Optimal Asynchronous Newton Method for the Solution of Nonlinear Equations

A. BOJAŃCZYK

University of Warsaw, Warsaw, Poland

Abstract. A modification of Newton's method for the solution of the equation $F(x) = 0$ on a multiprocessor computer is studied. A class of asynchronous Newton methods is introduced and an optimal method in this class, as well as its optimal parallel implementation, is shown. Then the optimal asynchronous parallel method is compared with the optimal asynchronous sequential method. It turns out that no matter how many processes are used, a Newton process (in the class of asynchronous Newton methods) can be speeded up by at most a factor of 4.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*parallel algorithms*; G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*iterative methods*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Newton method, asynchronous algorithms

1. *Introduction*

We consider the numerical solution of the equation

$$F(x) = 0,$$

where F is a nonlinear function, F: $D \subset R^N \to R^N$. We assume that there exists a simple zero x^* of F; that is, $F(x^*) = 0$ and $F'(x^*)^{-1}$ exists.

We solve a nonlinear equation $F(x) = 0$ by iteration, constructing the sequence $\{x_i\}$, which tends to zero x^*, $\lim x_i = x^*$. Let x_0 be a sufficiently close approximation to x^*. To get the next approximation, we need some information about F and its derivatives. If these pieces of information are independent, we can evaluate them in parallel. In this paper we deal with a modification of Newton's method for the solution of a nonlinear equation $F(x) = 0$ on a parallel machine.

The paper is organized as follows. A definition of a class of the asynchronous Newton method (ANM) is given in Section 2. An optimal method in this class, as well as its optimal implementation, is shown in Section 3. In Section 4 the optimal asynchronous parallel method and the optimal asynchronous sequential method are compared. Throughout the paper the phrases *uniprocess*, *sequential process*, and 1-*process* are used interchangeably.

Author's present address: CMA, The Australian National University, G.P.O. Box 4, Canberra, Australia, ACT 2600.

2. *Asynchronous Newton Method*

As an illustration assume that we deal with the scalar case and we have two processes available, one for the evaluation of F and the other for the evaluation of F'. At each step of the Newton iteration, $F(x_i)$ and $F'(x_i)$ are evaluated in parallel. After both evaluations are completed, the next approximation x_{i+1} is computed from the following relations:

$$F'(x_i)\Delta x_i = -F(x_i), \qquad x_{i+1} = x_i + \Delta x_i.$$

This parallel two-process algorithm generates exactly the same sequence of iterates as the sequential algorithm. The only difference between the sequential and parallel versions is in the cost of single iteration. For the two-process algorithm the cost is $\max[c(F), c(F')]$ as opposed to $c(F) + c(F')$ in the uniprocess case, where $c(F)$ and $c(F')$ denote the cost of one function and one derivative evaluation, respectively. If $c(F') = pc(F)$, $p \geq 1$, we achieve speedup of the factor S_2:

$$S_2 = \frac{1+p}{p}.$$

For large p, the process assigned to evaluate F is idle most of the time, waiting for the other process to finish the evaluation of F'.

Now consider a multidimensional case, $N \geq 2$. When F is a vector function of N components, then the standard Newton's method requires $N + N^2$ evaluations of component functions, N evaluations for the function F, and N^2 evaluations for the derivative F'. Assume that the cost of evaluation dominates the solution of a linear system. This will be true if N is not too large. If we split k processes, $k \geq 2$, among the $N + N^2$ evaluations and if the cost of all evaluations is not the same, the speedup is $S_k = k$. If the cost of the evaluations is not the same, our problem cannot be decomposed into independent tasks of the same complexity. As in the scalar case, some processes have to wait at each iteration for the other processes to finish their part of the job. Thus, such a k-process algorithm may also waste much of its potential power. This drawback is due to synchronization of all the processes involved in the computations. Synchronization, in turn, guarantees that the parallel algorithm generates exactly the same iterates as the sequential algorithm. By removing the synchronization restrictions and letting the processes continue their evaluations according to the information currently available, we obtain an asynchronous algorithm [3]. This can be done in several ways. As an example consider the following modification of the two-process algorithm.

Let A, B, and C be global variables containing current values of $F(x)$, $F'(x)$, and x. The first process updates A and C, while the second process updates B. The processes are controlled by the following program [3].

