An Approach to Parallelization of Krawczyk’s Method*

Takashi HisAKADO† and Kohshi OKUMURA†

This paper describes a parallel algorithm of Krawczyk’s method that lends itself most naturally to the original sequential algorithm extended by R.E. Moore and S.T. Jones. It also presents an improvement to achieve efficient computation in parallel. The effectiveness is confirmed by an example.

1. Introduction

Finding all solutions of nonlinear equations is the significant part of nonlinear circuits analysis and in the past it received the attention of many researchers [6-10]. This subject is now being continued to interest researchers[11-14]. The most powerful method is considered to be the interval method developed first by R.Krawczyk, being extended by R.E.Moore and S.T.Jones[1-5]. It is based on the interval operation, called KMJ method, and it can find all the solutions in a given region. However, KMJ method cannot practically be applied to the nonlinear equations when they have many variables, since it requires tremendously fast computing as well as vast amounts of memory. To overcome the defects, the algorithm of KMJ method has been so far reformed from the standpoint of sequential computation[15-17]. However, the restriction of KMJ method is presumably provided as far as we are interested in sequential algorithm.

In this paper, in order to improve this situation we first present the parallel KMJ algorithm that has not been attained so far in the research field of interval analysis. Parallelization is carried out in MIMD computer. The algorithm is parallelized naturally through the sequential algorithm of the original KMJ algorithm. Second, to make the parallel computation more effective, we describe an improvement to avoid over-concentration of communication between processors. Finally, we confirm the effectiveness of the parallel implementation of KMJ method.

2. KMJ Algorithm in Sequential Computation

We try to find all real solutions in a given closed region \( X_{\text{initial}} \in I(\mathbb{R}^n) \) to a nonlinear system of equations:

\[
\mathbf{f}(\mathbf{x}) = \mathbf{0},
\]

where real vector-valued function \( \mathbf{f}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n \) is continuously differentiable and \( I(\mathbb{R}^n) \) denotes the set of all intervals. Other definitions about interval are shown in Appendix 1. We briefly state the outline of KMJ algorithm which is based on the Moore test shown in Appendix 2. The search procedure of all solutions consists of a recursive application of the test, beginning with the region \( X = X_{\text{initial}} \). At each level, we test the region \( X \). If region \( X \) satisfies condition 1 or 2, we delete this region from the list \( T \) which is the list of subregions of \( X_{\text{initial}} \) yet to be tested. If region \( X \) satisfies condition 3, we apply the Newton method to eq.(1) and find a solution. If the region satisfies neither 1, 2, nor 3, then we bisect \( X \) and select one of the two resulting half-regions for analysis at the next level. We save the other half-region in the list \( T \). If the list is implemented with a stack or queue, the search procedure is depth-first search or breadth-first search of a binary tree, respectively. The search procedure is described in detail below.

[Sequential KMJ Algorithm]
S1: Set $X$ to $X_{\text{initial}}$. Set list $T$ to be empty.
S2: Test the region $X$ by computing $F(X)$ and $K(X)$.
   (i) If there are no solutions in $X$, i.e., $F(X)$ $\neq$ 0 or $K(X)$ $\cap$ $X$ = $\phi$, then go to S4.
   (ii) If there exists a unique solution, i.e., $K(X)$ $\subseteq$ $X$ and $\|I - YF'(X)\|$ < 1, then we apply Newton method to eq.(1). That is, the following iteration
   
   
   $$x^{(k+1)} = x^{(k)} - Yf(x^{(k)}), \quad k = 0, 1, \cdots$$ (2)
   
   
   from the initial value $x^{(0)}$ $\in$ $X$ converges to a solution. Find a solution and go to S4.
   (iii) If neither of the above two conditions is satisfied, then go to S3.
S3: Bisect the region $X$ according to rules described below, i.e. $X = X - X$. Add the region $X$ to list $T$ and set $X$ to region in list $T$ and go to S2.
S4: Test list $T$.
   (i) If list $T$ is empty, then terminate.
   (ii) If list $T$ is not empty, then set $X$ to region in list $T$, delete this region from list $T$ and go to S2.

The flow chart of the algorithm is shown in Fig. 1.

