ABC: A double-conversion Compiler / Solver for Nanoscience Calculus

I. O. Kulik

Department of Physics, Bilkent University, Ankara 06533, Turkey
kulik@fen.bilkent.edu.tr

ABSTRACT

"CompLab" ("Compiler Laboratory") or "ABC" ("Advanced Basic-C") is a double conversion compiler working in two parallel regimes (A and B) one of which is similar to that of MATLAB while the other is a professional compilation routine similar to C and FORTRAN languages. Double-conversion scheme of compilation allows making advantage of both the simplicity of used code (program, commands) and at the same time of the power of C language in speed and in the capacity of CPU memory usage. The "A" regime allows for on-line calculation of multidimensional integrals, eigenvalues of large matrices, roots of nonlinear equations, plotting functions and their derivatives, etc. The "B" regime is an exportable program of conversion from the pseudobasic input code to C-code, compilation and solution. The C-code is fit for usage in any computational platform including Windows, OS/2, Linux and Unix machines.

Keywords: Compiler, C, Fortran, Windows, Linux

1 INTRODUCTION

The progress and demand in nanoscience and nanotechnology is ultimately related to the progress and perspectives of computational science. The latter develops by improving both the hardware (higher level of integration and increasing speed, as well as the consideration of the new types of computing, in particular quantum computation [1]) and the software (advanced mathematical programs and algorithms). The trend in the latter direction is in making programming simpler from the user side by considering higher-level compilers and solvers, and from the machine side by developing the more efficient computational codes of higher level of mathematical complexity. We address both these issues. The goal is achieved with a double level of connection between the programmer and the computer. We report on the development of a new computing device

Complab: ABC Compiler / Sollver

or shortly "ABC" with a meaning "Advanced Basic-C" compiler / solver. We use a double-conversion scheme with the input code formulated in simple terms of the pseudo quickbasic dialect which is further automatically translated to an appropriated C-code, and after that compiled to a stand-alone executable. This benefits from the algorithm simplicity, on one side, and from the code efficiency, on the other. The other advantage is that, by using the universality of C code to all computational platforms, we can export C-code by a demand (which is provided with the ABC, when requested) and compile it to a stand alone executable to any of the existing computational platforms (Windows, OS/2, Linux and Unix) with higher resources. The code is full professional and allows for a number of advanced routines including, in particular, the eigenvalue problem for large Hermitian matrices and for *extremely large* (up to dimension of 1.000.000 if executed on a standard Pentium PC) sparse matrices, multidimensional integration, arbitrary-precision floating-point arithmetics, special number types like e.g. the creation / annihilation second-quantized operators of many-particle quantum theory, etc. Below we illustrate several examples of the ABC programming, which is in fact supplemented by few service routines like curve plotting and multidimensional numeric integration with the highly user-friendly rules.

2 THE "A" REGIME

Illustration of this regime is presented with the integration command

```
int (x, 0, inf, exp(-x^2))
```

for calculation of an integral

$$\int_0^\infty dx \exp(-x^2)$$

and the command

int(x,0,inf,y,0,inf,exp(-x^2–y^2)*j0(c+x-y))[c=0 100 20][p]

for double integral

$$\int_0^\infty dx \int_0^\infty dy \exp(-x^2 - y^2) j_0(c + x - y)$$

where $j_0(x)$ is a Bessel function, to be plotted (see Fig.1) for c values from c=0 (100 points) to c=20. [p] is an option for plotting, and the [f] or [f=xxx] option can be added for saving the file of the data c, int(c) with the names 'fff' or 'xxx', respectively.

In similar way, computation of higher order (unrestricted) dimensional integrals may continue with extra variables introduced and limits of integration specified as arbitrary functions of arguments and parameters. The method of integration is a recursive Gaussian algorithm with 96 Gauss-Legendre points [2]. The accuracy of calculatuion is typically 10^{-12} (the relative error). Option is added to deal with fast oscillating functions, with a specific alrorithm invented to test and report on case when the high precision can not be achieved.