Given starting values: $\{F(x_0), F'(x_0), x_1 = x_0 - [F'(x_0)]^{-1}F(x_0)\}$.

process P_1:

```
begin
  while —CON do
    begin
      A := F(C);
      C := C − B⁻¹*A
    end
end;
```

process P_2:

```
begin
  while —CON do
    B := F'(C)
end;
```

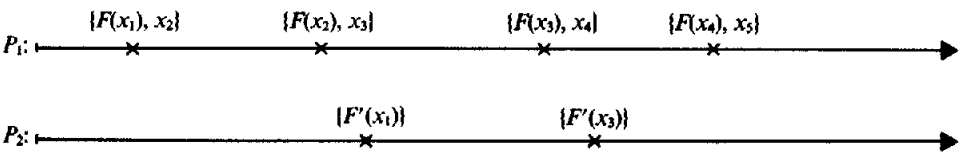

FIGURE 1

where CON denotes some termination criterion. In this algorithm the processes are never idle; as soon as a process finishes updating a global variable, it starts the next evaluation by using the current values of needed variables. If $c(F) < c(F')$, then the first process performs several Newton iterations based on the fixed value of the derivative and changing the value of the function. This algorithm can be illustrated as shown in Figure 1 where the x's on each line denote the time instants corresponding to the termination of evaluation of the current values of global variables $\{A, C\}$ and $\{B\}$, respectively.

In general the program defines the sequence of iterates $\{x_i\}$ through the formula

$$x_{i+1} = x_i - [F'(x_j)]^{-1}F(x_i), \qquad j \le i.$$

This motivates the generalization of Newton's method to the class of asynchronous Newton methods (ANMs).

Definition. An iterative method ϕ belongs to the class of ANMs iff ϕ generates the sequence $\{x_i\}$ of approximations of a zero x^* of a function F as follows:

Given starting values: $\{F(x_0), F'(x_0), x_1 = x_0 - [F'(x_0)]^{-1}F(x_0)\}$.

$$x_{i+1} = x_{i-k(i)} - [F'(x_{i-m(i)})]^{-1}F(x_{i-k(i)}) \quad \text{for} \quad i \ge 2, \tag{2.1}$$

where $\{k(i)\}$ and $\{m(i)\}$ are sequences of integers such that

$$\{i - k(i)\} \quad \text{and} \quad \{i - m(i)\} \quad \text{are nondecreasing,} \tag{2.2}$$

$$1 \le k(i) \le m(i) \le i, \tag{2.3}$$

$$\lim_{i \to \infty}(i - k(i)) = +\infty. \tag{2.4}$$

Note that in formula (2.1) the value of F is taken at point $x_{i-k(i)}$; the same iterate appears on the right-hand side of this formula. Such a restriction is not crucial. Roughly speaking, the convergence rate depends on the smaller of two indices, the index of the iterate on the right-hand side of formula (2.1), and the index of the iterate at which the value of function F is given. This explanation relates also to assumption (2.3). Finally, assumption (2.4) guarantees that eventually more and more recent values of iterates will be used in the evaluation of F and that early iterates will no longer appear.

Remark. We treat the evaluation of F or F', the information used by ANM, as indivisable operations in the sense that, when F is a vector function, we evaluate the components of F or F' at the same point. A different approach is presented by Baudet [1] who considers the possibility of evaluating each component f_i, $i = 1, \ldots, N$, of a vector function F at different points.

We now formulate the conditions under which ANM converges to the solution of a nonlinear equation $F(x) = 0$.

Let $e_i = \|x_i - x^*\|$ and $J = \{x : \|x - x^*\| \le \Gamma\}$. Assume $F'(x)$ exists and is Lipschitz continuous for all $x \in J$. Define

$$A_2 = A_2(\Gamma) = \sup_{x,y \in J} \frac{\|[F'(x^*)]^{-1}(F'(x) - F'(y))\|}{2\|x - y\|}.$$

Let q be any number such that $0 < q < 1$.

THEOREM 2.1. *If*

(*i*) $A_2 \le q/(3 + 2q)$,
(*ii*) $x_0 \in J$,

then

(*iii*) *every ANM is well defined,*
(*iv*) $\lim x_i = x^*$, $e_i \le q e_{i-k(i)}$,
(*v*) $e_i \le C_i e_{i-k(i)} e_{i-m(i)}$, *where*

$$C_i = \frac{3A_2}{1 - 2A_2 e_{i-m(i)}}.$$

PROOF. The proof is based on the proof of Theorem 2.1 in Traub and Woźniakowski [7] and therefore is omitted. □

Remark. For the standard Newton method, we have $k(i) = m(i) = 1$. In this case, the condition of the theorem can be relaxed by assuming that [7]

$$A_2 \le \frac{q}{1 + 2q}.$$

Note that Newton's method requires the solution of a linear system with the matrix $F'(x)$. We shall assume that the evaluation of $F'(x)$ is performed in such a way that we obtain triangular decomposition of $F'(x)$; that is, $F'(x) = L(x)U(x)$, where $L(x)$ and $U(x)$ are lower and upper triangular matrices, respectively. By $c(F')$, we denote the cost of evaluation of $F'(x)$ in a decomposed form, that is, the cost of evaluation of $L(x)$ and $U(x)$ given x and F'. Let $c(F)$ denote the cost of evaluation of $F(x)$ and d be the combinatory cost of determination of the next iterate given current values of $F'(x) = L(x)U(x)$, $F(x)$, and x.

