functions satisfy a recurrence relation which is helpful to their evaluation. From Equation (5) we see that

$$F^+_{i+j}(x) = \frac{[F^+_i(x)\,F^+_j(x)+N]}{[F^+_i(x)+F^+_j(x)]},\ i,j \geqslant 1. \quad (12)$$

It is of interest to note that certain convergents to periodic continued fractions representing quadratic surds also satisfy this recurrence relation as shown by Frank;[7] such convergents, where the surd consists of a square root only, can be thought of as special cases of the estimates generated by $F^+_k(x)$. If we set $i = j$ we obtain from Equation (12) an expression of the same form as the Newton Raphson formula. Hence if k is a power of two, 2^l say, we can evaluate $F^+_k(x)$ by l applications of Newton's method with an efficiency of unity. If k is not a power of two the efficiency cannot be exactly unity because of the logarithm in its definition, and hence must be less than unity in this case.[6] Hence the greatest computational efficiency is obtained when k is a power of two, where the iterative process is equivalent to multiple applications of the Newton Raphson method.

When several processors are available the calculation of the polynomials may be done in parallel. Furthermore, if the processors are arranged in a tree-like structure the evaluation of each polynomial can be broken down into parts to be evaluated concurrently in a logarithmic cascade (Kogge).[8] We generalise the efficiency such that $\log_2(k)$ is divided by the number of steps involving division and multiplication (except by constants) where at each step several multiplications and/or divisions may be done concurrently; we do not consider the cost of the processors here. We have seen that we should at least choose k even for greatest efficiency. The number of steps is equal to the number required to evaluate one polynomial plus one for the ratio, which is the lowest integer upper bound of $1+\log_2(k-2)$ for k not two; the efficiency is maximised to unity only for $k = 4$. For $k = 2$ the efficiency is unity but extra processors are unnecessary. The use of several processors does not produce an efficiency greater than that of the sequential application of the Newton Raphson process; except for $k = 2$ and 4 even with parallel processing we cannot achieve the same efficiency. The multiple application of the Newton process cannot be accelerated in time by parallel processing because each application of it must await the completion of the previous one.

It was observed in Section 2 that we obtain an algorithm with a root-bracketing property by using the function $F^+_k(x)$ with k odd. In the light of the previous paragraph let us choose $k = 2^l+1$; we use l applications of the Newton Raphson method to obtain $F^+_{2^l}(x)$ and, noting that $F^+_1(x) = N/x$ from Equation (5), we have

$$F^+_k(x) = \frac{N[F^+_{2^l}(x)+x]}{[xF^+_{2^l}(x)+N]} \quad (13)$$

The efficiency falls from unity by around $2/l$ if l is not too small; in a parallel implementation it falls by around $1/l$ giving little advantage. The extra cost incurred to obtain root bracketing is not great.

We turn to the evaluation of the function $F^-_k(x)$. For even k it is best evaluated as N divided by $F^+_k(x)$, this single extra operation making its use slightly less efficient. Sequential Newton Raphson, l times, followed by division yields an efficiency around $1-1/l$. For odd k the function is the product of x and a ratio of polynomials of degree $(k-1)/2$ in x^2; the multiplier x can be absorbed in the denominator polynomial. The efficiency is not greatly enhanced by parallel computation. In general the function $F^-_k(x)$ is less useful than $F^+_k(x)$.

5. Conclusion

Iterative processes based on formula (5) have kth order convergence to $\sqrt{N}$ for any positive starting value and to $-\sqrt{N}$ for any negative starting value. The convergence is slow at first if the magnitude of the starting value is very small or very large. The processes are stable. No significant gain is obtained by the use of parallel processors (even when measured by latency alone). The most efficient formula is equivalent to multiple applications of the Newton Raphson method. It can be modified simply at little extra computational cost to give root bracketing (Equation (13)).

The formulae (5) include a number of recently published iterative expressions.[2,3,9] The above remarks apply to all of them. Those of Morrison and Moler[2] and Dubrulle[3] are equivalent to $F^-_k(x)$. The formula I derived differently[9] is equivalent to $F^-_k(x)$ for k odd and $F^+_k(x)$ for k even; I withdraw the suggestion[9] that parallel computation of it would be beneficial!

M. J. JAMIESON
Department of Computing Science,
University of Glasgow,
Glasgow G12 8QQ.

References

1. J. Bentley, Programming pearls – birth of a cruncher. *Communications of the Association for Computing Machinery* **29** (1986), 1155–1161.
2. C. Moler and D. Morrison, Replacing square roots by Pythagorean sums. *IBM Journal of Research and Development* **27** (1983), 577–581.
3. A. A. Dubrulle, A class of numerical methods for the computation of Pythagorean sums. *IBM Journal of Research and Development* **27** (1983), 582–589.
4. J. F. Traub, Computational complexity of iterative processes. *SIAM Journal on Computing* **1** (1972), 167–179.
5. M. S. Paterson and L. J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing* **2** (1973), 60–66.
6. H. T. Kung, The computational complexity of algebraic numbers. *SIAM Journal on Numerical Analysis* **12** (1975), 89–96.
7. E. Frank, On continued fractions and binomial quadratic surds. *Numerische Mathematik* **4** (1962), 85–95.
8. P. M. Kogge, *The Architecture of Pipelined Computers*, McGraw-Hill, Washington, New York, London (1981).
9. M. J. Jamieson, A note on the convergence of an iterative scheme for solving a quadratic equation. *The Computer Journal* **30** (1987), 189–190.

Announcement

10–14 July 1989

ECOOP '89

European Conference on Object-Oriented Programming
East Midlands Conference Centre, University of Nottingham

ECOOP '89 will be the third annual European Conference on Object-Oriented Programming. The conference will present the latest research in the object-oriented paradigm, and will provide a focus for discussion and exploration of practical applications of object-oriented systems. There will be one-day tutorials and three days of presentations of invited and refereed papers, with associated workshops and panel sessions. One half-day session will focus on papers describing object-oriented research from Esprit projects.

The topics to be addressed at ECOOP '89 include but are not limited to: Languages, Theory, Implementation, Applications, Databases, User interfaces, Design, Tools and environments, Concurrency and Teaching.

Other activities

Proposals are invited for workshops, panels, demonstrations, videos or other activities covering issues relevant to the object-oriented paradigm. Send a one- or two-page proposal, including the organiser's name, affiliation, address and phone number to the Programme Chairman before 27 February 1989.

Call for Exhibitors

If you or your company is interested in reserving exhibition space at ECOOP '89, please advise Julia Allen.

Conference venue

The East Midlands Conference Centre is situated on the campus of Nottingham University, two miles away from the city centre and its InterCity rail network. The M1 motorway is five minutes drive away, while the East Midlands International Airport can be reached in 20 minutes. The high-speed InterCity 125 train takes about $1\frac{1}{2}$ hours to London.

For accommodation, there are comfortable study bedrooms available within the University campus. Alternatively Nottingham has numerous 3- and 4-star hotels.

Programme Chairman
Steve Cook, *Queen Mary College*

General Enquiries
Julia Allen, *British Informatics Society Ltd, 13 Mansfield Street, London W1M 0BP, UK*