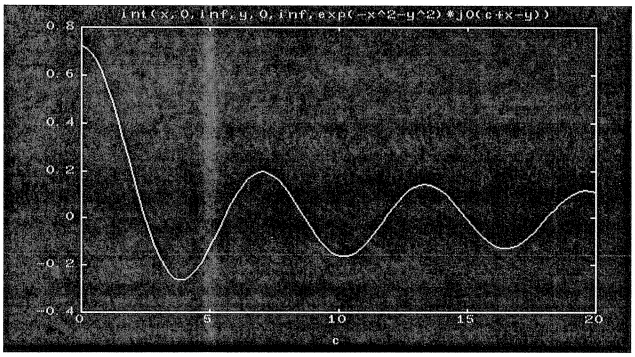

Fig.1. Screen output after clicking at the command of double integration

Functions and the expressions can be plotted as exemplified in Fig.2 by simultaneously plotting the dependences

$$\sin(x)/x; \exp(j_0(x));$$

and

$$(\exp(j_0(x)))''$$

(second derivative of $\exp(j_0(x))$ with respect to x). The ABC command for that operation is

plot sin(x)/x; exp(y0(x)); (exp(y0(x)))'' [x=0 100 20]

for plotting 100 points between x=0 and x=20 (Fig.2).

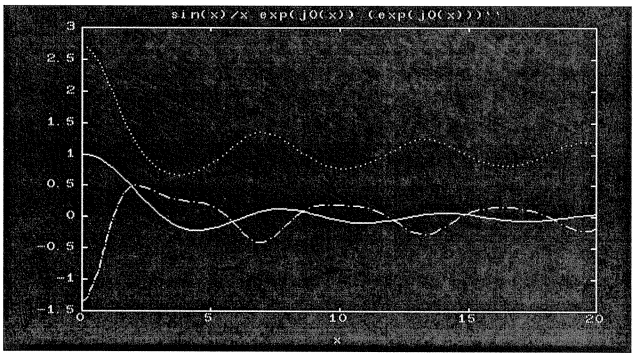

Fig.2. Screen output for plot of 3 functions

The execution of longer procedures can be performed by writing program on a file "xxx.abc" and executing it with a command

abc xxx

This is an adaptive regime of programming in which the commands "plot", "int", "eigval", "eigsparse", etc. are intermixed with the mathematical tasks ("solve", "find", "compare", etc.) and conditions ("where", "and", "therefore", ...) making the program similar to mathematical text from the paper or a textbook, with formulas and words explaining the purpose of calculation.

3 THE "B" REGIME

The B regime of ABC includes authomatic conversion from pseudobasic dialect (the B-code) to C language (the C-code) and is providing for full professional programming with novel routines required in nanoscience calculus (arbitrary precision arithmetics, very large sparse matrices, special numbers (like operators of quantum mechanics).

The mathematical numeric arsenal of ABC includes integrers, standard double-precision floating-points, arbitrariy precision floating point numbers, anticommuting Grassman variables and second-quantized quantum mechanical variables a, a^+ (a^+ is the creation and a

the annihilation operator). With those, the tight-binding Hamiltonian of system of electrons localized at atomic sites and allowed to hop between the sites due to a tunnel effect, e.g.

$$H = -t\sum_{k=1}^{N} a_k^+ a_{k+1} e^{i\alpha} + U\sum_{k=1}^{N} n_{k\uparrow} n_{k\downarrow} \qquad (1)$$

is presented in a computer code by an expression not much different from one written with formula (1). Provided the matrix $H(*,*)$ is introduced from (1), the program of eigenvalue calculation will continue as

```
DIM k, n, nn AS LONG
nn  = 100
DIM E(nn) AS DOUBLE, H(nn, nn) AS DOUBLE
E = eigval(H, nn)
OPEN "xxx" FOR OUTPUT AS #1
FOR n = 1 TO nn: PRINT #1, E(n): NEXT n
CLOSE #1
```

where "eigval" is a command for eigenvalues of matrix H of dimension nn, $E(*)$ is an array of eigenvalues, and the last 3 lines specify the command of writng the eigenvalues to the file 'xxx'.