We define quantities $C(F^{(j)})$ as

$$C(F^{(j)}) = c(F^{(j)}) + d, \qquad j = 0, 1.$$

From now on we assume that

$$C(F') = \frac{n_1}{n_0} C(F) \tag{2.5}$$

for some integers $n_1 \ge n_0 \ge 1$, which are relatively prime. □

Remark. The cost of evaluating F and F' and the combinatory cost d differs depending on whether the computation is performed sequentially or in parallel. In the case of sequential computation we can assume that each arithmetic or logic operation costs unity. Then $c(F)$ denotes the total number of operations needed to evaluate F. Similarly, $c(F')$ denotes the total number of operations needed to evaluate F' plus the cost of decomposing the matrix $F'(x)$, $F'(x) = L(x) \cdot U(x)$.

Then by d we can mean the cost of solving the linear system

$$L(x)\cdot U(x)\,\Delta x = F(x).$$

Thus $d = \theta(N^2)$.

In the case of parallel computation with k processes, the cost of evaluating F and F' will usually be smaller than in the sequential case and will depend on how well the evaluation of F and F' can be parallelized. The cost of solving the linear system with k processes is $d = \theta(N^2/k)$, $1 \le k \le N$, or $d = \theta(\log^2 N)$ for $k = N^3$. □

In both sequential and parallel computation $C(F)$ and $C(F')$ are integer multiples of a basic unit of time. By multiplying this basic unit of time by the greatest common divisor of $C(F)$ and $C(F')$, we get a new unit of time for which eq. (2.5) is satisfied, with n_0, n_1 relatively prime.

3. *Optimal Method in the Class ANM*

In a real multiprocessor environment the execution time of a given process, for various reasons [3], is usually not a constant and may vary substantially in successive executions. In particular, the time needed to evaluate $F(x)$ and $F'(x)$ can vary in an unpredictable way. This makes the analysis of an asynchronous algorithm very difficult. In order to derive properties of the sequence of iterates, we shall assume the following model of an abstract parallel machine:

(a) Any number of processes can cooperate simultaneously.
(b) No time is needed for a process to obtain any piece of information or to communicate with other process.

Condition (a) ensures the possibility of creating any number of simultaneous independent computations. Condition (b) justifies our assumption (2.5) that the costs of function and derivative evaluations, $C(F)$ and $C(F')$, are related to each other through the relation

$$C(F') = \frac{n_1}{n_0} C(F),$$

where $n_1 \ge n_0 \ge 1$ are relatively prime integers.

For a given function F satisfying the assumptions of Theorem 2.1, we are interested in finding the best (in a sense to be specified later) method belonging to ANM class.

Let $t_\phi(i)$ be the evaluation time (cost) of x_i for a method ϕ from the ANM class. The evaluation of x_i requires multiple evaluations of F and F', and the solution of linear systems. Thus there exist nonnegative integers α_i and β_i, depending on method ϕ, such that

$$t_\phi(i) = \alpha_i C(F) + \beta_i C(F').$$

From Theorem 2.1 it follows that any method $\phi \in$ ANM generates the sequence $\{x_i\}$ such that the corresponding sequence of errors e_i, $e_i = \| x_i - x^* \|$ satisfies the inequality

$$e_i \le C_i e_{i-k(i)} e_{i-m(i)}.$$

Instead of the sequence of errors, we shall consider a majorant sequence $\{f_i\}$, $f_i \ge e_i$, defined by

$$f_0 = e_0, \qquad f_1 = e_1, \qquad f_i = C_i f_{i-k(i)} f_{i-m(i)}.$$

We call $\{f_i\}$ a majorant sequence of errors.

Let $\text{comp}(\epsilon, \phi)$, $\epsilon > 0$, denote the minimal cost (time) of the evaluation of an iterate $x_{k(\phi)}$ where $k(\phi)$ is the smallest index such that

$$f_{k(\phi)} \leq \epsilon f_0.$$

We are interested in finding a method ϕ, $\phi \in \text{ANM}$, which minimizes the complexity $\text{comp}(\epsilon, \phi)$.

Let

$$\text{comp}(\epsilon, \text{ANM}) = \min_{\phi \in \text{ANM}} \text{comp}(\epsilon, \phi)$$

be the ϵ-complexity of the $\text{ANM} = \text{ANM}(F)$ class, where F is a given function whose zero is being sought. We shall show that the method ψ, which evaluates F and F' at every computed iterate and calculates a new iterate based on the last known values of F and F', minimizes complexity $\text{comp}(\epsilon, \psi) = \text{comp}(\epsilon, \text{ANM})$. The following is an algorithm of method ψ:

(i) Every working process computes F and F' alternatingly.
(ii) For every iterate x_i, the evaluations of $F(x_i)$ and $F'(x_i)$ start simultaneously and are carried on by two processes; that is, one process works on $F(x_i)$, the other on $F'(x_i)$.
(iii) For every pair of nonnegative integers α, β, at time $t = \alpha C(F) + \beta C(F')$ a new iterate is evaluated on the basis of the last known values of F and F'.

