Solving Mathematical Problems by Parallel Processors

Dr. M Raja Sekar
Department of Computer Science & Engineering
VNR VJIET, Hyderabad, Telangana,
rajasekarm9[at]gmail[dot]com

Abstract — the use of parallel processors to solve mathematical and numerical problems is growing rapidly from the past few decades. In this study, we designed a methodology that solves many mathematical and numerical problems with high computational speed. In many real time or real life applications very often we need to perform a lot of numerical computations, which may range from matrix multiplication, polynomial interpolation, solving polynomial equations, solving system of equations and so on. We consider the use of parallel processors to solve some numerical problems, namely Lagrange interpolation and polynomial root finding. The experiments were conducted on the financial dataset obtained from publicly available AWS database. We tested proposed algorithms on two different models of parallel processors and achieve the performance of $O(\log n)$.

Keyword: — Time complexity, ring, Lagrange interpolation, Durand-Kerner methods and mesh of trees.

I. INTRODUCTION

Polynomial interpolation is heavily used in geological mapping, cardiology, petroleum exploration etc. Similarly in digital signal processing, automatic control etc. Often we require fast extraction of all the roots of a high degree polynomial equation[1,2,3 and 4]. Solution to such problems are usually based on some standard numerical methods. However, the computations involved in these techniques are usually performed through some sequential steps and hence are very slow and cannot fulfill the demand of the real time or real life applications. Therefore, the sequential algorithms of these computations are usually not encouraged in solving the problems in the above applications. Efficient parallelization of various steps in these computations is thus called for so as to reduce the total execution time for effectively using them in the above applications. Design of efficient parallel algorithms for such problems has thus become one of the research areas in the field of parallel processing. In this study we present two parallel algorithms for the solution of two numerical problems namely polynomial interpolation and solving polynomial equation. For the polynomial interpolation, we choose Lagrange interpolation and we use Durand-Kerner and Ehrlich methods for solving polynomial equation. The algorithms are run on two different models of parallel computers, namely mesh of trees and ring [5, 6 and 7]. The rest of the paper is organized as follows. We describe the parallel algorithm in section 2. The parallel algorithm of the Durand-Kerner method is presented in section 3 followed by a conclusion in section 4.

II. LAGRANGES INTERPOLATION

The italic n-point Lagrange interpolation formula is follows [13]:

$$P(x) = \prod(x) \sum_{i=0}^{n-1} \frac{y_i}{(x - x_i)\prod(x)}$$

$$y_i = f(x_i)$$

$$(x) = (x_0, x_1, x_2, ..., x_n)$$

$$(y) = (y_0, y_1, y_2, ..., y_n)$$

In recent years, many parallel algorithms have been reported for polynomial interpolation in the literature [8, 9, 10, 11, 12, 13 and 14]. Capello, Gallopolous and Koc [15] presented a systolic algorithm that uses 2n-1 steps on n/2 processors in which each step required two substractions and one division. In [16] another systalic algorithm has been described with $O(n)$ time complexity. A parallel algorithm for rational interpolation has been reported in [17] with the time complexity of $O(n)$ on n+1 processors. A parallel algorithm for Lagrange interpolation has been described in [18]. It is shown that the algorithms runs in $O(\log n)$ time using $n^2$ processors. Ben Goertzel [19] presented a parallel algorithm for the evaluation of n-point Lagrange interpolating polynomial that use n/2+ $O(\log n)$ steps.
on an n processor tree with additional ring connections. The parallel algorithm described in [6] has the time complexity of \(O(\log n^3)\) on an EREW-PRAM model. For the parallel implementation, we choose the model of computation as described below.

2.1 Mesh of Trees (Computational Model)
We use an nxn mesh of trees (MOT) [8] consisting of \(n^2\) processors organized as follows:

1. Processors are placed in an nxn lattice with n rows and n columns. The processors placed in the \(\alpha\)th row and the \(\beta\)th column is denoted by \(P(\alpha, \beta)\).

2. In the \(\alpha\)th row, 1\(<\alpha\<n, the processor \(P(\alpha, \beta)\) is directly connected to the processors \(P(\alpha, 2\beta)\) and the \(P(\alpha, 2\beta+1)\) if they exist for \(\beta = 1\ to \ n/2\). That is there exist a binary tree connection in every row \(\alpha, 1<\alpha<n\ rooted\ at\ P(1, \beta)\).

3. Every processor has three logical registers, namely \(B(\alpha, \beta), A(\alpha, \beta)\) and \(C(\alpha, \beta)\) in the order.