Fig. 1 The flow chart of the procedure for finding all solutions by Krawczyk, Moore and Jones

We bisect a region $X$ in the coordinate direction which maximizes the width $w(X_i)$, i.e.

$$X = X - X = (X_1, \cdots, X_{k-1}, X_k, X_{k+1}, \cdots, X_n), \quad X_k = [x_k, m(X_k)]$$

3. Parallelization of the KMJ Algorithm

We parallelize the procedure of sequential KMJ algorithm on message passing parallel computer systems. The memory is local to a processor and messages must be exchanged between the local memory of the other processors. Efficient communication is very important to this message passing implementation. We show the procedure implemented by a general master-slave algorithm.

Let us consider $N$ processing elements denoted by $PE_i$ ($i = 0, \cdots, N-1$). These are separated to one master and $N-1$ slave;

Master: $PE_0$

Slave: $PE_i, \quad i \in S, \quad S = \{1, 2, \cdots, N-1\}$ (3)

The master process is responsible for coordinating the work of the others and the slave processes do not communicate with one another. That is, the communication is limited to

$$C_{0\rightarrow i}[\text{message}] \text{ or } C_{i\rightarrow 0}[\text{message}], \quad i \in S \quad (4)$$

where the communication from $PE_i$ to $PE_j$ is denoted by $C_{i\rightarrow j}[\text{message}]$ and message denotes the contents of communication, i.e., a region “$X$”, a flag “Terminate” or “Request”.

The master $PE_0$ has the list $T$ and requests the slaves to test the region. When one task by a slave is completed, the master sends other region to the slave for the next task. Once all tasks have been handed out, termination messages are sent instead. The procedure of slaves are shown below.

[Algorithm for slaves $PE_i, \quad i \in S$]

S1: Get the region $X_i$ from master $PE_0$ $C_{0\rightarrow i}[X_i]$.
S2: Test the region $X_i$ by computing $F(X_i)$ and $K(X_i)$.
   (i) If there are no solutions in $X_i$, then go to S4.
   (ii) If there exists unique solution in $X_i$, then find a solution and go to S4.
   (iii) If neither of the above two conditions is satisfied, then go to S3.
S3: Bisect the region $X_i = X_i - X_i$. Send the region $X_i$ back to the master $PE_0$ $C_{i\rightarrow 0}[X_i]$ and set $X_i$ to $X_i$; go to S2.
S4: Request the next region from the master $C_{i\rightarrow 0}[$Request$]$ and go to S5.
S5: Wait for another task.
   (i) If the master sends a new region $C_{0\rightarrow i}[X]$ or $C_{0\rightarrow i}[X_j]$, then set $X_i$ to the new region and go to S2.
   (ii) If the master sends a termination message $C_{0\rightarrow i}[$Terminate$]$, then the slave terminates the process.

On the other hand, the procedure of the master $PE_0$ is described in detail below. List $W$ is the list of waiting slave $PE_i$. The number of element in the list $W$ is denoted by $|W|$.

[Algorithm for master $PE_0$]

S01: Set the region $X$ to $X_{\text{initial}}$ and send the region $X$ to the slave $PE_i$ $C_{0\rightarrow i}[X]$. Set list $T$ to empty and add remaining slaves $PE_i$ ($i = 2, \cdots, N-1$) to list $W$. 


S02: Wait for the messages from slaves.
(i) If the slave PE$_i$ $i \in S$ sends the region $X_i$ ($C_{i \rightarrow 0}[X_i]$), then go to S03.
(ii) If the slave PE$_i$ $i \in S$ requests a new region ($C_{i \rightarrow 0}[\text{Request}]$), then go to S04.

S03: Test list $W$.
(i) If $W$ is not empty, then send $X_i$ to a slave PE$_j$ in list $W$ ($C_{0 \rightarrow j}[X_i]$) and delete PE$_j$ from list $W$.
(ii) If list $W$ is empty, add the region $X_i$ to list $T$ and go to S02.

S04: Test list $T$.
(i) If list $T$ is not empty, then set $X$ to region in list $T$, delete this region from list $T$ and send the region to PE$_i$ ($C_{0 \rightarrow i}[X]$); go to S02.
(ii) If list $T$ is empty, then go to S05.

S05: Test list $W$ for termination.
(i) If $|W| < N - 2$, then add PE$_i$ to list $W$ and go to S02.
(ii) If $|W| = N - 2$, then send termination messages to all slaves ($C_{0 \rightarrow i}[\text{Terminate}]$, $i \in S$), and terminate.