We illustrate method ψ when $C(F') = \frac{3}{2}C(F)$; that is, the cost of the function evaluation takes two units of time. Assume that five processes are available. Figure 2 illustrates the execution of the algorithm with five processes.

Recall that starting values $F'(x_0)$ and $x_1 = x_0 - [F'(x_0)]^{-1}F(x_0)$ are known. At time $t = 0$, processes P_1 and P_4 start to evaluate $F(x_1)$ and $F'(x_1) = L(x_1)U(x_1)$, respectively. When the value of $F(x_1)$ is known, process P_1 computes $x_2 = x_1 - [F'(x_0)]^{-1}F(x_1)$ and starts to evaluate $F'(x_2)$. Simultaneously, process P_2 is activated and starts to evaluate $F(x_2)$. Similarly, when the value of $F'(x_1)$ is known (it is known at time $t = 3$), process P_4 computes $x_3 = x_1 - [F'(x_1)]^{-1}F(x_1)$ and starts to work on $F(x_3)$. Simultaneously, process P_5 is activated and starts to evaluate $F'(x_3)$.

At time $t = 5$ the values of $F'(x_2)$ and $F(x_3)$ are known. Thus the next approximation $x_5 = x_3 - [F'(x_2)]^{-1}F(x_3)$ is the same for both processes P_1 and P_4. Since the values of $F(x_5)$ and $F'(x_5)$ can be evaluated by P_1 and P_4, we do not need more processes to be activated. The same is true for $x_6, x_7, \ldots$, and, in general, we get the following relation between successive iterates:

$$x_{i+1} = x_{i-1} - [F'(x_{i-2})]^{-1}F(x_{i-1}) \qquad \text{for} \quad i \geq 4.$$

In the general case when $C(F') = (n_1/n_0)C(F)$, we need $n_1 + n_0$ processes to perform method ψ. At time $t = 0$, processes P_1 and P_{n_1+1} start to evaluate $F(x_1)$ and $F'(x_1) = L(x_1)U(x_1)$. When the value of the function is known, process P_i, $1 \leq i \leq n_1 - 1$, supplies the new approximation and starts to evaluate the value of the first derivative. Simultaneously, process P_{i+1} is activated and starts to evaluate the value of the function. Similarly, when the value of the derivative is known, process P_j, $n_1 < j \leq n_0 + n_1 - 1$, supplies the new approximation and starts to evaluate the value of the function. Simultaneously, process P_{j+1} is activated and starts to evaluate the value of the first derivative. At time $t = n_0 n_1$ (i.e., after n_1 function evaluations and n_0 first-derivative evaluations), the new iterate is the same for processes P_{n_1} and $P_{n_0+n_1}$. Since the functions and derivatives can be evaluated by P_{n_1} and $P_{n_0+n_1}$, respectively, we need not activate more processes. (See Figure 3.)

We need the following lemma.

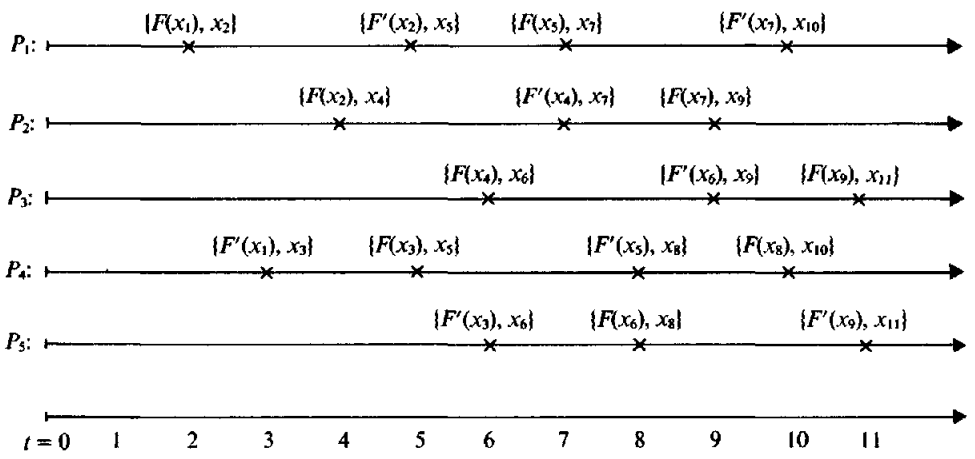

FIGURE 2

LEMMA. *If n_0 and n_1 are relatively prime numbers, then for every nonnegative integer n there exist nonnegative integers α_n and β_n such that*

$$\alpha_n n_0 + \beta_n n_1 = n_0 n_1 + n.$$

PROOF. The lemma immediately follows from the Euclidean algorithm. □

The lemma states that if one function evaluation takes n_0 units of time and one derivative evaluation takes n_1 units of time, then in every unit of time $t = n_0 n_1, n_0 n_1 + 1, \ldots,$ one of the iterates x_i is evaluated; that is, $t(i+1) = t(i) + 1$ for $i \geq i_0$ where $t(i_0) = n_0 n_1$. Therefore, the generated sequence satisfies the formula

$$x_i = x_{i-n_0} - [F'(x_{i-n_1})]^{-1} F(x_{i-n_0}), \qquad i \geq n_0 n_1.$$

We are now in a position to prove the optimality of method ψ.