III. FIRST PROPOSAL ALGORITHM

Step 1. For all \(\alpha=1\ to \ n\) do in parallel

1.1 \(A(\alpha, 1)=x;\)

1.2 Broadcast the content of \(A(\alpha, 1)\) to \(A(\alpha, \beta)\), 1\(<\beta\<n\ using\ binary\ tree\ connection.\)

1.3 \(C(\alpha, \alpha) = A(\alpha, \alpha).\)

Step 2. For all \(\alpha=1\ to \ n\) do in parallel

2.1 \(A(\alpha, 1)=x_{\alpha, 1} .\)

Broadcast the content of \(A(\alpha, 1)\) to \(A(\alpha, \beta), 1<\beta<n\ using\ binary\ tree\ connection\ along\ the\ \alpha\)th row.

2.2 \(B(1, \beta) = x_{\beta, 1} .\)

Broadcast the content of \(B(1, \beta)\) to \(B(\alpha, \beta), 1<\alpha<n\ using\ binary\ tree\ connection\ along\ the\ \beta\)th column.

2.3 \(A(\alpha, \alpha) = C(\alpha, \alpha) \)

Step 3. For all \(\alpha\ and \ \beta = 1\ to \ n\) do in parallel

3.1 \(A(\alpha, \beta) = A(\alpha, \beta) - B(\alpha, \beta)\)

3.2 \(B(\alpha, \alpha) = A(\alpha, \alpha)\)

The contents of the registers after this step as shown in table 1.

Table 1. Simulation after step 3.

<table>
<thead>
<tr>
<th>(x_0)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_0)</td>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
</tr>
<tr>
<td>(x_1)</td>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
</tr>
<tr>
<td>(x_0)</td>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
</tr>
<tr>
<td>(x_0)</td>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
</tr>
<tr>
<td>(x_0)</td>
<td>(x_1)</td>
<td>(x_2)</td>
<td>(x_3)</td>
</tr>
</tbody>
</table>

Step 4. For all \(\alpha=1\ to \ n\) do in parallel

4.1. Using the tree connection, compute the product of the contents of \(A(\alpha, \beta), 1<\beta<n\ along\ the\ \alpha\)th row and store it in \(A(\alpha, 1)\).

Step 5. For all \(\beta=1\ to \ n\) do in parallel Move the contents of \(B(\beta, \beta)\) to \(B(1, \beta)\).

Step 6. Using tree connection along the first row, compute the product of the contents of \(B(1, \beta), 1<\beta<n\ and\ store in\ \ B(1, 1)\).

Step 7. \(C(1, 1) = B(1, 1)\).

Step 8. For all \(\alpha=1\ to \ n\) do in parallel

8.1 \(B(\alpha, 1) = A(\alpha, 1)\)

8.2 \(A(\alpha, 1) = y\)

8.3 \(B(\alpha, 1) = A(\alpha, 1)/B(\alpha, 1)\)

Step 9. For all \(\alpha=1\ to \ n\) do in parallel Using tree connection along the first column sum up the contents \(B(\alpha, 1)\) and store the result in \(B(1, 1)\).

Step 10. \(B(1, 1) = B(1, 1) \ast D(1, 1)\)

Step 11. Stop

The final interpolated value emerges from the register \(B(1, 1)\).

IV. POLYNOMIAL ROOT FINDING
Consider the problem of finding all the roots of
\[ P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = 0 \]
an \( n \)-degree polynomial equation say, where, the coefficients \( a_i \)'s \( i = 0, 1, 2 \ldots n \) are assumed to be real.

Many software routines are available for finding the roots of a polynomial equation. The algorithms used in these library routines determine approximations to the polynomial roots by deflation method and hence provide very little scope for parallelization. However, there exist classes of algorithms, which are more suitable for parallelization. Durand-Kerner algorithm and Ehrlich algorithms are of this kind. In the recent years there has been reported several parallel algorithms based on Durand-Kerner and Ehrlich methods and can be found in the literature [9, 10, 11 and 12]. In these algorithms main attempt has been made to parallelize the computation involved in a single iteration. The iterative schemes to find the roots due to Durand-Kerner and Ehrlich are given as follows [9].

\[ x_j^{(k+1)} = x_j^{(k)} - \frac{P_j(x_j^{(k)})}{\sum_{i=1, i \neq j}^{n} x_i^{(k)}}, i = 1, 2, \ldots, n \]

Equation 4 represents Durand-Kerener

\[ x_j^{(k+1)} = x_j^{(k)} - \frac{1}{\sum_{i=1, i \neq j}^{n} \frac{1}{x_i^{(k)}}}, i = 1, 2, \ldots, n \]

Equation 5 represents Ehrlich Kerner

Where

\[ i^{(k)} = \frac{n}{\sum_{j=1, j \neq i}^{n} x_j^{(k)}} \]

\[ i^{(k)} = \frac{m}{\sum_{j=1, j \neq i}^{n} y_j^{(m)}} \]

The Durand-Kerner and Ehrlich methods are locally convergent with quadratic and cubic rate of convergence respectively [9]. We now present the parallel algorithm for a single iteration of the Durand-Kerner method. Because of the similarity of the computation, the parallel algorithm for Ehrlich method can similarly be developed. We use the following computational model.