The flow chart of the parallel KMJ algorithm is shown in Fig. 2. First, the master send the given region to PE$_1$ and go to the state “Wait”. On the other hand the slaves starts from the state “Start”. In this case, the message of instruction is $C_{i \rightarrow 0}[\text{Request}]$ or $C_{0 \rightarrow i}[\text{Terminate}]$ which are implemented by a communication of an integer flag. The messages of region are $C_{0 \rightarrow i}[X_i]$, $C_{i \rightarrow 0}[X_i]$ and $C_{0 \rightarrow j}[X_i]$ which consist of $2n$ real numbers.

4. Performance Results

We apply the parallel KMJ algorithm to a simultaneous algebraic equation with eight unknowns shown in Appendix 3. The procedure is implemented on a distributed memory parallel processor system HITACHI SR2201 which has a 3-dimensional crossbar network and whose transmission rate between two processors is 300 mega bytes/s. The performance of one processing element is 0.3 GFLOPS.

In order to test the effect of communication, we compare the computing time of the parallel KMJ algorithm with two processing elements to that of the sequential KMJ algorithm. In this parallel implementation, one of the two processing elements is master and the other is slave. The performance results are illustrated in Table 1. This result shows that the effect of communication which is defined by

$$1 - \frac{\text{Computing time with 1 PE (not parallelized)}}{\text{Computing time with 2 PEs (parallelized)}}$$

is about 1.8%.

<table>
<thead>
<tr>
<th></th>
<th>Time[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1PE (not parallelized)</td>
<td>$2.13 \times 10^5$</td>
</tr>
<tr>
<td>2PEs (parallelized)</td>
<td>$2.17 \times 10^2$</td>
</tr>
</tbody>
</table>

The efficiency of parallel computation is shown in Fig. 3 from which we can see that increasing the number of processors leads to decrease in computing time. The speedup[18] defined by

$$\text{Speedup} = \frac{\text{Computing time with 1 PE}}{\text{Computing time with i PEs}}, \quad i = 4, 8, 16, 32, 64$$

shows the efficiency with respect to the case of two processing elements and is shown in Table 2. In general, because one of the processing elements is used
for master, the theoretically optimal speedup in the parallelized algorithm is proportional to the number of slaves

\[(\text{number of processing elements}) - 1.\]  

In the case of 4, 8 and 16 PEs, Table 2 shows that the performance achieves the theoretically optimal speedup. However, the overconcentration of communication to master make the performance worse when the number of PEs is 64.

5. Improvement

In order to improve the performance, it is crucial to optimize the communication and idle time. In general, saving of communication time causes the increase of idle time. On the other hand, the decrease of idle time increases message passing. We try to saving the communication time by distributing the control of the list \(T\) which is the list of regions yet to be tested.

In this case we test the region with depth-first search. We give a number \(B\) which denotes the maximum number of successive bisection. Each slave \(\text{PE}_i, i \in S\) has the bisected regions in each list \(T_i\) which is implemented by a stack. A slave \(\text{PE}_i\) set the region received from master to list \(T_i\) and test the regions based on the list \(T_i\). The slave \(\text{PE}_i\) sends the regions at head of list \(T_i\) to the master when the region is consecutively bisected for \(B\) times. Fig. 4 shows an example of bisected regions tested by a slave \(\text{PE}_i\) when \(B = 4\). The regions \(T_i[2], \ldots, T_i[5]\) and \(X'_i\) is send to the master. Because the successive bisection of a region indicates that the neighborhood of this region has to be bisected many times in the same way, the slave requests to test the neighborhood by other slaves.

The previous parallel KMJ algorithm corresponds to the case of \(B = 0\). If we set \(B\) large number, the communication decreases. It causes, however, the unbalance of the task between the slaves as well as the increase of idle time. Optimizing the trade-off of communication and idle time is crucial. The improved procedure of slaves is shown below.

[ Improved algorithm for slaves \(\text{PE}_i, i \in S\) ]

\(S'_1\): Get the region \(X_i\) and the number \(B\) from master \(\text{PE}_0\) \((C_{0 \to i}[X_i])\).

\(S'_2\): Test the region \(X_i\) by computing \(F(X_i)\) and \(K(X_i)\).

(i) If there are no solutions in \(X_i\), then set the bisection number \(b = 0\); go to \(S'_4\).