THEOREM 3.1. *Method ψ is optimal; that is,*

$$\min_{\phi \in ANM} comp(\epsilon, \phi) = comp(\epsilon, \psi).$$

PROOF. Let $N(n_0, n_1) = \{k \in N : k = \alpha n_0 + \beta n_1, \alpha, \beta \in N\}$. Let $k_i \in N(n_0, n_1)$, $k_i < k_{i+1}$, and $\{k_1, k_2, \ldots\} = N(n_0, n_1)$. Define $T: \{k_i\} \to N$ as $T(k_i) = i$. Choose any method ϕ from the ANM class. The method ϕ generates the sequence of approximations $\{x_i\}$ and the corresponding majorant sequence of error $\{f_i\}$. Let $\{x_i^*\}$ and $\{f_i^*\}$ denote the sequence of approximations and the majorant sequence of the error generated by method ψ. Assume that f_k is known at time $t(i)$. From the definition of ψ it follows that at time $t(i)$ the approximation $x^*_{T(k(i))}$ is also computed. Assume by induction that

$$\forall j \leq k-1, \quad f^*_{T(k(j))} \leq f_j.$$

For $j = 1$ this is, of course, true. For $j = k$,

$$f_k = \frac{3A_2}{1 - 2A_2 f_{k-m(k)}} f_{k-k(k)} f_{k-m(k)},$$

$$f^*_{T(t(k))} = \frac{3A_2}{1 - 2A_2 f^*_{T(t(k)-n_1)}} f^*_{T(t(k)-n_0)} f^*_{T(t(i)-n_1)}.$$

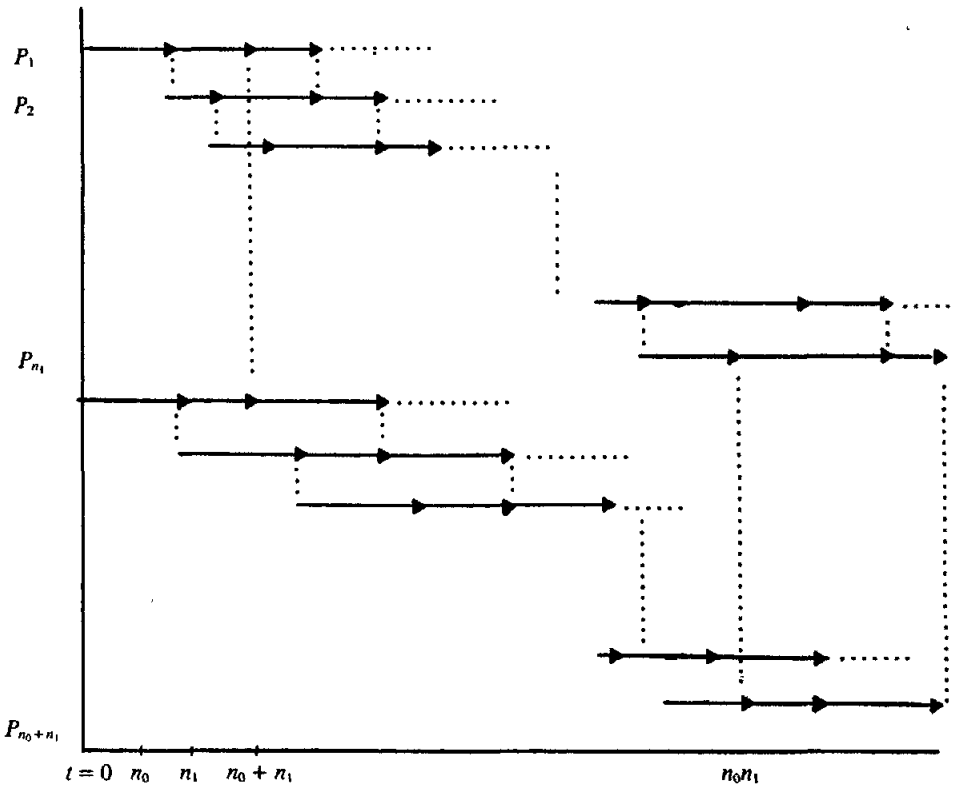

FIGURE 3

Since $t(k) - n_0 \geq t[k - k(k)]$, $t(k) - n_1 \geq t[k - m(k)]$, and the sequence $\{f_j^*\}$ is nondecreasing, $f^*_{T[u(k)]} \leq f_k$. This completes the proof. □

Remark. Note that in order to achieve the fastest convergence, the method generates and makes use of all possible information available for ANM class methods. Moreover, all the processes created during the computation are kept busy evaluating the most recent information.

