The concept of "range" used in experimental calculations

Yablokova L.V.

Samara State Aerospace University

Abstract. In article considered the concept of range, as main design tool and implementation of algorithms of processing of the big data sets used in experimental calculations of mathematical physics.

Keywords: the range concept, algorithms, first, last, interval function

Citation: Yablokova L.V. The concept of "range" used in experimental calculations. Proceedings of Information Technology and Nanotechnology (ITNT-2015), CEUR Workshop Proceedings, 2015; 1490: 402-405. DOI: 10.18287/1613-0073-2015-1490-402-405

Wise use of modern computer technology is unthinkable without the skillful use of the approximate and numerical analysis. This explains the extremely increased interest in methods for approximate computation and analysis of experimental data containing a big volume of information. Like no other branch of science, numerical methods, as the main tool of mathematical physics, is closely intertwined with many software applications, as a means or an object of research. At the same time the problems solved by modern science and modern technology, much more complex, and the way to solve them is not limited to general questions. Numerical methods and algorithms for processing big data sets, as a means to successfully meet the new challenges, are utilized in areas such as numerical solution of Maxwell's equations and d’Alambert's equation. As a rule, that to create complex mathematical computer model from the specialist require not only an impeccable mathematical background, but a basic knowledge of modern, high-performance algorithms, as well as effective methods for their use for big data. Currently, the bulk of the library for scientific calculations includes the implementation of the set of basic algorithms. However, manual implementation of simple variants of the basic algorithms it allows for better understanding and, consequently, more efficient using and configuring more advanced version of the library. Another important fact to re-implement the basic algorithms, is the fact that we are often faced with new computing environments with their new properties, which cannot be well involved in older implementations.

Realizing basis versions of algorithms, more fitted to specific objectives, and, not based on system subprograms, specialist’s researchers can achieve bigger portability and more long save relevance, applied in the course of the decision, algorithms and data structures. Besides, despite the enhancements which are built in program libraries for scientific computations, the mechanisms used to sharing of programs aren't always rather powerful that library functions could be adjusted easily for effective implementation within a specific objective. For example, the imperative programming paradigm with which use codes of the print_array programs (Fig. 1) and print_array_1 (Fig. 2) bringing array contents to the console provides to the developer of means of the description of sequence of commands and logical transitions which the computer is going to execute. It is very similar to the orders expressed by an imperative mood in natural languages, and the basic syntax concept supporting this paradigm is the operator. Within this it is quite enough simple, provided in examples, tasks, language tools, means supported by the selected coding style and the used idioms for implementation of the decision. But similar approach can be hardly conceptually widespread on more complex problems where the abstractions used in design process shall be already higher level.

```
program print_array
   implicit none
   integer, parameter :: n = 10
   integer :: i, arr(n) = (/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/)
   do i = 1, n
      print *, arr(i)
   end do
end program
```

Fig. 1. – The output to console with using a subscript operator

```
program print_array_1
   implicit none
   integer, target :: i, arr(10) = (/(i, i = 1,10)/)
   integer, pointer :: p_arr(:)
   p_arr => arr(:)
   print "(i4)", (p_arr(i), i = 1, size(p_arr))
end program
```

Fig. 2. – The output to console with using a pointer

Provision of the most effective implementation of the specific algorithm directed on operation with a big data set will allow to use it for the solution of complex challenges of computing character and repeatedly. In an example of print_array_2 (Fig. 3), procedural and operator programming style which is development of an imperative paradigm is applied and allows to use concept the range for implementation of subprograms.

We will assume that there is a sequence which elements shall be processed by any algorithm. In the course of such processing there is an appeal to all elements of sequence, beginning from the first and finishing last. The similar situation meets in implementation process of algorithms so often that for processed elements of sequence there is a special notation: the range. For example, the range [first, last) consists of all elements from first to last, but not including the last. Such asymmetric form of record is used to focus attention that [first, last) is a half-open interval which includes all elements, since first, but not last. The range [first, last) is admissible if to all its elements, excepting last, it is possible to get access and if the element facing last is achievable from first namely if sequentially being moved, since first, on all line items of the range a finite number of times it is possible to get to a line item the previous last. For example, the range [0, *N*) is valid. The empty range [0,0) also is valid, and here the range [*N*-1,0) is invalid as the element in a line item of *N*-1 appears after an element in a line item 0. Therefore it doesn't make sense to speak about elements in line items from *N*-1 to 0 as about tolerance range.

```
program print_array_2
   implicit none
   integer, parameter :: n = 100000
   integer :: arr(n)
   ! .....
   ! array initialization
   ! .....
   call print(loc(arr(1)), loc(arr(n)))
end program
subroutine print(first, last)
   integer :: first, last, val
   pointer(first, val)
   do while(first <= last)
      print *, val
      first = first + sizeof(integer)
   end do
end subroutine
```

Fig. 3. – The output to console with using the range concept

Generally, the ranges satisfy the following properties:

1. id range.
2. For any element in *i* line items, the range [*i, i*) is a valIf [*i*, *N*) is a valid range, then [*i*+1, *N*) is also a valid range.
3. If [*i*, *N*) is a valid range, and an element in a line item of *k* is achievable from an element in *i* line item, and the element in a line item of *N*-1 is achievable from an element in *k* line items, the ranges [*i, k*) and [*k*, *N*) are both valid ranges.
4. If both ranges [*i*, *k*) and [*k*, *N*) are both valid ranges, the range [*i*, *N*) is also a valid range.

From the fundamental point of view the ranges are defined thus because their asymmetric form helps to avoid errors, the so-called, lost unit, i.e. the quantity of elements in the range [*i*, *N*) is equal to *N*-*i*, namely it is exactly so much, how many it is expected.

Using the concept of the range it is possible to pass to other more general concept of interval function. The concept of interval function provides possibility of creation of algorithms capable to process any data structures of the elements supporting concept of one-dimensional sequence, for example one-dimensional arrays in the FORTRAN and similar languages. Besides, writing of a code of the program with use of interval functions requires smaller efforts, and decisions with their application usually look more visually and logically.

References

1. **Artemov IL.** Fortran: the basics of programming. M.: Dialog-MIFI, 2007.
2. **Golub J, Van Lone C.** Matrix calculations: Trans. from English. M.: Mir, 1999.
3. **Demmel J.** Computational linear algebra. Theory and Applications. Trans. from English. M.: Mir, 2001.
4. **Podbelsky VV, Fomin SS.** The course Programming in C: Proc. Allowance. M.: DMK Press, 2013.
5. **Vorotnikova DG, Golovashkin DL.** Long vectors algorithms for solving grid equations of explicit difference schemes. Computer Optics, 2015; 39(1): 87-93.
6. **Kazanskiy NL, Protsenko VI, Serafimovich PG.** Comparison of system performance for streaming data analysis in image processing tasks by sliding window. Computer Optics. 2014; 38(4): 804-810.