(ii) If there exists unique solution in \(X_i\), then find a solution and set \(b = 0\); go to \(S'_4\).

(iii) If neither of the above two conditions is satisfied, then set \(b\) to \(b + 1\); go to \(S'_3\).

\(S'_3\) : Bisect the region \(X_i = X_i \cup X'_i\) and set \(X_i\) to \(X_i\).

(i) If \(b = B + 1\), then send \(B + 1\) regions \(X_i\) which consists of the \(B\) regions at the head of list \(T_i\) and \(X'_i\) to master \((C_{i \to 0}[X_i])\), and delete the \(B\) region from list \(T_i\); go to \(S'_2\).

(ii) Otherwise add \(X'_i\) to list \(T_i\) and go to \(S'_2\).

\(S'_4\) : Test list \(T_i\)

(i) If \(T_i\) is not empty, then set \(X_i\) to region at head of list \(T_i\) and delete this region from list \(T_i\). Go to \(S'_2\).

(ii) If \(T_i\) is empty, request the next region from

![Fig. 3 The efficiency of parallel computation](image_url)

![Fig. 4 The bisected regions tested by a slave PE_i when B=4.](image_url)

Table 2  Efficiency of parallel computation

<table>
<thead>
<tr>
<th>PE</th>
<th>Time[s]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.17×10^5</td>
<td>–</td>
</tr>
<tr>
<td>4</td>
<td>7.19×10^4</td>
<td>3.0</td>
</tr>
<tr>
<td>8</td>
<td>3.10×10^4</td>
<td>7.0</td>
</tr>
<tr>
<td>16</td>
<td>1.45×10^4</td>
<td>15.0</td>
</tr>
<tr>
<td>32</td>
<td>7.05×10^3</td>
<td>30.5</td>
</tr>
<tr>
<td>64</td>
<td>3.62×10^3</td>
<td>59.9</td>
</tr>
</tbody>
</table>
the master \((C_{i-0}[\text{Request}])\) and go to \(S'_5\).

\(S'_5\): Wait for another task.

(i) If the master sends a new region \((C_{0\rightarrow i}[X])\) or \(C_{0\rightarrow i}[-X]\), then set \(X\) to the new region and go to \(S'_2\).

(ii) If the master sends a termination message \((C_{0\rightarrow i}[\text{Terminate}])\), then terminate.

On the other hand, the procedure \(S_0\) of master is changed as below.

[Improved algorithm for master PE\(_0\)]

\(S'_0\): Test of list \(W\).

(i) If \(|W| \geq B + 1\), then send the \(B + 1\) regions of \(X\) to \(B + 1\) PEs in list \(W\) \((C_{0\rightarrow i}[X])\).

(ii) If \(|W| < B + 1\), then send \(|W|\) regions in the received \(B + 1\) regions \(X\) to each processing element in list \(W\) \((C_{0\rightarrow i}[X])\) and other \(B + 1 - |W|\) regions are added to the head of list \(T\); go to \(S'_2\).

In this case, the message of region \(C_{0\rightarrow i}[X]\) and \(C_{0\rightarrow i}[-X]\) from master to slave is one region which consists of \(2n\) real numbers. The message of region \(C_{i-0}[-X]\) from slave to master is \(B + 1\) regions which consists of \((2B + 1)n\) real numbers.

The performance of the improved algorithm for 64 PEs are shown in Table 3. In the case of \(B = 2\) and 4, the performance is better than 63, which means the overconcentration of communication is eliminated. In addition, the case \(B = 8\), the performance is worse than the case \(B = 4\). This indicates that large \(B\) lead to the increase of idle time. This improvement is efficient for larger parallel processings in which more overconcentration of communication will occur.

<table>
<thead>
<tr>
<th>(B)</th>
<th>Time[s]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.62\times10^3</td>
<td>59.9</td>
</tr>
<tr>
<td>2</td>
<td>3.43\times10^3</td>
<td>63.3</td>
</tr>
<tr>
<td>4</td>
<td>3.41\times10^3</td>
<td>63.6</td>
</tr>
<tr>
<td>8</td>
<td>3.52\times10^3</td>
<td>61.5</td>
</tr>
</tbody>
</table>

6. Concluding Remarks

We have presented the parallel KMJ algorithm and have confirmed its effectiveness compared with the sequential one. Furthermore, by saving the communication time between processing elements and idle time, we realize an efficient implement in message passing parallel computer. In addition, we suggest a method of optimizing the idle and communication time and realize the further decrease of communication by distributing master work and confirm the effect.