4.1 Ring Computation Model

1. \( n \) processors are arranged to form a ring. The processors \( P_i \) is directly connected to the processor \( P_{i \mod n +1} \) for \( i = 1, 2 \ldots n \) by means of a bi-directional links.

2. Each processor \( P_i \), has four local registers namely \( A(i), B(i), C(i) \) and \( D(i) \). The registers \( A(i)'s \) and \( B(i)'s \) are used for the data communication in the clockwise and anti-clockwise respectively.

4.2 Second Proposed Algorithm

Step 1. For all \( \alpha = 1 \) to \( n \) do in parallel

\[ A(i) = x_i, B(i) = x_i \]
\[ C(i) = x_i, D(i) = 1 \]

**Step 2.** For \( k = 1 \) to \( n/2 - 1 \) do

For all \( \alpha = 1 \) to \( n \) do in parallel

\[ A(\alpha \MOD n +1) = A(\alpha) \]
\[ B(\alpha) = A(\alpha \MOD n +1) \]
\[ D(\alpha) = D(\alpha)^* [C(\alpha) - A(\alpha)]^* [C(\alpha) - \beta(\alpha)] \]

**Step 3.** For all \( \alpha = 1 \) to \( n \) do in parallel

\[ A(\alpha \MOD n +1) = A(\alpha) \]

If \((n \text{ is odd}) \) then

\[ B(\alpha) = A(\alpha \MOD n +1) \]
\[ D(\alpha) = D(\alpha)^* [C(\alpha) - A(\alpha)]^* [C(\alpha) - \beta(\alpha)] \]

Else

\[ D(\alpha) = D(\alpha)^* [C(\alpha) - A(\alpha)] \]

**Step 4.** For all \( \alpha = 1 \) to \( n \) do in parallel

\[ A(\alpha) = P_n(x) \]

**Step 5.** For all \( \alpha = 1 \) to \( n \) do in parallel

\[ D(\alpha) = C(\alpha) - A(\alpha)/D(\alpha) \]

**Step 6.** Stop.
It is easy to note that only step 2 requires $O(n)$ time and the rest of the steps require constant time. Hence the overall time complexity of the above algorithm is $O(n)$ time.

V. RESULT AND CONCLUSION

The algorithms for solving the problem have been implemented in Java with openMP directives. The experiments carried out on dual processor Quad-Core Xeon (3.2 GHz, 12 MB L2 cache, 32 GB RAM running under Linux) workstation using Intel Java compiler shows that new algorithm achieves much better performance than simple scalar algorithm. Table 2 shows exemplary results of the performance of the algorithm for various problem sizes and block sizes. The performance of the new parallel non-square algorithm is much better. The performance is better for the bigger problem size ($n=3097157$) and the algorithm achieve reasonable speedup. We have shown that the performance of proposed algorithms for solving financial data is highly improved by using parallel processing. As the calculation time complexity of large problems with various constraints and variables is significant, calculation of set of problems are performed in parallel.

Table 2: Exemplary results of the performance of proposed algorithm on dataset1

<table>
<thead>
<tr>
<th>Problem</th>
<th>Stages</th>
<th>Blk</th>
<th>Assets</th>
<th>Scenario</th>
<th>Constraints</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM1</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>11111</td>
<td>66.66</td>
<td>165</td>
</tr>
<tr>
<td>ALM2</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>112</td>
<td>666.76</td>
<td>154</td>
</tr>
<tr>
<td>ALM3</td>
<td>4</td>
<td>14</td>
<td>11</td>
<td>131</td>
<td>1222</td>
<td>345</td>
</tr>
<tr>
<td>ALM4</td>
<td>5</td>
<td>24</td>
<td>8</td>
<td>342</td>
<td>2.07</td>
<td>224</td>
</tr>
<tr>
<td>ALM5</td>
<td>6</td>
<td>31</td>
<td>11</td>
<td>214</td>
<td>2140</td>
<td>542</td>
</tr>
<tr>
<td>ALM6</td>
<td>7</td>
<td>22</td>
<td>10</td>
<td>266</td>
<td>3.41</td>
<td>251</td>
</tr>
<tr>
<td>ALM7</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>214</td>
<td>19.76</td>
<td>345</td>
</tr>
<tr>
<td>ALM8</td>
<td>4</td>
<td>39</td>
<td>11</td>
<td>212</td>
<td>11.67</td>
<td>211</td>
</tr>
<tr>
<td>ALM9</td>
<td>6</td>
<td>40</td>
<td>8</td>
<td>245</td>
<td>19.67</td>
<td>224</td>
</tr>
<tr>
<td>ALM10</td>
<td>4</td>
<td>75</td>
<td>11</td>
<td>288</td>
<td>19.45</td>
<td>345</td>
</tr>
</tbody>
</table>