4. *Comparison with Sequential Method*

We are interested in how much one can gain by applying the ANM class optimal parallel method instead of its optimal sequential method. To investigate this, we first compare optimal method ψ with a certain two-process method ϕ belonging to the ANM class and show that this particular method is, at most, twice as bad as ψ.

Our comparisons are based on the behavior of the majorant sequences of errors. Let $\{f_i\}$ and $\{g_i\}$ denote the majorant sequences of errors associated with methods ϕ and ψ, respectively. We say that method ϕ is superior to method ψ if, for $f_0 = g_0$ and any ϵ, $\epsilon > 0$, the time needed to find f_u such that $f_u \leq \epsilon \cdot f_0$ is smaller than the time needed to find g_L such that $g_L \leq \epsilon \cdot g_0$. This criterion is equivalent to comparing the complexity index of two iterations, where the complexity index is the ratio of the cost of one iteration to the logarithm of the convergence rate of the iteration [6].

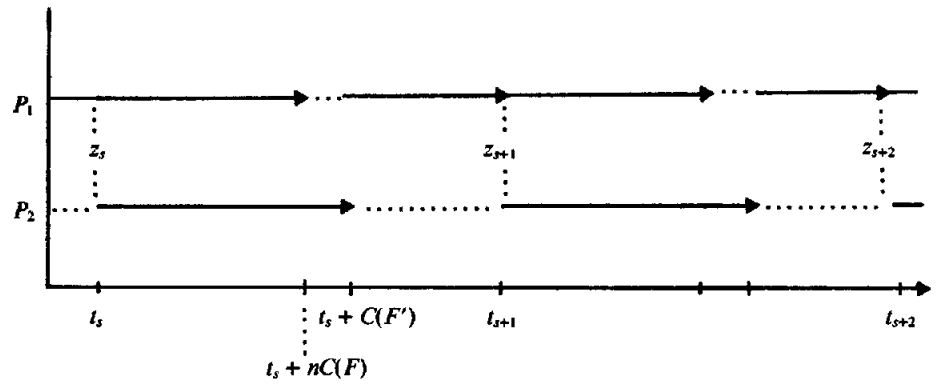

FIGURE 4

Let $n = [n_1/n_0]$ where $[x]$ denotes integer part of a nonnegative number x. For a nonnegative integer k, we define the two-process method $\phi = \phi(n, k)$, $\phi \in \text{ANM}$, as follows.

Denote two processes as P_1 and P_2, respectively. Process P_1 evaluates the function F and computes the sequence of approximations $\{x_i\}$, whereas process P_2 evaluates only the derivative F'.

In the starting phase process P_1 computes iterates $x_2, x_3, \ldots, x_{k+1}$ on the basis of the fixed value of derivative $F'(x_0)$ and changes the value of the function; that is,

$$x_{i+1} = x_i - [F'(x_0)]^{-1}F(x_i), \qquad i = 1, \ldots, k.$$

(Recall that from the definition of the ANM class, $F'(x_0)$ and $x_1 = x_0 - [F'(x_0)]^{-1}F(x_0)$ are given.) For $k = 0$, there is no starting phase.

Let us introduce the following notation.

$$\begin{aligned} z_{-1} &= x_0, \\ z_i &= x_{i(n+k)+(k+1)} && \text{for } i \geq 0, \\ y_{i,j} &= x_{i(n+k)+(k+1)+j}, && j = 0, \ldots, n+k-1, \\ t_i &= kC(F) + i[C(F') + kC(F)]. \end{aligned}$$

Assume that at time t_s, $s \geq 0$, iterate z_s and derivative $F'(z_{s-1})$ are known. The next iterate z_{s+1} is computed at time t_{s+1} by the following procedure (see Figure 4): At time t_s, both processes P_1 and P_2 are activated. Process P_1 executes $n-1$ Newton iterations on the basis of the fixed value of derivative $F'(z_{s-1})$ and updates the value of the function; that is,

$$\begin{aligned} y_{s,0} &= z_s, \\ y_{s,j} &= y_{s,j-1} - [F'(z_{s-1})]^{-1}F(y_{s,j-1}), \qquad j = 1, \ldots, n-1. \end{aligned}$$

Next, process P_1 evaluates $F(y_{s,n-1})$. Simultaneously, $F'(z_s)$ is being computed by process P_2. As $nC(F) \leq C(F')$, the completion of function evaluation $F(y_{s,n-1})$ precedes the completion of derivative evaluation $F'(z_s)$. Thus, process P_1 may have to wait until process P_2 finishes its evaluation. When $F'(z_s)$ is known, process P_1 computes the new approximation

$$y_{s,n} = y_{s,n-1} - [F'(z_s)]^{-1}F(y_{s,n-1}).$$

Note that $y_{s,n}$ is known at time $t_s + C(F')$. Now, process P_2 becomes idle while process P_1 executes another k Newton iteration with the fixed derivative $F'(z_s)$ and

updates the value of the function; that is,

$$y_{s,j} = y_{s,j-1} - [F'(z_s)]^{-1}F(y_{s,j-1}), \qquad j = n+1, \ldots, n+k.$$

As the next approximation z_{s+1}, we take $y_{s,n+k}$, which is computed at time t_{s+1}. This completes the description of method $\phi = \phi(n, k)$.