References


Appendix

Appendix 1. Definition of interval computation

Let \( \mathbb{R} \) be the set of all real numbers. The interval number is defined by

\[ A = [a, b] = \{ a \in \mathbb{R} | a \leq x \leq b \} \] (A1)
For an interval $A$, we use the following notation defined by

$$m(A) \triangleq \frac{1}{2}(a + b)$$
$$w(A) \triangleq b - a$$
$$|A| \triangleq \max\{ |a|, |b| \}.$$ 

Denote by $I(\mathbb{R}^n)$ the set of $n$-dimensional rectangles. An element of $I(\mathbb{R}^n)$ can be represented by an interval vector

$$X \triangleq (X_1, X_2, \ldots, X_n)^t \quad (A2)$$

where $X_1, X_2, \ldots, X_n \in I(\mathbb{R})$. The midpoint of vector $X$ is defined by

$$m(X) \triangleq (m(X_1), m(X_2), \ldots, m(X_n))^t.$$ 

An interval $n \times n$ matrix $A$ is defined by

$$A \triangleq (A_{ij}), \quad A_{ij} \in I(\mathbb{R})$$
$$i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, m.$$ 

We denote the set of interval matrices by $I(\mathbb{R}^{n \times m})$. The norm of $A \in I(\mathbb{R}^{n \times m})$ is defined by

$$\|A\| = \max \sum_{j=1}^m |A_{ij}|. \quad (A3)$$

**Appendix 2. Moore test**

The inclusion monotonic interval extension[4] of $f(x)$ is denoted by

$$F(X) \triangleq (F_1(X), \ldots, F_n(X))^t, \quad (A4)$$

where the symbol $^t$ denotes the transposition. A natural interval extension of the real function is derived by replacing the real variables by corresponding interval variables and replacing the real arithmetic operations and functions by the corresponding interval arithmetic operations and functions. The Jacobian matrix of the system (1) and its interval extension are denoted by

$$f'(x) \triangleq (f'_{ij}(x)) \quad \text{and} \quad F'(X) \triangleq (F'_{ij}(X))$$

respectively.

The Krawczyk operator of eq.(1) is defined by

$$K(X) \equiv y - Y f(y) + [I - Y F'(X)]^{-1} (X - y)$$

where $I$ is the $n \times n$ identity matrix.

It has been shown to have the following conditions about the existence and nonexistence of solutions in a region $X$.

1. If $F'(X) \not= 0$, then there are no solutions in $X$.
2. If $K(X) \cap X = \phi$, then there are no solutions in $X$, where $\phi$ is empty set.
3. If $K(X) \subseteq X$ and $\|I - Y F'(X)\| < 1$, then there is a unique solution to Eq.(1) in $X$.

**Appendix 3. Circuit equation**

We show a circuit equation of 1/3-subharmonic oscillation in three-phase circuit[14]. This is a simultaneous algebraic equation with eight unknowns, i.e., $a_0, \ldots, a_7$.

$$-\frac{9}{8} \xi a^2 a_1 - \frac{9}{4} \xi a_2 a_1 - 9 \xi a_2 a_1 + \frac{9}{8} \xi a_3 a_1 - \frac{9}{2} \xi a_5 a_1$$
$$-\frac{9}{4} \xi a_2 a_3 - \frac{9}{4} \xi a_3 a_2 - \frac{9}{4} \xi a_2 a_1 - \frac{9}{8} \xi a_3 a_5$$
$$-\frac{9}{8} \xi a_1 a_2 - \frac{1}{2} a_0 - \frac{9}{8} a_2 a_3 + \frac{9}{2} a_2 a_3 + 8 a_3 a_5$$
$$-2 a_2 a_3 + 2 a_2 a_2 + a_2 a_5 + 2 a_4 + 2 a_3 a_5 - a_3 a_4 + a_2 a_3 + 8 a_3 a_5 + 4 a_5 a_3 = 0.$$
We set the parameter $\xi = 0.122, \eta = 0.28, \phi_0 = -0.133, E_m = 0.141$. In order to investigate the performance of the parallel algorithm, we search a solution $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (-0.160786, -0.014021, -0.142453, 0.191192, -0.099277, -0.117876, 0.028300, -0.002895).