Table 2: Exemplary results of the performance of proposed algorithm on dataset2

<table>
<thead>
<tr>
<th>Problem</th>
<th>Stages</th>
<th>Blk</th>
<th>Assets</th>
<th>Scenario</th>
<th>Constraints</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM1</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>11111</td>
<td>66.66</td>
<td>165</td>
</tr>
<tr>
<td>ALM2</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>112</td>
<td>666.76</td>
<td>154</td>
</tr>
<tr>
<td>ALM3</td>
<td>4</td>
<td>14</td>
<td>11</td>
<td>131</td>
<td>1222</td>
<td>345</td>
</tr>
<tr>
<td>ALM4</td>
<td>5</td>
<td>24</td>
<td>8</td>
<td>342</td>
<td>2.07</td>
<td>224</td>
</tr>
<tr>
<td>ALM5</td>
<td>6</td>
<td>31</td>
<td>11</td>
<td>214</td>
<td>2140</td>
<td>542</td>
</tr>
<tr>
<td>ALM6</td>
<td>7</td>
<td>22</td>
<td>10</td>
<td>266</td>
<td>3.41</td>
<td>251</td>
</tr>
<tr>
<td>ALM7</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>214</td>
<td>19.76</td>
<td>345</td>
</tr>
<tr>
<td>ALM8</td>
<td>4</td>
<td>39</td>
<td>11</td>
<td>212</td>
<td>11.67</td>
<td>211</td>
</tr>
<tr>
<td>ALM9</td>
<td>6</td>
<td>40</td>
<td>8</td>
<td>245</td>
<td>19.67</td>
<td>224</td>
</tr>
<tr>
<td>ALM10</td>
<td>4</td>
<td>75</td>
<td>11</td>
<td>288</td>
<td>19.45</td>
<td>345</td>
</tr>
</tbody>
</table>

Table 3: Exemplary results of the performance of proposed algorithm on dataset3.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Stages</th>
<th>Blk</th>
<th>Assets</th>
<th>Scenario</th>
<th>Constraints</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM1</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>11111</td>
<td>66.66</td>
<td>165</td>
</tr>
<tr>
<td>ALM2</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>112</td>
<td>666.76</td>
<td>154</td>
</tr>
<tr>
<td>ALM3</td>
<td>4</td>
<td>14</td>
<td>11</td>
<td>131</td>
<td>1222</td>
<td>345</td>
</tr>
<tr>
<td>ALM4</td>
<td>5</td>
<td>24</td>
<td>8</td>
<td>342</td>
<td>2.07</td>
<td>224</td>
</tr>
<tr>
<td>ALM5</td>
<td>6</td>
<td>31</td>
<td>11</td>
<td>214</td>
<td>2140</td>
<td>542</td>
</tr>
<tr>
<td>ALM6</td>
<td>7</td>
<td>22</td>
<td>10</td>
<td>266</td>
<td>3.41</td>
<td>251</td>
</tr>
<tr>
<td>ALM7</td>
<td>8</td>
<td>30</td>
<td>8</td>
<td>214</td>
<td>19.76</td>
<td>345</td>
</tr>
<tr>
<td>ALM8</td>
<td>4</td>
<td>39</td>
<td>11</td>
<td>212</td>
<td>11.67</td>
<td>211</td>
</tr>
<tr>
<td>ALM9</td>
<td>6</td>
<td>40</td>
<td>8</td>
<td>245</td>
<td>19.67</td>
<td>224</td>
</tr>
<tr>
<td>ALM10</td>
<td>4</td>
<td>75</td>
<td>11</td>
<td>288</td>
<td>19.45</td>
<td>345</td>
</tr>
</tbody>
</table>

In Table 2 priority is given to constraints, Table 3 priority is given to variables and Table 4 priority is given to assets. The classification accuracy is greatly affected by the number of variables and constraints of the selected dataset. Classical algorithms are not efficient in solving problems associated with large data sets as they use traditional solution methodologies. Parallel algorithms for polynomial interpolation and solving polynomial equation have been presented. The algorithm for polynomial
interpolation, based on n-point Lagrange formula has been shown to run in O(log n) time on a mesh of trees model of parallel computers. For solving polynomial equation of degree n, we have shown the parallelization of Durand-Kerner method on ring of n processors in O(log n) time.

REFERENCE