Arbitrary precision floating point numbers, A@, are presented in a machine code as arrays

$$\text{A@}=(A_1, A_2, ..., A_m; A_{m+1}, ..., A_n; n)$$

where A@ are integers in a numeric system with radix R, and n is specifiedv in an ABC code with an instruction

'@NN

where NN specifies the presision. Choosing R as large as 2^{30}, and using the standard C routine of multiplication, addition, etc. of the numbers, will allow for quite fast alrorithm of numeric arithmetics with numbers of class A@.

Second quantized Fermi operators a in Eq.(1) are presented as 2 –dimensional matrices [3,4]

$$a_m = (v\otimes)^{m-1} a (\otimes u)^{N-m}, m = 1, ..., N \qquad (2)$$

where a, u, and v are 2 x 2 matrices

$$a = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, u = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, v = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad (3)$$

and $\otimes$ is a symbol of direct (Kronecker) product of matrices. The operators a, a^+ anticommute

$$[a_m, a_l^+] = \delta_{ml} \qquad (4)$$

which is specific to particles obeying the Fermi statistics (electrons). Matrices a, a^+ are very sparse, and this sparsity is automatically taken care of by the numeric routine of ABC representing equations (1)-(3) in C-code produced after translating the commands to C.

The sparse matrix code of ABC is employing the identity

$$\psi(t) = \exp(-Ht)\psi(0) \qquad (5)$$

where $\psi(0)$ is a state vector of dimension 2^N, and t is the imaginary time in Schroedinger evolution. Given the random initial value of $\psi(0)$ which can be expanded in eigenvectors ϕ_n of H, $\psi(0) = \sum_{n=1}^{N} C_n \phi_n$ we receive at time t

$$\psi(t) = \sum_{n=1}^{N} C_n e^{-E_n t} \phi_n \qquad (6)$$

such that at large t only the lowest eigenvalue $E(1) = \min E_n$ of H gives the dominant contribution to ψ. Then we receive this eigenvalue as

$$E(1) = <\psi^*(t), H\psi(t)> / <\psi(t), \psi(t)>$$

At the next step, ψ is generated randomly and by subtracting the properly weigheted contibution of $\phi^{(1)}$, placed to the subspace orthogonal to first vector $\phi^{(1)}$, and then transformed with (5) within this space. Second minimal eigenvalue then will be found, etc. This method works fast since numeric implementation of transformation (5) received by Taylor expansion of the exponent addresses only nonzero matrix elements of matrix H.

The double-conversion scheme of compilation in both regimes "A" and "B" is fast since the conversion, being long in its algorithmic implementation, nevertheless nowhere has long loops and is therefore non time consuming. This is demonstrated by fast (of order of fraction of second) execution of full root of ABC: conversion-compilation-solution. In the "B" regime, in fact, the compilation/conversion time doesn't matter at all,

since this regime is assumed to apply for large programs. The economy of speed follows from the possibility of exporting the C-code to faster machine.

REFERENCES

[1] I. O. Kulik, T. Hakioglu, and A. Barone, Quantum Computational Gates with Radiation Free Couplings. European Physical Journal B 30, 219, 2002.

[2] Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, M. Abramowitz and I. Stegun, eds. Dover Publ., New York, 1972.

[3] I. O. Kulik, Hopping and Correlation Effects in Atomic Clusters and Networks. Techn. Proc. of Internat. Conference on Computational Nanoscience, p. 196. M. Laudon and B. Romanowicz, eds. Computational Publ., Boston, 2001

[4] H. Boyaci, Z. Gedik, and I. O. Kulik, Superconductivity in Mesoscopic Clusters and Networks. Journ. of Superconductivity, 13, 1031, 2000.