To investigate the convergence rate of the sequence $\{x_i\}$ generated by method $\phi(n, k)$, we introduce a normalized majorant sequence of the error. Recall that the majorant sequence of the error $\{f_i\}$ satisfies relations

$$\begin{aligned} f_0 &= e_0 = \| x_0 - x^* \|, \\ f_1 &= C_0 e_0^2, \\ f_{i+1} &= C_0 f_i f_{j(i)}, \end{aligned} \tag{4.1}$$

in which the sequence of indices $j(i)$ is fully determined by method $\phi(n, k)$. If we define

$$E_i = C_0 f_{i(n+k)+(k+1)},$$

then E_i satisfies

$$\begin{aligned} E_0 &= (C_0 f_0)^{k+2}, \\ E_1 &= E_0^{[n+(k+3)(k+1)]/(k+2)}, \\ E_{i+2} &= E_{i+2}^{k+2} E_i^{n-1} \qquad \text{for} \quad i > 0. \end{aligned} \tag{4.2}$$

We call a sequence $\{E_i\}$ a normalized majorant sequence of the error. (Note that the sequence $\{E_i\}$ corresponds to the sequence $\{z_i\}$.) Set $p_i = \log_{E_0} E_i$. Then from (4.2) we obtain

$$\begin{aligned} p_0 &= 1, \\ p_1 &= \frac{n + (k+3)(k+1)}{(k+2)}, \\ p_{i+2} &= (k+2)p_{i+1} + (n-1)p_i \qquad \text{for} \quad i > 0. \end{aligned} \tag{4.3}$$

This difference equation has the solution

$$p_i = c_i \left[\frac{(k+2) + \sqrt{(k+2)^2 + 4(n-1)}}{2} \right]^i, \qquad c_i \to c. \tag{4.4}$$

Thus we have

$$E_i = (E_0)^{p_i}. \tag{4.5}$$

For a given $\epsilon > 0$ let $M = M(\epsilon, \phi(n, k))$ denote the smallest index such that

$$E_M \le \epsilon C_0 f_0. \tag{4.6}$$

From (4.4) and (4.5) we get the following bounds on M,

$$\begin{aligned} \frac{\log \log 1/\epsilon}{\log \frac{1}{2}[(k+2) + \sqrt{(k+2)^2 + 4(n-1)}]} - 1 \le M \\ \le \frac{\log \log 1/\epsilon}{\log \frac{1}{2}[(k+2) + \sqrt{(k+2)^2 + 4(n-1)}]}. \end{aligned} \tag{4.7}$$

Method $\psi(n, k)$ depends on parameter k. We are interested in a choice of k for which the convergence is as fast as possible.

Let $t_\psi(i) = t[i, \psi(n, k)]$ denote the evaluation cost (time) of the ith iterate generated by method $\psi(n, k)$. Then the best method (with respect to k) is method

$\psi(n, k^*)$, where k^* is determined by relation

$$t\{M[\epsilon, \phi(n, k^*)], \phi(n, k^*)\} = \min_{k\geq 0} t\{M[\epsilon, \phi(n, k)], \phi(n, k)\}. \tag{4.8}$$

The transition cost from E_i to E_{i+1} for a method $\phi(n, k)$ is $n_1 + kn_0$ units of time, where $n_1 = C(F')$ and $n_0 = C(F)$. From this, (4.6), and (4.7), it follows that k^* also minimizes the quantity

$$\frac{n_1 + kn_0}{\log \frac{1}{2}[(k+2) + \sqrt{(k+2)^2 + 4(n-1)}]}. \tag{4.9}$$

Note that $k^* = k^*(n_0, n_1)$.

Below we compare the optimal parallel method $\psi = \psi(n_0, n)$ with the optimal two-process method $\phi = \phi(n, k^*)$.

Analogously with method $\phi(n, k)$ (see (4.2)), we define a normalized majorant sequence of the error $\{F_i\}$ corresponding to the optimal parallel method $\psi = \psi(n_0, n_1)$ through relations

$$F_0 = C_0 e_0, \qquad F_1 = F_0^2, \qquad F_i = F_{i-k(i)} F_{i-m(i)} \quad \text{for} \quad i \geq 2, \tag{4.10}$$

where $k(i)$ and $m(i)$ are defined by method $\psi(n_0, n_1)$. Setting $\hat{p}_i = \log_{F_0} F_i$ we get

$$\hat{p}_0 = 1, \qquad \hat{p}_1 = 2, \qquad \hat{p}_i = \hat{p}_{i-k(i)} \hat{p}_{i-m(i)},$$

and it is known that $F_i = F_0^{\hat{p}_i}$.

