nuclear explosions now that testing nuclear weapons for real is banned. Like climate modelling, this is a very computationally intensive task because seismic events produce several different types of waves, each of which has to be analysed to determine its characteristic frequency and amplitude.

Moving with the times

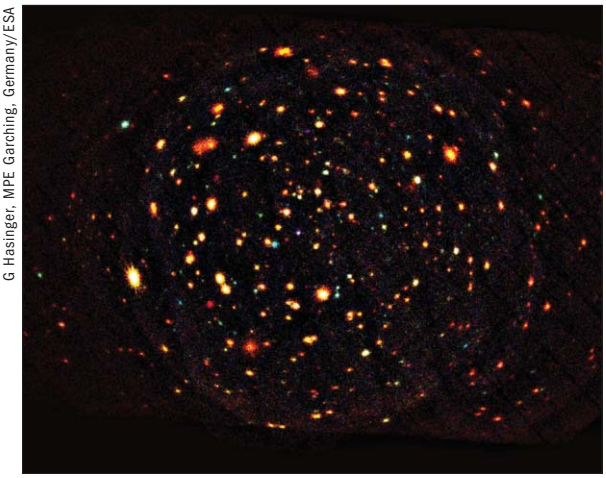
The current version of Fortran – as used in HadSM3 – would be almost unrecognizable to Backus and his colleagues. The language was originally designed to be easy for scientists and engineers to learn so that they could readily program their own problems into a computer. But as IBM and the other computer manufacturers became more aware of how physicists actually wanted to use computers, these firms steadily added new features to their compilers. By 1964 there were 43 different Fortran compilers in total.

However, the large number of compilers led to problems in exchanging programs between computers. In 1962 the American Standards Association therefore asked the Computer Business Equipment Manufacturers Association (CBEMA) to develop a standard for Fortran, which was basically a list of properties and facilities that each standard-compliant compiler must provide. In 1966 the CBEMA brought out the first ever standard for a programming language – ANSI Standard X3.9-1966, now known as Fortran 66 – giving rise to a "portable" language that could be used with all compliant compilers. The success of this standard was assured because the US government required that all Fortran compilers that it bought had to conform to it. In the 1970s Fortran was revised to incorporate the latest programming developments and to remove some anomalies from the Fortran 66 standard. The next standard, Fortran 77, introduced a new data type, and allowed arrays to have up to seven dimensions. It was also adopted as an international standard, ISO 1539:1980.

During the 1980s there were major disagreements in the Fortran community about the direction the language should take. There were clashes between traditionalists who did not want to lose any existing features and revisionists who wanted to add new features; and also between "featurists" whose primary interest was in particular features and "generalists" who were concerned with the language as a whole. Resolving these differences took considerable time and effort, and the next revision of the standard – known as Fortran 90 – was not published until 1991. The outcome of the debate was that Fortran 90 introduced a set of tools that – among other things – made it much easier to work with arrays of numbers, and allowed modular programming.

By this time, however, advances in hardware and programming-language theory had resulted in many new languages being introduced. In particular, increasing processor speeds meant that it was not essential to use a compiled language for small-scale problems. Instead, each command could be translated immediately into machine code as the programmer entered it, which led to the development of non-compiled languages like Java, and integrated mathematical and graphical environments such as Matlab and Mathematica (see box opposite). In addition, many university physics departments began to outsource their teaching of computing to computer-science departments, where many staff did not know that Fortran had been continually evolving and preferred to teach one of the newer languages such as C, Pascal or Modula-2. At this time many astronomers and particle physicists also started to view Fortran as outdated and switched to using C or C++ instead.

Nevertheless, there has recently been a resurgence of interest in Fortran. The speed of computer processors – although well known for doubling about every 18 months – has now reached something of a plateau, while the amount of data physicists want to analyse has continued to grow rapidly. As a result, performance gains are now likely to be achieved through the use of parallel computing. Physicists are therefore turning to Fortran once again, because of its good support for parallel-machine architectures. Some supercomputers are currently running a version of Fortran that uses data structures known as "co-arrays". These provide a simple method of running identical code on a computer cluster or supercomputer while allowing simple but controllable transfers of data from one processor to another. This feature is expected to be standardized in the next version of the international Fortran standard.



G Hasinger, MPE Garching, Germany/ESA

Deep impact
Fortran is used in many areas of science, including astronomy, where it has been used to analyse distant galaxies imaged by the XMM-Newton mission.

In fact, there are many reasons why physicists still choose Fortran over other languages. Even though the first Fortran standard from 1966 bears little resemblance to the latest version from 2003, a very high degree of "backward compatibility" has been retained. Nearly all code written to the Fortran 77 standard therefore works perfectly well with current compilers, and code that is even older usually needs only minor changes. This has led to the accumulation of a huge volume of well-tested code that will be valuable for years to come, much of which is freely available in central databases called procedure libraries. Another advantage of modern Fortran over many of the alternative languages is that it is easier to write programs that are robust and bug-free. Fortran compilers nearly always include options to detect many common programming errors, whereas C compilers do not. It is also generally accepted that a high proportion of all the security vulnerabilities of the Internet arise from "buffer overflows" or "memory leaks" in programs written in C and it is much more difficult – although it is not impossible – to make corresponding mistakes in Fortran.

Despite the wealth of off-the-shelf software packages, there often comes a time in scientific research when they do not do quite what is needed. It is then usually much simpler for a researcher to write the necessary software than for a software expert to understand the scientific requirements. And, true to the design goals of John Backus and his team back in the mid-1950s, Fortran is still one of the easiest languages for a scientist or engineer to learn. ■

There are many reasons why physicists still choose Fortran over other languages