Let $\hat{M} = \hat{M}[\epsilon, \psi(n_0, n_1)]$ be the smallest index such that

$$F_{\hat{M}} \leq \epsilon F_0 = \epsilon C_0 f_0, \tag{4.11}$$

(compare to (4.6)) and let $t_\psi(i) = t[i, \psi(n_0, n_1)]$ be the evaluation time (cost) of the ith iterate generated by the method ψ. The following theorem holds.

THEOREM 4.1. *For a given function F satisfying the assumptions of Theorem 2.1 positive integers n_0, n_1, n such that $n_0 = C(F)$, $n_1 = C(F')$, $n = [n_1/n_0]$, we have*

$$\frac{1}{2} \leq \frac{t_\psi(M)}{t_\phi(M)} \leq 1;$$

that is, in the ANM class the optimal two-process method $\phi = \phi(n, k^*)$ takes at most twice as long as the optimal parallel method $\psi = \psi(n_0, n_1)$.

PROOF. For the proof of Theorem 4.1, see [2]. □

We now describe an optimal sequential method from the ANM class.

For $k \geq 1$ let ϕ_k be the sequential method from the ANM class for which x_{i+1} is generated from x_i in the following way,

$$y_{i,0} = x_i,$$
$$y_{i,j} = y_{i,j-1} - [F'(x_i)]^{-1} F(y_{i,j-1}), \qquad j = 1, \ldots, k.$$

It is easy to show that when conditions of the convergence Theorem 2.1 are satisfied, then method ϕ_k yields the sequence $\{x_i\}$, which converges to a zero of F with order $k+1$ [4, 5]. Each outer iteration costs $kC(F) + C(F')$. Thus, the optimal method from the $\{\phi_1, \phi_2, \ldots\} \subset$ ANM class is $\phi_{\bar{k}}$ where $\bar{k}$ minimizes the value of

$$\frac{kC(F) + C(F')}{\log(k+1)}, \tag{4.12}$$

which is the complexity index of method ϕ_k [6]. From (4.9) and (4.12) we get the following costs of obtaining an ϵ-approximation of a zero of F for the optimal sequential method $\phi_{\bar{k}}$ and the optimal two-process method $\phi = \phi(n, k^*)$, respectively:

$$t_{\phi_{\bar{k}}}(\bar{M}) = \log \log \frac{1}{\epsilon} \frac{\bar{k}n_0 + n}{\log(1 + \bar{k})},$$

$$t_{\phi}(M) = \log \log \frac{1}{\epsilon} \frac{k^* n_0 + n_1}{\log \frac{1}{2}[(k^* + 2) + \sqrt{(k^* + 2)^2 + 4(n - 1)}]}.$$

Hence the cost ratio is bounded by

$$1 \leq \frac{t_{\phi_{\bar{k}}}(\bar{M})}{t_{\phi}(M)} \leq 2.$$

THEOREM 4.2. *The ANM optimal parallel method is at most four times better than the ANM optimal sequential method.*

From Theorem 4.2 it follows that no matter how many processes are used we can speed up Newton iteration in the ANM class *at most* by a factor of 4.

REFERENCES

1. BAUDET, G. M. Asynchronous iterative methods for multiprocessors. *J. ACM* 25, 2 (Apr. 1978), 226–244.
2. BOJAŃCZYK, A. Solving systems of algebraic equations in different models of computation. Ph.D thesis, Institute of Informatics, Univ. Warsaw, Warsaw, Poland, June 1981, (in Polish).
3. KUNG, H. T. Synchronized and asynchronous parallel algorithms for multiprocessors. In *New Directions and Recent Results in Algorithms and Complexity*, J. F. Traub, ed. Academic Press, New York, 1976.
4. SHAMANSKII, V. E. A modification of Newton's method. *Ukr. Mat. Zh. 19* (Sept. 1967), 210–220.
5. TRAUB, J. F. *Iterative Methods for the Solution of Equations*. Prentice-Hall, Englewood Cliffs, N.J., 1964.
6. TRAUB, J. F., AND WOŹNIAKOWSKI, H. Strict lower and upper bounds on iterative computational complexity. In *New Directions and Results in Algorithms and Complexity*, J. F. Traub, ed. Academic Press, New York, 1976, pp. 15–34.
7. TRAUB, J. F., AND WOŹNIAKOWSKI, H. Optimal radius of convergence of interpolatory iterations for operator equations. *Aequationes Math. 21*, 159–172.

RECEIVED JULY 1982; REVISED MARCH 1984; ACCEPTED APRIL 1984